Universidade Federal da Paraíba –
Centro de Ciências e Tecnologia - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA – COPIN

Filtros Espaciais de Alto Desempenho para Imagens em Sistemas de Informações Geográficas

Rute Freitas Queiroz de Barros

Campina Grande – PB

Dezembro de 1998

Rute Freitas Queiroz de Barros

Filtros Espaciais de Alto Desempenho para Imagens em Sistemas de Informações Geográficas

Dissertação apresentada ao Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus II, em cumprimento às exigências parcais para obtenção do grau de Mestre.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sistemas de Software

Orientador: Marcelo Alves de Barros

Co-Orientador: Elmar Uwe Kurt Melcher

Campina Grande, Paraíba, Brasil
Dezembro de 1998

Barros, Rute Freitas Queiroz de

B277F

Filtros Espaciais de Alto Desempenho para Imagens em Sistemas de Informações Geográficas;

Dissertação apresentada ao Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus II, em cumprimento às exigências parciais para obtenção do grau de mestre;

Orientador: Marcelo Alves de Barros
Co – Orientador: Elmar Uwe Kurt Melcher

1. Processamento de Imagens de Satélite;
2. Processamento de Alto Desempenho;
3. Arquiteturas Reconfiguráveis.

CDU – 681.332.3*

FILTROS ESPACIAIS DE ALTO DESEMPENHO PARA IMAGENS EM SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

RUTE FREITAS QUEIROZ DE BARROS

DISSERTAÇÃO APROVADA EM 18.12.1998

PROF. MARCELO ALVES DE BARROS , Dr.
Orientador

PROF. ELMAR UWE KURT MELCHER, Dr.
Co-Orientador

PROF. JOÃO MARQUES DE CARVALHO, Ph.D
Examinador

HAMILTON SOARES DA SILVA, M.Sc
Examinador

CAMPINA GRANDE - PB

Dedico este trabalho as minhas queridas e amadas filhas que comigo entraram nesta batalha e muito me ajudaram:

Jacinta Barros (sua força foi fundamental);
Ana Tereza (suas observações foram sensacionais);
Helena Freitas (seus carinhos foram valiosos).

A nossa união, força, garra e determinação...

Resumo

Os filtros espaciais são aplicados a imagens de Sensoriamento Remoto em formato digital e visam a remoção de ruídos ou alguma forma de segmentação. A partir da investigação dos algoritmos de filtragem espacial e as suas exigências de esforço computacional, um algoritmo é selecionado e implementado utilizando uma plataforma de *hardware* reconfigurável com circuitos *FPGA (Field Programmable Gate Arrays)*. O algoritmo selecionado é o filtro da mediana vetorial reduzido (RVMF). Os resultados demostram que tarefas de filtragem espacial implementadas em arquiteturas reconfiguráveis são viáveis para resolver o problema da exigências de intenso processamento e alta flexibilidade.

Abstract

Spatial filters are used for digital images to remove noise or perform some form of segmentation. Based on studies of filter algorithms and their computational complexity, an algorithm is implemented using reconfigurable hardware consisting of FPGA's (Field Programmable Gate Arrays). The algorithm chosen is the reduced vector median filter (RVMF). Results show the viability of spatial filter implementations using reconfigurable hardware in order to resolve the complexity problem and maintain flexibility of systems.

AGRADECIMENTOS

Fui na lua buscar o que parecia tão distante,
tão difícil e muito muito impossível mas cheguei,
aportei, investiguei e descobri que poderia não so visitar a lua
como também poderia explorar toda a sua riqueza e apreciar tudo tudo
o que eu estivesse vontade de ver, bastava ter força de vontade,
garra e determinação ...

Sabe, aqueles momentos em que você se dispõe a optar por realizar um objetivo na vida?, e este objetivo as vezes embora seja difícil te faz a cada dia mais crescer e aprender?

Comigo o mestrado foi assim: uma aprendizagem constante, um desafio constante, uma luta constante e por fim tudo chegou ao fim e hoje estou aqui a escrever e agradecer a todos os que comigo participaram em prol de atingir este objetivo: "*o diploma de mestra*"!!!

- Aos meus pais: Hélio de Freitas e Tereza Dias pela força, amizade, compreensão e amor;
- Aos meus irmãos: Lúcia Freitas, José Tadeu e Hélio pelo carinho e as palavras de apoio;
- A Adamor Barbosa (pai e filho), D. Helena Maria, D. Hilda pelas pessoas maravilhosas que são, sempre perto da nossa família, incentivando o crescimento e a união;
- A Universidade Particular "CIESA", do estado do Amazonas que sempre acreditou no meu trabalho;
- Ao professor Luiz Antonio, pelo apoio e incentivo constante para a realização deste trabalho;
- Aos meus colegas manauaras: Lourdes, Idarclei, Sergio Mestrinho, Geraldo;
- Aos professores da COPIN pelos ensinamentos transmitidos: Hattori, Bruno, Walfredo, Jacques, Evandro, Marcus Sampaio, Antão, Edilson Ferneda, Bernardo, Agamemnon Fubica, Izabel;

- Ao professor Elmar pela atenção dedicada, disciplina, apoio, incentivo. Aprendi muito trabalhando com sua pessoa, **obrigada**;
- Ao professor Marcelo Barros pelas lições transmitidas;
- Aos professores e funcionários da COPIN e do DSC, pelas informações e atenção dispensadas: Lilian, Alberto, Manuela, Adalberto enfim a todos vocês;
- A Aninha, Vera, Zeneide pela atenção e incentivo;
- Aos professores: Evandro e Jair da Universidade Federal de Alagoas, que tornaram possível o seminário sobre meu trabalho, levantando indagações e muitas contribuições;
- Ao professor Alejandro Freiry da Universidade de Recife;
- A todos os professores do Instituto Nacional de Pesquisas Espaciais, que me enviaram informações e artigos para a concretização deste trabalho;
- Aos professores do laboratório de Sensoriamento Remoto: Eustáquio, Edilberto, Lúcia e Ianna, que sempre, sempre tiveram a paciência de discutir minhas indagações, e principalmente pelo carinho dedicado;
- Ao meu colega Luciano Reis pelo apoio, amizade, e paciência em transmitir seus conhecimentos de programação na linguagem C;
- Aos meus colegas de mestrado, pelas horas de estudo, pelo apoio nas horas difíceis e amizade: Zane, Vera, Milena, Roberta, Patrícia, Malty, Giovanni, Josenildo, Marcão, Jean, Claudionor, Tatiana, Guga, Isabel, Maria Luiza, Adriana, Soninha, Helmut;
- Aos colegas paraibanos: Janete, Hélio, Josineide, Wilson, Walter, Carlos e Socorro;
- À CAPES e CNPQ, pelo suporte financeiro oferecido durante o período do trabalho de mestrado.

Lista de Figuras

SUMÁRIO

Lista de Tabelas

Lista de Abreviaturas

SIG – Sistemas de Informações Geográficas

PDI – Processamento Digital de Imagens

PI – Processamento de Imagens

SR – Sensoriamento Remoto

TM – Thematic Mapper

ERS-1 – European Remoting Sensing

AVIRIS – Airbone Visible Infrared Imaging Spectrometer

AVHRR – Advanced Very High Resolution Radiometer

FPGA – Field Programmable Gate Array

ROM – Filtro da Seqüência da Mediana Ordenada

PE – Filtro da Permutação Estendida

Capítulo 1
Introdução

1.1 Apresentação

Os Sistemas de Informações Geográficas (SIG) consistem em uma base computacional com dados integrados de forma a tornar possível a coleta, o armazenamento, o processamento e a manipulação de dados georreferenciados[1], tais como aqueles presentes em mapas, cartas, imagens de satélite, fotografias e outras formas de informação georreferenciada **[Tei, 95] [Barros, 97]**.

Diversas ferramentas destinadas ao processamento de informação geográfica (ER MAPPER, ENVI, SPRING, IDRISI) utilizam as tecnologias de Sensoriamento Remoto (SR) como meio de obter os seus dados de entrada. As imagens digitais adquiridas por Sensoriamento Remoto servem como base de entrada dos objetos georreferenciados para mapeamento, monitoramento, modelagem e mensuração das informações manuseadas pelos SIG. Dentre estas tecnologias se destaca a dos sensores orbitais, para aquisição de imagens de cenas da superfície terrestre (imagens de satélite). Essas imagens podem ser monoespectrais (imagens formadas pela composição de uma única banda) ou multiespectrais (imagens formadas pela composição de várias bandas[2]).

Para análise das imagens de satélite são empregadas, de forma integrada, técnicas de processamento de imagens e de análise espacial de uma base cartográfica[3]específica possibilitando assim a associação de informações textuais, na forma de tabelas de bancos de dados, a informações georreferrenciadas de interesse do usuário. Tais informações permitem ao usuário dispor de meios necessários para a interpretação e análise de dados facilitando, assim, o mapeamento e análise de

---

[1] Dados georeferenciados são dados geograficamente referenciados a um sistema oficial de coordenadas geográficas;

[2] Os sistemas de SR existentes adquirem imagens em intervalos do espectro eletromagnético, os quais são denominados de bandas ou canais.

[3] A base cartográfica permite que um dado georreferenciado qualquer (como uma cidade, montanha, etc..) seja localizado em relação a outros dados através de suas posições previamente conhecidas, ou através de um sistema de coordenadas pré-definido.

fenômenos e eventos que ocorrem na superfície terrestre. Tal estratégia torna viável o planejamento e tomadas de decisão relativos a eventos os mais diversos, incluindo por exemplo, as calamidades do mundo moderno (poluição, desflorestamento, desastres naturais, etc), decisões de aplicações de verbas (seleção do melhor solo para plantio de determinada cultura, melhor trajetória de percurso para distribuição de produtos comerciais, etc), projeto de expansão de redes de telefonia ou de distribuição de energia e gerenciamento de serviços de infraestrutura urbana.

Na maioria das aplicações envolvendo geoprocessamento, a informação espacial está contida em imagens do tipo «raster» (na forma de matrizes de pontos ou pixels), obtidas principalmente por sensores orbitais. Algumas propriedades básicas destes sensores orbitais são: a resolução espectral (dada pelo número de bandas do espectro eletromagnético contidas nas imagens), a resolução radiométrica (dada pela capacidade do sensor de diferenciar as variações na intensidade da energia coletada) e a resolução espacial (área da superfície terrestre observada instantaneamente por cada sensor) **[Hemerly, 96]**.

Os recentes avanços da tecnologia de aquisição de imagens através das técnicas de SR tem produzido sensores orbitais capazes de gerar imagens digitais contendo grandes volumes de informações. Para se ter uma idéia da quantidade de dados contidos nestas imagens foram elaboradas as Tabelas 1.1 e 1.2. A Tabela 1.1 ilustra o volume de dados gerados pelos sensores orbitais dos novos satélites. A Tabela 1.2 indica a relação dos futuros satélites e as estimativas da quantidade de pixels por cena.

Tabela 1.1: Volume de dados que serão gerados pelos sensores orbitais dos novos satélites:

Nome do satélite	Previsão lançamento	Volume de dados por dia
EOS AM - 1	Jun 1998	862 Gbytes
TRMM	Agos 1997	12 Gbytes
PM – 1	Dez 2000	549 Gbytes
ADEOS II	Fev – 1999	4 Gbytes

Fonte: EOS DATA PRODUCTS / Nasa.gov

Tabela 1.2 Futuros Satélites – Características das Imagens geradas por satélites no modo Pancromático[42]

Nome do satélite	Resolução (metros)	Tamanho de cada cena Imageada (Km)	Quantidade de Pixel por cena (aproximadamente)	Periocidade
LANDSAT 7	15	185 × 185	1 Gbyte	16 dias
SPOT 4	10	60 × 60	180 Mbyte	26 dias
EARLY BIRD	3	6 × 6	4 Mbytes	20 dias
SPACE IMAGING	1	60 × 60	3 Gbytes	14 dias
QUICK BIRD	1	6 × 6	36 Mbytes	20 dias
CBERS	20	120 × 120	360 Mbytes	26 dias

Fonte: Fator Gis Anuário 1997/ GeoPlace.com/ Nasa – EOS

A maioria das imagens que serão obtidas por estes novos satélites são imagens multidimensionais, ou seja, compostas por vários planos ou bandas (matrizes superpostas) correspondentes a imagens monoespectrais, obtidas em faixas específicas do espectro eletro magnético. A fim de possibilitar um aumento de precisão na análise espacial realizada pelas ferramentas SIG, as imagens multidimensionais no formato digital necessitam, freqüentemente, serem submetidas a algum tratamento preliminar (pré-processamento) para a remoção de ruídos e/ou para a produção de alguma forma de realce nas características dos objetos estudados da imagem (contornos, regiões, textura, padrões, etc). Este pré-processamento é realizado principalmente através da aplicação de filtros espaciais.

1.2 Os Algoritmos de Filtragem Espacial

A aplicação dos algoritmos de filtragem espacial nas imagens de satélite realça as correlações existentes entre os pixels vizinhos, facilitando o processo de interpretação da imagem.

Os ruídos presentes nas imagens de satélite (digitais) apresentam-se como deformações em que os pixels ruidosos são representados por pontos que possuem valores de níveis de luminosidade bem diferentes dos valores dos pixels vizinhos. Estes pontos (os ruidosos) podem estar distribuídos aleatoriamente ou de forma sistemática (linhas verticias e horizontais), sobre toda a imagem.

[4] As imagens geradas pelos satélites no modo Pancromático são obtidas por filmes preto e branco, sensível à energia com comprimento de onda entre 0,30 e 0,70 μm [Crósta, 97].

As técnicas de filtragem espacial podem ser consideradas como uma operação que substitui os valores de níveis de cinza ou cores dos pixels da imagem de entrada por novos valores. Esses novos valores levam em conta a correlação com os valores dos pixels vizinhos na imagem. As tarefas de filtragem espacial para remoção de ruído devem preservar as regiões de transição entre regiões homogêneas.

1.3 Quantidade de Dados nas Imagens digitais e Aplicação das Técnicas de Filtragem Espacial

As tarefas de filtragem espacial para as imagens de satélite usualmente requerem muito esforço computacional. Uma das razões para tanto é a quantidade de pontos que necessitam ser filtrados (bilhões de pixels por imagem), uma vez que aos algoritmos estão associados inúmeras operações matemáticas por pixel processado.

Para se aplicar a operação de filtragem espacial é preciso acessar todos os pixels da imagem de entrada para gerar a imagem de saída. Para cada pixel da imagem de entrada o algoritmo de filtragem espacial produzirá um pixel da imagem de saída, cujo valor é determinado como uma função do valor do pixel de entrada e do valor dos pixels de sua vizinhança.

Uma conseqüência da alta densidade de dados e das operações envolvidas nestas imagens é que elas passam a exigir cada vez mais um grande desempenho dos computadores assim como o emprego de técnicas avançadas de processamento, tanto em nível de *hardware* quanto de *software*.

1.4 Implementação dos Algoritmos de Filtragem espacial

Os algoritmos de filtragem espacial podem ser seqüenciais ou paralelos. A estratégia mais usada na implementação dos algoritmos seqüenciais é *software* e nos algoritmos paralelos é *hardware* dedicado (*ASICs)* ou máquinas paralelas programáveis.

Apesar dos algoritmos seqüenciais serem os mais simples de implementar para este tipo de operação eles não são os mais indicados. Nas imagens de alta resolução a quantidade de operações normalmente necessárias para o cálculo dos pixels de saída torna proibitivo o tempo de processamento quando os operadores são implementados

segundo o conceito de instrução em máquina de uso geral do tipo ISP (*Instruction Set Processors)* **[DeMichelli, 97].** Este é o caso de aplicações envolvendo imagens de alta resolução, mesmo em situações de processamento off-line.

Nas implementações baseadas em arquiteturas paralelas, projetadas para intercalar leitura e escrita de memória, existe uma latência relativamente pequena de processamento entre a imagem de entrada e a imagem de saída. Esse modelo de cálculo e implementação é denominado de "*data flow*", e pode ser altamente otimizado quando implementado em *hardware*. Neste modelo o desempenho e a capacidade de armazenamento dos dados para realizar os cálculos dependem diretamente do tamanho da janela usada.

1.5 As Arquiteturas Reconfiguráveis

Os circuitos reconfiguráveis são circuitos integrados que podem ser configurados e reconfigurados, em que os operadores dedicados (específicos para determinado fim) são definidos no software.

As arquiteturas reconfiguráveis compartilham todas as características de uma arquitetura de propósito geral mas de modo diferente. As funções são computadas através das unidades funcionais de configuração instalando-as na área disponível. Apresentam a vantagem de fornecer maior aproveitamento da unidade de área que as arquiteturas convencionais.

1.6 *Field Programmable Gate Arrays* – FPGA

Os circuitos *FPGA* geralmente consistem de uma matriz de células lógicas flexíveis relativamente simples, registradores e memórias, cujas funções e interconexões são controladas através do carregamento de bits de memória de configuração **[Oldifield, 95].** A configuração do circuito FPGA é realizada através do carregamento de um arquivo binário (*bit stream*) em um arranjo de células de memória estática, cujo conteúdo (0 ou 1) define o estado de um elemento reconfigurável do FPGA ou de parte deste elemento. Os circuitos FPGA são ricos em registros facilitando assim o uso de técnicas de pipeline para aumentar a taxa de processamento do sistema. As tecnologias de programação, comercialmente disponíveis, nos circuitos *FPGA* são: *RAM estática, EPROM, EEPROM, FUSÍVEL e ANTIFUSÍVEL.*

Os circuitos FPGA reprogramáveis de tecnologia RAM, EPROM e EEPROM suportam múltiplas reconfigurações do hardware, isto é, a configuração do *hardware* pode ser aplicada mais de uma vez a um mesmo sistema físico construídos com estes tipos de circuitos. A reconfigurabilidade de um sistema aumenta sua usabilidade, explora a flexibilidade e suporta mudanças de funções.

1.7 A Metodologia Hardware/Software Codesign

Nos últimos anos o avanço no desenvolvimento de circuitos reconfiguráveis (ex: *FPGA*), permitiu a implementação de algoritmos de processamento de imagens em arquiteturas dedicadas reconfiguráveis. O interesse em aplicar a metodologia *hardware/software codesign* nas tarefas de processamento de imagens provêm do fato desta metodologia fornecer critérios para atenuar o contínuo aumento de complexidade que estas tarefas exigem.

O propósito da metodologia *hardware/software codesign* é apresentar uma otimização no projeto de construção dos componentes de *hardware* e de *software*. A meta é realçar o desempenho das tarefas computacionalmente intensas usando para tal as informações extraídas do programa fonte para posteriormente sintetizá-las e com isto aumentar a funcionalidade da arquitetura.

Como os recursos do *hardware* reconfigurável são limitados, as técnicas de *codesign* são necessárias para selecionar quais as partes do algoritmo que ao serem implementadas através dos circuitos dedicados resultam numa aceleração na execução do *software* **[Michelli, 97]**.

O particionamento do sistema em *hardware* e *software* é de importância crítica pois tem um grande impacto na relação custo/desempenho do projeto final. Qualquer decisão de particionamento realizada ou por uma ferramenta CAD ou pelo projetista deve levar em conta as propriedades resultantes dos blocos que serão implementados em *software* e em *hardware*. As etapas executadas em *hardware* são aquelas que restringem o tempo de processamento quando todo o sistema é implementado em *software*. Outros critérios para se decidir quais as partes do algoritmo devem ser implementados em *hardware* são:

- tempo de projeto gasto para o desenvolvimento da implementação em

hardware;

- disponibilidade de descrições para *hardware* já prontas e reutilizáveis.

Segundo **[Micheli, 97]** o *codesign* é realizado em três etapas: *modelagem* (nesta etapa o algoritmo é definido e especificado), *implementação* (implementação dos componentes de *hardware* e de *software*) e a etapa de *validação* a nível de sistema significa ganhar uma certeza razoável que o projeto está livre de erros e portanto funciona adequadamente tanto em *hardware* como em *software*. A validação pode ser realizada por meio de verificação formal, simulação ou emulação. A ferramenta de verificação checa a congruência da representação do projeto e/ou tenta provar uma propriedade específica. A simulação é um método tradicional de validar o *hardware* corretamente, examinando um conjunto de respostas a estímulos de entrada. Na figura 1.2 é apresentada uma descrição da metodologia *hardware/software codesign* para sistemas reconfiguráveis.

1.7.1 A metodologia Codesign para o mapeamento de FPGA reprogramável

O procedimento usado para implementar um algoritmo em um *FPGA* reprogramável tem inicio com a especificação comportamental do sistema como um todo. Nesta fase, estuda-se o desempenho global do sistema, o custo, a testabilidade e a interação de sub-sistemas que se adequam melhor se implementados em *hardware*.

A descrição do comportamento das partes do sistema que será implementada em *hardware* é modelada usando uma linguagem de descrição de *hardware* (*HDL*).

Esta descrição comportamental é então processada por uma ferramenta de síntese lógica. A síntese lógica a nível transferência de registro, que é realizada a partir da descrição funcional, produz uma lista de elementos lógicos e interconexões (*netlist*), e é um processo semi-automático que nem sempre produz resultados satisfatórios **[Morais, 98]**.

Após a síntese, o *netlist* obtido é submetido à ferramenta de mapeamento para alocar os recursos do *FPGA*. Na etapa final, a ferramenta realiza o roteamento escolhendo os *switches* programáveis para estabelecer as conexões requisitadas entre os blocos lógicos.

Depois que a etapa de *roteamento* for completada com sucesso, é gerado o *bit stream* necessário para a programação dos dispositivos que é colocado em uma unidade de programação. Essa unidade de programação configura o *FPGA*.

Depois que for validada a funcionalidade do sistema a implementação do circuito está completa e pronta para ser usada.

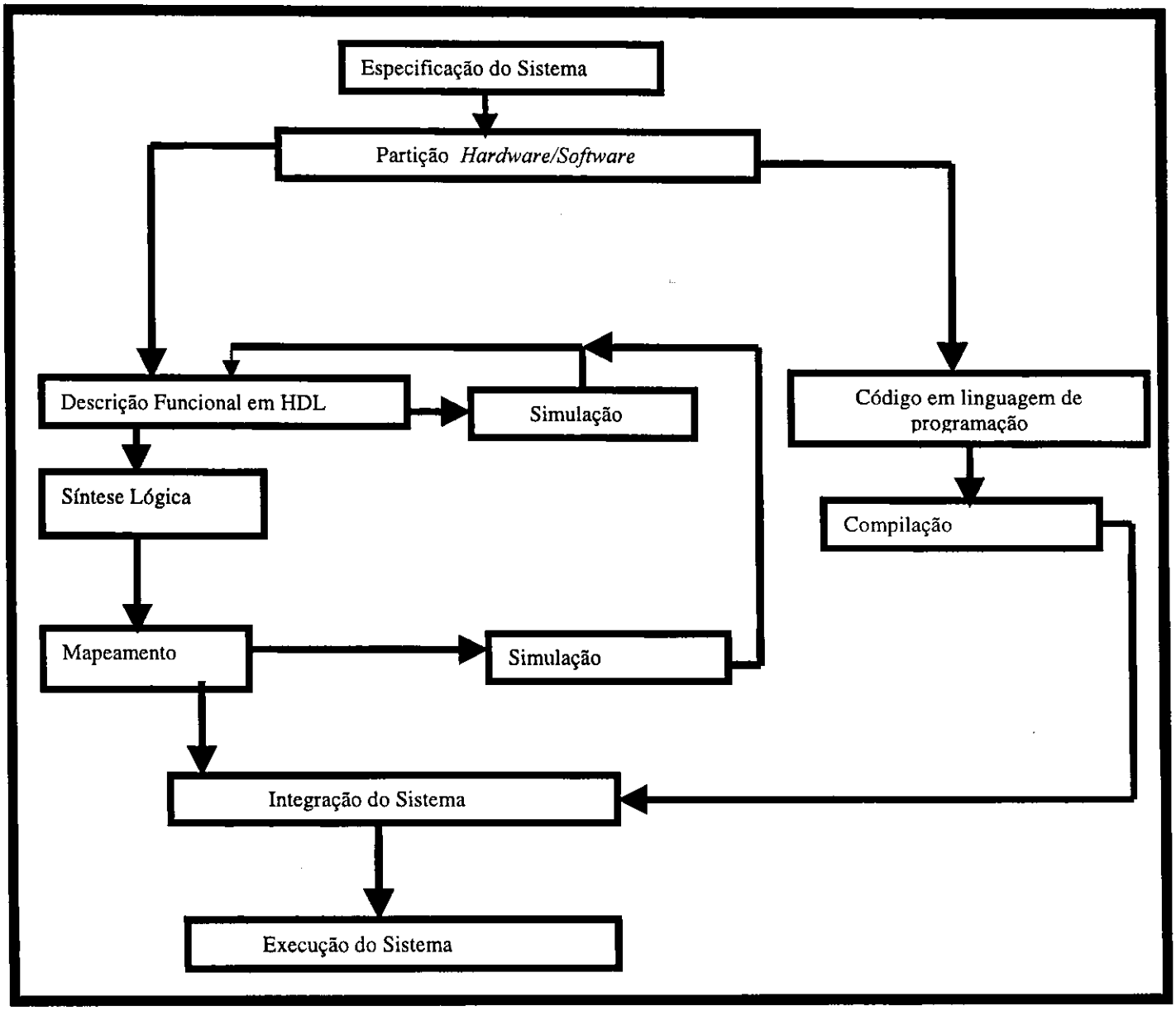


Figura 1.1: *Hardware/software codesign* para Sistemas Reconfiguráveis

1.8 Motivação

A importância das tarefas de filtragem espacial na análise das imagens de satélites é muito grande, pois essa fase de pré-processamento prepara a imagem para ser

segmentada e classificada. É portanto fundamental que as ferramentas SIG disponíveis no mercado ofereçam eficientes ferramentas de filtragem espacial.

A maioria dos recursos de filtragem espacial disponíveis nas ferramentas SIG representativas do mercado atual baseiam-se em algoritmos lineares ou não-lineares relativamente simples, os quais são adequados para uma implementação baseada unicamente no conceito de instrução ("implementação em *software*") e ao processamento das tarefas de Filtragem Espacial nas bandas isoladas da imagem orbital considerada. O processo de filtragem das imagens multidimensionais quando é realizado separadamente sobre cada banda que compõe a imagem, compromete a correlação existente entre as diferentes bandas e resulta em possíveis perdas de informações **[Astola, 90].**

Os algoritmos de Filtragem espacial são normalmente dedicados para o Processamento Digital de Imagens em Baixo Nível e se caracterizam pela exigência de uma grande quantidade de cálculo. A quantidade de cálculo é função do tamanho da janela, número de operações por pixel. Eles são geralmente definidos a partir de operações locais, envolvendo uma vizinhança preestabelecida (janela ou máscara). Essas operações, normalmente baseadas na convolução bidimensional e/ou em transformações lineares e não-lineares, são usadas para construir filtros passa-baixas e passa-altas, detetores de contornos, filtros da ordem, operadores morfológicos, etc. Estas tarefas de pré-processamento, em função dos requisitos de velocidade, em particular nas situações de processamento em tempo real ou de tratamento de imagens de alta densidade, exigem o projeto de arquiteturas paralelas dedicadas, freqüentemente usando um modelo arquitetural SIMD (Single Instruction Multiple Data), e a implementação de processadores elementares sob a forma de circuitos integrados dedicados (ASICs - *Application Specific Integrated Circuits*) **[Barros92a], [Villeumin96] [Barros98].**

A tendência dos algoritmos de filtragem espacial para as imagens orbitais dos futuros satélites aponta para uma maior complexidade computacional. A necessidade de maior desempenho (tempo de processamento) a baixos custos, e também a necessidade da disponibilização destas técnicas nas ferramentas SIG comerciais serviu de motivação ao presente trabalho. A motivação maior é a propor uma estratégia de implementação para os algoritmos de filtragem espacial destinadas as imagens orbitais de alta resolução gerados pelos novos satélites. Esta estratégia será baseada em arquiteturas

reconfiguráveis implementadas com circuitos *Field Programmable Gate Arrays (FPGA)* através da metodologia *hardware software codesign.*

O uso de circuitos FPGA como solução tecnológica adequada para a implementação física de Filtros Espaciais foi demonstrada em **[Barros94b] e [Barros93b]**.

A aplicações das técnicas de implementação híbrida de filtros espaciais para imagens mono-canais tem sido muito utilizados nos últimos anos, demonstrando assim o benefício do emprego de tecnologias de circuitos reconfiguráveis, quando condicionado a um esforço de adequação entre o algoritmo, o modelo arquitetural e a tecnologia empregada na implementação física **[Barros, 92a], [Barros, 92b], [Barros, 93a], [Barros, 93b], [Barros, 94a], [Barros, 94b], [Barros, 94c], [Leite, 94], [Athanas, 95], [Lulich, 96], [Rencher, 96].**

Uma metodologia de implementação híbrida para filtros espaciais com base em metodologia de *Hardware/Software Codesign* associada à implementação com o uso de circuitos reconfiguráveis FPGA (*Field Programmable Gate Arrays*) foi proposta em **[Barros96]**. Esta abordagem considera as características dos algoritmos de processamento digital de imagens em baixo nível e as particularidades envolvendo sua implementação sob a forma de arquiteturas dedicadas, construídas em plataformas de circuitos reconfiguráveis do tipo FPGA.

Neste trabalho, serão empregados os conceitos demonstrados e serão explorados os benefícios do uso de tecnologia de circuitos reconfiguráveis FPGA no estudo da implementação física de um algoritmo adequado para a filtragem de imagens de Sensoriamento Remoto. Para isto, além do estudo das particularidades das imagens de Sensoriamento Remoto de alta densidade e de aspectos ligados à sua implementação, as características dos algoritmos vetoriais para processamento digital de imagens multi-espectrais e as particularidades de uma estratégia específica de implementação de hardware são considerados na busca por uma boa relação custo-desempenho de ferramentas de filtragem espacial.

Com o emprego das arquiteturas reconfiguráveis (implementadas usando placas com *FPGA* inseridas em um computador hospedeiro), visa-se a operacionalização da implementação das técnicas de filtragem espacial para imagens de alta resolução nas ferramentas SIG. Visa-se também demonstrar a viabilidade desta solução em termos de

custo/desempenho com alta flexibilidade e com possibilidade de acoplamento a computadores pessoais e estações de trabalho.

1.9 Contribuição

A maior contribuição esperada com a execução deste trabalho deverá ser a de gerar informações e dados destinados ao suporte de ações práticas de projeto, implementação e/ou otimização de uma ferramenta de filtragem espacial de alto desempenho para imagens em ambientes de SIG, adequada aos avanços da tecnologia de Sensoriamento Remoto.

O trabalho também servirá de subsídio para a especificação e implementação de um algoritmo de filtragem espacial para o processamento digital de imagens de alto desempenho, utilizando plataformas de hardware reconfiguráveis construídas com circuitos *FPGA (Field Programmable Gate Arrays)*. Com isto, visa-se demostrar o interesse e a viabilidade do *Codesign* usando a tecnologia *FPGA* para solução do problema estudado.

1.10 Organização da Dissertação

No capítulo 2 é realizada uma breve descrição dos algoritmos de filtragem espacial disponíveis nas ferramentas SIG comerciais, assim como alguns citados recentemente na literatura especializada (período 96/97).

No capítulo 3 é apresentada uma análise das operações usadas nos algoritmos dos filtros espaciais, com o objetivo de mensurar o esforço computacional dos algoritmos dos filtros espaciais descritos no capítulo 2.

No capítulo 4 é apresentada a estratégia de implementação do algoritmo da mediana vetorial reduzido RVMF na linguagem de programação *C* e na linguagem de descrição de hardware *Verilog* e a metodologia *software/hardware codesign*.

No capítulo 5 são apresentadas as conclusões do trabalho com as perspectivas para trabalhos futuros.

sui pouca ou nenhuma informação do ruído).

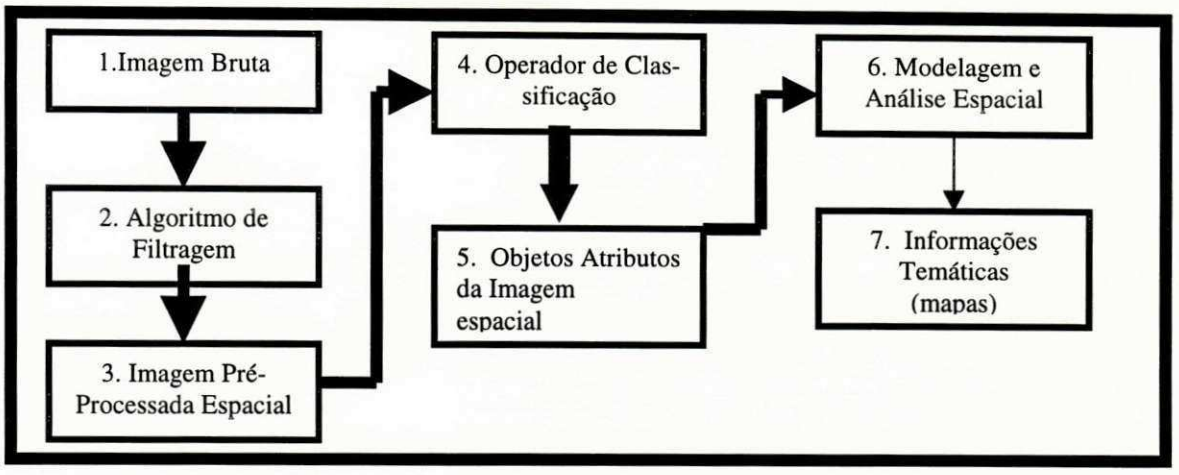


Figura 2.1: Cadeia de Filtragem Espacial para Imagens Multiespectrais

As janelas de filtragem espacial podem apresentar diferentes formatos (triangular, pentagonal, cruciforme, retangular ou quadrado), dependendo do algoritmo usado. Alguns algoritmos admitem que se modifique o formato da máscara, enquanto outros são projetados com um tipo de máscara específico. A Figura 2.2 apresenta resultados de diferentes técnicas de filtragem implementadas no IDRISI **[IDRISI, 96]** cada uma das quais produzindo diferentes resultados visuais.

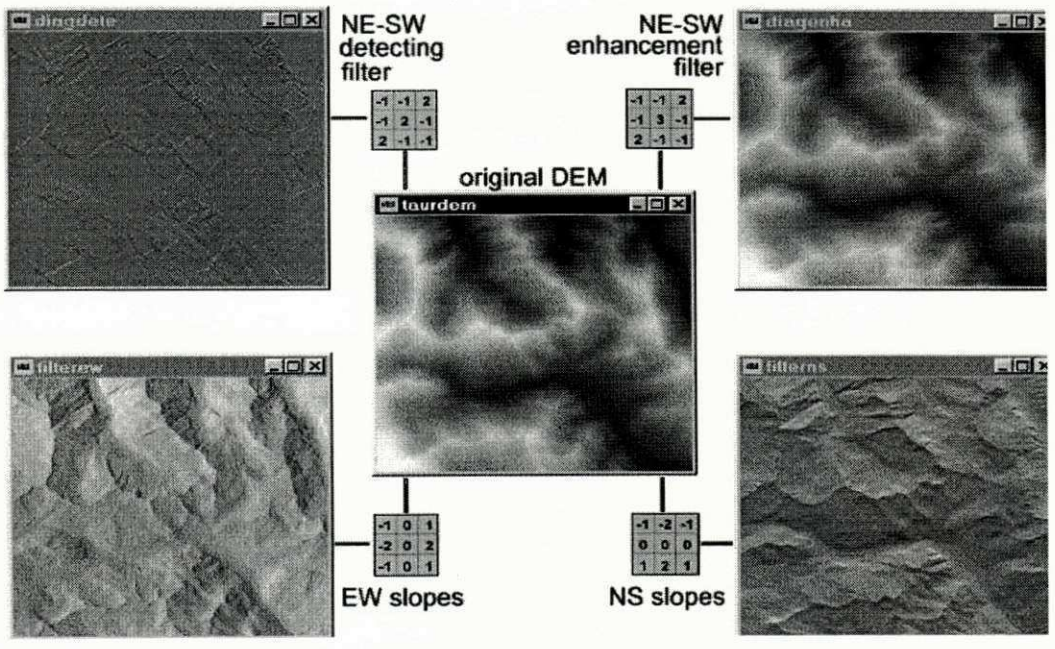


Figura 2.2: Exemplo de filtragem realizada pela ferramenta IDRISI

É de fundamental importância a presença de variados modelos de algoritmos de filtragem espacial nas ferramentas de processamento de imagens. Quanto maior o número de

Capítulo 2
Filtros Espaciais para Imagens de Sensoriamento Remoto

O propósito deste capítulo é apresentar um conjunto representativo de algoritmos de filtragem espacial aplicados em imagens de Sensoriamento Remoto. Estes algoritmos foram revisados através de um estudo envolvendo os recursos de processamento de imagens existentes em algumas ferramentas de processamento de informações geográficas (SIG), assim como através da leitura de artigos especializados.

2.1 Filtragem Espacial: Importância e Características

As imagens digitais manipuladas pelas ferramentas SIG são obtidas principalmente por sensores orbitais dos satélites tais como: Landsat, Spot, ERS-1, NOOA, JERS-1 ou similares. Essas imagens digitais que são manipuladas, processadas e transformadas em informações sobre a superfície terrestre geralmente podem ser corrompidas por ruídos independentes do sinal (aditivo), ruídos dependentes do sinal (multiplicativo) e ruídos impulsivos. Os ruídos independentes do sinal constituem um conjunto aleatório de níveis de cinza ou de cores adicionados aos pixels da imagem. Nos ruídos dependentes do sinal o valor do nível de cinza ou de cor em cada pixel da imagem é obtido em função do nível de cinza ou cor originais naquele pixel. As imagens corrompidas por ruídos impulsivos podem ser tratadas apenas por filtros não-lineares **[Berstein, 87]**. Esses ruídos são eliminados na etapa de pré-processamento através da aplicação de filtros espaciais. Na Figura 2.1 é apresentada o procedimento adotado pelas tarefas de filtragem espacial nas imagens multiespectrais.

Os filtros espaciais atuam diretamente nos pixels da imagem através da janela que se desloca sobre toda a imagem, efetuando operações lineares, não lineares ou adaptativas, baseadas em informações estatísticas localizadas. O comportamento das operações pode ser fixo (quando se conhece o comportamento do sinal e do ruído) ou adaptativo (os parâmetros são ajustados automaticamente durante o processo de filtragem pois neste caso se pos-

algoritmos disponíveis nestas ferramentas mais indicadas elas se tornam para auxiliar no tratamento de ruídos presentes nas imagens de satélite, assim como em aplicações para realce de bordas, restauração e pré-classificação de imagens em geral.

2.2 Descrição dos Algoritmos de Filtragem Espacial

No processamento das tarefas de filtragem espacial para as imagens de satélite a escolha de uma determinada técnica a ser usada depende dos fins específicos da aplicação envolvida. Deve ser considerado que estas tarefas apresentam comportamento similares em termos de processamento a baixo nível, onde a imagem é apresentada na forma de matriz **[Lulich, 96]**. Nessas matrizes são realizadas intensas operações matemáticas e/ou estatísticas, como pode ser observado na descrição dos algoritmos relacionados nesta seção, e o tempo de processamento destes filtros vai depender das características do algoritmo usado assim como dos recursos da arquitetura selecionada.

Esta seção apresenta um conjunto representativos de algoritmos de filtragem espacial que podem ser utilizados nas imagens de satélites multiespectrais e de radar. Conforme pode ser verificado são usadas diferentes operações matemáticas e estatísticas nestes algoritmos com o propósito de suavização, realce de bordas e eliminação do ruído *speckle* em imagens de radar.

Em **[Barros, 90]** foi realizado um estudo comparativo, seguido de implementação em *software* com aplicações no processamento de imagens *LANDSAT* para SIG. Neste trabalho o autor evidencia a importância dos filtros espaciais na extração de atributos espaciais, bem como demonstra a necessidade de desenvolver métodos de filtragem que considerem a correlação entre as diferentes faixas espectrais da imagem (banda). A busca desta correlação implica normalmente no aumento de complexidade do filtro e consequentemente na busca de uma abordagem eficiente de implementação.

Os algoritmos apresentados a seguir foram separados e classificados para serem empregados em imagens multiespectral e imagens de radar.

2.2.1 Algoritmos para Filtragem Espacial

A. Filtros Lineares:

1. Filtro da Média;

B. Filtros Não-Lineares:

2. Filtro da Mediana;
3. Filtro da Mediana Central Ponderada;
4. Filtro da Moda;
5. Filtros da Permutação Estendida;
6. Filtro ROM;

C. Filtros Operadores de Borda:

7. Operadores Diferenciais;
8. Operadores Direcionais;

D . Filtros Morfológicos:

9. Filtro Morfológico de Erosão;
10. Filtro Morfológico de Dilatação;

E. Filtros Adaptativos:

11. Filtro Racional
12. Filtro para realce usando lógica Fuzzy;
13. Filtro L Mínimo Quadrado Médio (LMS);

F. O Método Vetorial e os Filtros para as Imagens Multiespectrais:

14. Filtro da Mediana Vetorial;
15. Filtro da Mediana Vetorial Reduzido;
16. Filtros Adapitativos não Lineares para Múltiplos Canais;

G. Algoritmos para filtragem de imagens de Radar

1. Filtro de Estatística Local;
2. Filtro Sigma;
3. Filtro Sigma Ponderado;
4. Filtro Sigma Modificado;

2.2.2 Algoritmos para Filtragem de Imagens Multiespectrais

As imagens digitais multiespectrais são formadas pela composição de várias imagens (denominadas imagens monoespectrais), cada uma representando uma faixa espectral distinta de uma mesma cena da imagem **[Novo, 93]**. Essas imagens podem ser consideradas como uma extensão das imagens coloridas **[Candeiras, 97]**, pois a sua visualização no espaço RGB é usada empregando no máximo *3* bandas.

Os algoritmos a seguir são usados em imagens multiespectrais, sendo que a operação de filtragem é aplicada em cada banda separadamente, ou seja, nas imagens coloridas os algoritmos de filtragem são aplicados nas componentes escalares de cada banda *R*, *G* e *B* e posteriormente agrupadas para efeitos de visualização.

A. Filtros Lineares:

A característica dos filtros lineares é que estes algoritmos efetuam a soma do produto entre os coeficientes da máscara e a intensidade dos pixels sobre a máscara numa determinada posição da imagem.

1. O *filtro da média* **[Pratt, 78]** é um procedimento linear de suavização espacial e apresenta um ótimo desempenho na eliminação do ruído gaussiano. Embora consiga atenuar o ruído, tal procedimento apresenta bons resultados apenas em regiões homogêneas, por conta da degradação das bordas.

B. Filtros Não-Lineares:

Os filtros não lineares também operam na vizinhança dos pixels. Mas em geral essas operações são realizadas diretamente nos valores do pixels sobre a vizinhança considerada. Apresentam melhor desempenho que os filtros lineares na remoção dos ruídos, sem distorcer as características da imagem.

2. O *filtro da mediana* é um procedimento não linear usado com muito sucesso no processamento de sinais unidimensionais e bidimensionais. Como definido em **[Heygster, 82]** este filtro é considerado um filtro de ordem estatística sendo empregado para eliminação de ruído tanto em imagens de radar como em imagens multiespectrais. Seu algo-

ritmo serve de base para vários outros tipos de filtros tais como: filtro da mediana vetorial (VMF), filtro da mediana vetorial reduzido (RVMF), filtro da mediana adaptativa, filtro da mediana cruzada, e filtro da mediana ponderada, dentre outros.

3. O *filtro da mediana Central Ponderada* **[Lee, 91]** é caracterizado por atribuir um peso ao valor do pixel central da janela. Este filtro é uma extensão do filtro da mediana e apresenta a capacidade de realçar imagens corrompidas por ruídos dependentes ou independentes do sinal, tais como impulsivos e aditivos brancos. É simples de ser projetado e implementado, além de preservar detalhes da imagem filtrada.

4. O *filtro da moda* **[Gomes, 94]** é um filtro não-linear e atua de forma análoga ao filtro da mediana. Depois que os pixels de uma vizinhança são ordenados, o pixel central é substituído pelo valor que ocorre com maior freqüência, ou seja, a moda dessa vizinhança.

5. O *filtro de Permutação Estendida* (PE) **[Hardie, 96]** é caracterizado como um filtro que realça bordas, baseado em propriedades obtidas pelo seqüênciamento ordenado dos pixels pertencentes a janelas da imagem. Este filtro apresenta a característica de realizar o processo de filtragem usando amostras de observação da imagem. Através destas amostras, são construídos vetores de observação estendidos compostos de N amostras de observação (N depende do tamanho da máscara do filtro) e K operações estatísticas obtidas em função destas amostras. O resultado da filtragem é obtido pela restrição de uma ordem estatística no conjunto de observação.

A cada posição n, as N amostras de observação que são atravessadas pela máscara de filtragem são indexadas e escritas como um vetor $x(n)$:

$$x(n) = [x_1, x_2, ..., x_n], \qquad \text{(eq. 2.1)}$$

Em seguida é então realizada uma classificação na seleção destas amostras de observação para, posteriormente, ser selecionada uma estatística de ordenação. Essa estatística é computada como função do vetor de observação.

O vetor de observação estendido é definido como:

$$\sim x = [\sim x_1, \sim x_2, ..., \sim x_n] = [x_1, x_2, ..., x_n, F_1(x), F_2(x), ..., F_n(x)], \qquad \text{(eq. 2.2)}$$

tal que, quando submetido ao processo de ordenação, resulta em:

$$\sim x_1 \leq \sim x_2 \leq \cdots \leq \sim x_{n+k} \quad \text{(eq.2.3)}$$

Para cada divisão do espaço de observação, uma ordem estatística de ~x é selecionada como a saída do filtro. A saída do processo de filtragem é obtida através da operação:

$$F_{EPRS} = \sim x(S(\sim\tau)), \quad \text{(eq.2.4)}$$

onde S: $\Omega_z \rightarrow \{1, 2, \ldots, n + k)$ para valores arbitrários, a ordenação da função de saída deve estar entre 1 e $n + k$, sendo Ω_z é a cardinalidade.

A cardinalidade Ω_z é tal que

$$|\Omega_z| \leq (n + k)! / (n - m + k - l)! \quad \text{(eq.2.5)}$$

Essa inequação é exata se F_1 (.) é uma amostra média tal que $x_{(1)} \leq F_1(.) \leq x_{(n)}$, e usa um critério de ordenação entre as amostras selecionadas.

A principal vantagem desse filtro é a habilidade que apresenta para realçar as bordas.

6. O *filtro da seqüência mediana ordenada* (ROM) **[Abreu, 96]** é indicado para remoção de ruído impulsivo não estacionário, ao mesmo tempo em que preserva os detalhes e a integridade da borda em restauração de imagens. Para tanto adota-se um modelo para o ruído impulsivo. Este modelo, apresentado a seguir, permite que o pixel ruidoso assuma valores arbitrários de acordo com uma distribuição de probabilidade.

Considerando, respectivamente *v(n)* e *x(n)* como os valores de luminância da imagem original e da imagem ruidosa na posição $n = [n_1, n_2]$, modela-se, então, o ruído impulsivo com probabilidade de erro p_e, tal que:

$$x(n) = \begin{cases} v(n) \text{ com probabilidade } 1 - p_e \\ n(n) \text{ com probabilidade } p_e \end{cases} \quad \text{(eq.2.6)}$$

O procedimento de filtragem é realizado do seguinte modo:

i) Seleciona-se inicialmente na imagem vetores $w(n) \in \Re^8$ formando janelas de 8 elementos, denominados de vetores de observação. Estes elementos estão presentes em janelas 3×3, centradas em $x(n)$, onde o próprio $x(n)$ é excluído das amostras de obser-

vação, resultando em w(n):

$$w(n) = [x_1(n), x_2(n), ..., x_8(n)], \quad \text{(eq. 2.7)}$$

Quando é feito um ordenamento dos elementos deste vetor temos:

$$r(n) = [r_1(n), ..., r_8(n)], \quad \text{(eq.2.8)}$$

tal que $r_1(n) \leq r_2(n) \leq ... \leq r_8(n)$ sendo então calculada uma média $m(n)$ desta seqüência ordenada, como segue:

$$m(n) = (r_4(n) + r_5(n)) / 2, \quad \text{(eq.2.9)}$$

ii) Num segundo momento, determina-se a *variável de estado*, que é definida como a saída de um classificador que opera na diferença entre os pixels de entrada e os que permanecem ordenados na janela. Esta *variável de estado* é definida como:

$$s(n) = C(x(n), w(n)), \quad \text{(eq.2.10)}$$

$$y(n) = F(x(n),w(n),s(n)) \equiv f(x(n),w(n),C(x(n),w(n)) \equiv \alpha_{s(n)}x(n) + \beta_{s(n)}m(n) \quad \text{(eq. 2.11)}$$

em que $\alpha_i : i = 1, ..., M$ e $\beta_i : j = 1, ..., M$ são considerados os coeficientes escalares correspondentes a cada um dos M estados possíveis e são escolhidos pela estimativa do valor do pixel atual.

A função da variável de estado é fornecer uma indicação probabilistica da presença do ruído impulsivo, tal que os coeficientes $\{\alpha_i\}$ e $\{\beta_i\}$ possam ser escolhidos para estimar o valor do pixel. Quando a variável de estado indica a presença de corrupção, o pixel central é descartado e substituído. Essa substituição é baseada na ordem estatística dos pixels restantes na janela. Então, o valor original do pixel é substituído apenas se este for detectado como ruidoso.

O classificador opera na diferença entre o pixel de entrada $x(n)$ e os elementos restantes de posição ordenada do vetor $r(n)$. A diferença da posição ordenada é definida por:

$$d_k(n) = \begin{cases} r_k(n) - x(n) & \text{se } x(n) \leq m(n) \\ x(n) - r_{9-K}(n) & \text{se } x(n) > m(n) \end{cases} \quad \text{(eq. 2.12)}$$

$$K = 1,...,4$$

O filtro ROM pode também ser aplicado em duas etapas e em multietapas. No mé-

todo de duas etapas, o objetivo do classificador é determinar se o pixel corrente é ou não ruidoso. Essa decisão é efetuada através de limiares ($T_1 < T_2 < T_3 < T_4$), onde o algoritmo detecta $x(n)$ como amostra ruidosa e atribui $s(n) = 1$ se as inequações:

$$d_k(n) > T_K, \text{ para } K = 1, ..., 4, \qquad \text{(eq.2.13)}$$

são verificadas como verdadeiras. Caso contrário, atribui-se $s(n) = 2$. Geralmente, os melhores valores (já testados) para os limiares são:

$$T_1 < 15,\ 15 \leq T_2 \leq 25,\ 30 \leq T_3 \leq 50 \text{ e } 40 \leq T_4 \leq 60. \qquad \text{(eq.2.14)}$$

Por outro lado, o método de multiestado funciona com a mesma lógica do método anterior porem a saída do classificador provém da divisão do espaço vetorial $\Re^4$ em M regiões baseadas no valor do vetor $d(n) \in \Re^4$, dado por :

$$d(n) = [d_1(n),...,d_4(n)]; \qquad \text{(eq.2.15)}$$

em que,

$$d_1(n) = [-\infty, T_1\text{-}20, T_1\text{-}5, T_1, T_1\text{+}5, T_1\text{+}20, \infty];$$

$$d_2(n) = [-\infty, T_2\text{-}20, T_2\text{-}5, T_2, T_2\text{+}5, T_2\text{+}20, \infty];$$

$$d_3(n) = [-\infty, T_3\text{-}30, T_3\text{-}10, T_3, T_3\text{+}10, T_3\text{+}30, \infty];$$

$$d_4(n) = [-\infty, T_4\text{-}30, T_4\text{-}10, T_4, T_4\text{+}10, T_4\text{+}30, \infty].$$

O filtro ROM é um filtro adaptativo não linear que apresenta melhor desempenho do que o filtro da mediana para imagens com altas taxas de ruído impulsivo com valores fixos, tais como o *ruído salt-and-pepper*. Experimentos indicam que este método de filtragem também é indicado para tratar diversos outros tipos de ruídos tais como gaussiano, ou mistos de ruído impulsivo e gaussiano. Além disso, este método apresenta pouco crescimento computacional em relação ao filtro da mediana.

C. Filtros Operadores de Borda:

Pela maneira como detectam a direção das bordas, os *operadores de borda* podem ser classificados como *diferenciais* ou *direcionais*.

7. Os *operadores diferenciais* consistem de máscaras com pesos definidos execu-

tando uma diferenciação discreta em cada ponto da imagem para produzir uma imagem gradiente **[Araújo, 87]**. São exemplos de operadores diferenciais os operadores de Roberts, Sobel, Prewitt e os laplacianos. Para maiores detalhes sobre esses algoritmos consultar **[Araújo, 87] e [Barros, 90].**

8. Os *operadores direcionais* atuam sobre os pixels da imagem através de um conjunto de máscaras representando aproximações discretas das bordas ideais em várias direções **[Araújo, 87]**. São exemplos de operadores direcionais os operadores direcionais de *Robinson* e *Prewitt* assim como os de *Kirsch*.

D. Filtros Morfológicos:

A *morfologia matemática* (MM) é considerada um conjunto de métodos não lineares em processamento de imagens e compreende o estudo das propriedades topológicas e estruturais dos objetos a partir de suas imagens. A MM foi inicialmente desenvolvida para análise de imagens binárias, tendo sido posteriormente estendida para imagens em níveis de cinza. Atualmente é empregada em imagens coloridas **[Candeiras, 97]**. A aplicação da MM nas imagens de Sensoriamento Remoto vem sendo usada desde 1986, em imagens dos satélites SPOT, ERS1/SAR, LANDSAT TM, NOAA, JERS **[Candeiras, 97]**.

Os filtros utilizados na morfologia matemática são definidos a partir de um elemento estruturante **[Gomes, 94]**. Esse elemento consiste de um subconjunto do plano e pode ser usado para realçar aspectos específicos das formas dos objetos, de modo que estes possam ser contados ou reconhecidos.

9. O *filtro morfológico de erosão* efetua uma transformação morfológica a qual combina dois conjuntos (regiões), formados por pixels, usando a operação de subtração nos elementos da região. Ou seja, um determinado número de pixels correspondentes a um dado padrão (modelo) são excluídos da imagem, com isto ocorre o efeito de erosão na região de interesse.

10. O *filtro morfológico de dilatação* efetua uma transformação morfológica a qual combina dois conjuntos (regiões), usando a operação de adição nos elementos do conjunto. Ou seja provoca efeitos de dilatação na região de interesse da imagem.

A composição das operações de erosão e dilatação produzem dois outros filtros morfológicos, o de abertura e o de fechamento (*opening and closing filters).* A vantagem na aplicação destes 2 filtros para suavizar ruídos em imagens digitais provém do fato de que ao mesmo tempo em que a suavização é efetuada ocorre a preservação de detalhes importantes da imagem através de simples operações de filtragem.

E. Filtros Adaptativos:

Os *filtros digitais adaptativos* são empregados com sucesso em imagens com ruídos dependentes do sinal e gaussianos. Estes filtros apresentam a característica de mudarem seus coeficientes perto de bordas ou de impulsos presentes na imagem. São filtros cujo algoritmos tendem a se adaptar ao comportamento do sinal, ou seja, mudam suas propriedades de suavização de acordo com as propriedades das regiões da imagem. A principal desvantagem destes algoritmos é que a atualização de seus coeficientes é realizada de modo heurístico, além de que não são usualmente acompanhados de uma norma de minimização de erro **[Kotropoulos, 97]**. Os algoritmos a seguir são baseados em métodos adaptativos.

11. O *filtro racional* **[Ramponi, 96]** usa uma máscara que opera com uma média adaptativa nos pixels que apresentam luminância similares. A idéia básica é modelar os coeficientes da máscara, com o objetivo de limitar sua ação na presença de detalhes da imagem. Com este propósito, foi estabelecida a ação de um operador passa-baixas. Esse operador é formulado como uma função racional, isto é, como a razão entre 2 polinômios das variáveis de entrada descrevendo, assim, a relação de entrada e saída. O objetivo de tal operador é limitar a ação do filtro na presença de detalhes da cena, ao mesmo tempo que suaviza os ruídos da imagem. A ação deste operador torna-se ausente quando são detectada mudanças relevantes no sinal.

Conforme a eq.2.16, cada amostra de saída, y_n, é obtida através de um vetor das amostras de entrada $x = [x_{n-1}, x_n, x_{n+1}]$:

$$y_n = W(x_{n-1} + x_{n+1}) + (1 - 2w)x_n \qquad \text{(eq. 2.16)}$$

Partindo da suposição de que os pixels ruidosos geralmente são representados pelos pixels extremos da janela, o operador é construído a partir de uma norma quadrada de duas

amostras extremas na janela do filtro (ou seja a escolha dos pesos é estimada entre os pixels extremos da máscara). Este operador tem o aspecto de uma função racional na variável $\{x_n\}$, empregada para detectar mudanças no sinal.

Se o valor deste operador é grande, assume-se que a máscara do filtro está posicionada na transição do sinal, de modo que a resposta em freqüência do operador é realizada menos seletivamente. O operador é expresso através de um polinômio de segunda e terceira ordens, através das seguintes equações:

$$y_n = \frac{w(x_{n-1} + x_{n+1})}{wk(x_{n-1} - x_{n+1})^2 + 1} + (1 - \frac{2w}{wk(x_{n-1} - x_{n+1})^2 + 1}) x_n \qquad \text{(eq.2.17)}$$

e

$$y_n = \frac{x_{n-1} + x_{n+1} + x_n(k(x_{n-1} - x_{n+1})^2 + 1/w - 2)}{k(x_{n-1} - x_{n+1})^2 + 1/w}, \qquad \text{(eq. 2.18)}$$

O filtro consegue reduzir o efeito de suavização na presença de detalhes através do parâmetro k:

- para $k = 0$ tem-se o filtro linear (eq. 2.18);
- para $k \to \infty$, a ação do filtro não tem efeito e $y_n = x_n$;
- para valores intermediários de k, o termo $(x_{n-1} - x_{n-2})^2$ consegue identificar a presença de detalhes e ,desse modo, reduzir o efeito de suavização do operador.

Então, o filtro racional pode ser interpretado com um filtro linear passa-baixas cujos coeficientes são modelados por componentes sensíveis a bordas. Tal filtro apresenta a característica de preservar bordas, ao mesmo tempo que suaviza o ruído presente na imagem filtrada.

12. O *filtro para realçar imagens baseado em lógica fuzzy* **[Choi, 97]** aborda um método de filtragem adaptativa, que seleciona uma região da imagem, onde é aplicado um processo seletivo para determinar qual o processo de filtragem que será aplicado. O algoritmo se baseia em administrar metas que aparentemente são conflitantes, tais como:

- remover ruídos impulsivos;

- atenuar ruído não impulsivos;
- realçar ou preservar bordas, assim como outros detalhes.

Para realizar essas três tarefas, são derivados três filtros usando o critério de erro mínimo quadrado (LS), assim como são definidos regras para selecioná-los. Essas regras são baseadas em cláusulas de antecedentes/conseqüentes. Dependendo do valor dos pixels de entrada, certas condições são satisfeitas e, posteriormente, é selecionado um filtro. Cada filtro e suas condições constituem a regra de produção.

A vantagem em usar sistemas baseados em regras *fuzzy* é a flexibilidade que apresentam de resolver, de modo trivial e eficiente, a incerteza das informações contidas nas imagens (informações tais como o comportamento do ruído), pois tomam decisões baseados em condições específicas, agregam essas decisões e finalmente, tomam uma decisão global, baseada no resultado desta agregação.

O critério de decisão adotado para cada filtro constitui a cláusula antecedente e os filtros resultantes correspondentes constituem as cláusulas conseqüentes das regras *fuzzy*. Em suma, para cada pixel, qualquer uma das cláusulas antecedentes pode ser satisfeita e todos os filtros podem ser aplicados. O conjunto de regras *fuzzy* apresentado abaixo é usado para auxiliar e definir as condições precisas pelas quais cada um dos filtros devem ser aplicados.

A regra adotada neste sistema de filtragem é do tipo *se–então-senão,* sendo constituída de $M + 1$ regras com múltiplas variáveis de entrada, tal que:

- $A_{ki} \rightarrow$ rótulo lingüistico associado à i–ésima variável de entrada X_{ki} na regra K;
- $F_k \rightarrow$ ação desejada na regra K;
- $N_k \rightarrow$ número de variáveis de entrada na regra K;
- $\oplus \rightarrow$ operador;

Regras

Regra 1: se X_{11} é $A_{11} \oplus \ldots \oplus X_{1i} \wedge A_{1i} \oplus \ldots \oplus X_{1N_1}$ é A_{1N_1} então F_1

Regra 2: se X_{21} é $A_{21} \oplus \ldots \oplus X_{2i} \wedge A_{2i} \oplus \ldots \oplus X_{2N_2}$ é A_{2N_2} então F_2

Regra *k*: se X_{31} é $A_{31} \oplus ... \oplus X_{3i} \wedge A_{3i} \oplus ... \oplus X_{3N_3}$ é A_{3N_3} então F_3

Regra *M*: se X_{11} é $A_{11} \oplus ... \oplus X_{1i} \wedge A_{1i} \oplus ... \oplus X_{1N_1}$ é A_{1N_1} então F_1

Regra *M*+1: senão F_{m+1}

A ação F_k pode ser um filtro ou um conjunto de regras, sendo determinado um operador C_k (eq.2.19) com o objetivo de estabelecer o grau de satisfação do antecedente na regra *K*:

$$C_k = (A'_{k1} \circ A_{k1}) \oplus (A'_{k2} \circ A_{k2}) \oplus ...(A'k_{nk} \circ Ak_{nk}), \qquad \text{(eq. 2.19)}$$

tal que: $K = 1 - M$.

Então, a saída Y (eq.2.20) de um vetor de entrada para este sistema é :

$$Y = f[(C_1 \circ F_1) ..., (C_{M+1} \circ F_{M-1})] \qquad \text{(eq. 2.20)}$$

em que $\circ$ é um operador de composição e f é uma função de defuzzificação f_d, podendo ser selecionado:

- a multiplicação para o operador composição;
- a média ponderada para a f_d ;
- e a saída do filtro para o conseqüente,

Daí resulta então que a saída Y deste sistema de regra fuzzy para o realce da imagem pode ser escrita como:

$$Y = \frac{\sum_{K=1}^{M+1} C_k F_k}{\sum_{K=1}^{M+1} C_k} \qquad \text{(eq. 2.21)}$$

As regras heurísticas apresentadas a seguir sobre o pixel central são quem determinam as condições de como os filtros deverão ser aplicados:

Se o pixel central

- é um pixel impulsivo, o nível de cinza desse pixel é significativamente diferente do valor dos seus vizinhos, ou seja, o grau de compatibilidade desse pixel com os pixels

vizinhos é pequena, aplicando-se nesse caso o sistema de filtragem *B*, eq. 2.24;

- está situado numa região suave onde a variação do nível de cinza não é significante, existe uma alta possibilidade de os pixels vizinhos serem compatíveis com este pixel central. O grau de compatibilidade é grande e nesse caso aplica-se o sistema de filtragem *C*,eq.2.25;
- pertence a fronteira, então o grau de compatibilidade com os pixels vizinhos será médio comparado com os casos acima. Neste caso aplica-se o sistema de filtragem A, eq. 2.23;

Definições:

Para avaliar a situação de um determinado pixel em uma janela, usa-se a medida de compatibilidade total T_C dos pixels vizinhos em relação ao valor do pixel central. A medida grau de compatibilidade T_C é dada pela eq.2.22:

$$\frac{1}{k}\sum_{j=1,j\neq i}^{N}\mu_{ji} \qquad \text{(eq. 2.22)}$$

O algoritmo do filtro A é:

$$I(X_i)=\frac{\sum_{j=1}^{N}\mu_{ij}(1-(\frac{d_{ij}}{\beta_i}^2))I(X_j)}{\sum_{j=1}^{N}\mu_{ij}.(1-(\frac{d^2_{ij}}{\beta_i}.)} \qquad \text{(eq. 2.23)}$$

O algoritmo do filtro *B* é:

$$I(X_i)=\frac{\sum_{j=1,j\neq i}^{N}\frac{1}{\beta_j}.(X_j)}{\sum_{j=1,j\neq 1}^{N}\frac{1}{\beta_j}} \qquad \text{(eq. 2.24)}$$

O algoritmo do filtro *C* é dado por:

$$I(X_i) = \sum_{j=1}^{N} \frac{\mu_{ji} I(X_j)}{\beta_j} \qquad \text{(eq. 2.25)}$$

tal que $I(x_i)$ é o nível de cinza dos pixels pertencentes a uma janela, β_j representa um estimador determinado pela variação da intensidade dos pixels em uma dada janela espacial, e μ_{ji} é o grau de compatibilidade da vizinhança do pixel x_j com respeito a x_i.

As regras de decisão para realçar a imagem dependem do domínio de aplicação. Este algoritmo permite que sejam construídos variados tipos de sistemas de filtragem baseados em outras regras para uma determinada aplicação, apresentadas a seguir:

O *Sistema de Filtragem* R_1 é empregado para remover ruído impulsivo usando as seguintes regras:

*R*egra 1: se o grau de compatibilidade é pequeno, então:

$$Y_1 = \frac{\sum_{j=1, j\neq i}^{N} \frac{1}{\beta_j} \cdot (X_j)}{\sum_{j=1, j\neq 1}^{N} \frac{1}{\beta_j}} \qquad \text{(eq. 2.26)}$$

*R*egra 2: caso contrário,

$$y_2 = I(X_i) \qquad \text{(eq. 2.27)}$$

O *Sistema de filtragem* R_2, é baseado no fato de que para que um filtro preserve uma borda ruidosa, emprega-se uma regra que envolve o filtro *B* e *C*:

*R*egra 1: se a compatibilidade é pequena então,

$$Y_1 = \frac{\sum_{j=1, j\neq i}^{N} \frac{1}{\beta_j} \cdot (X_j)}{\sum_{j=1, j\neq 1}^{N} \frac{1}{\beta_j}}, \qquad \text{(eq. 2.28)}$$

*R*egra 2: caso contrário:

$$Y_2 = \frac{\sum_{j=1}^{N} \frac{\mu_{ji}}{\beta_j} I.(X_j)}{\sum_{j=1}^{N} \frac{\mu_{ji}}{\beta_j}}. \qquad \text{(eq. 2.29)}$$

O *sistema de filtragem* R_3 é baseado no fato de que se necessita filtrar os pixels ruidosos mas preservando detalhes.

*R*egra 1: Se a T_C é pequena, então:

$$Y_1 = \frac{\sum \mu_{iij} \frac{1-d_{ij}^2}{\beta_i}}{\sum \mu_{ij} \frac{1-d_{ij}^2}{\beta_i}}, \qquad \text{(eq. 2.30)}$$

*R*egra 2: Se T_C é média, então:

$$Y_2 = \frac{\sum_{j=1, j\neq i}^{N} \frac{1}{\beta_j} .(X_j)}{\sum_{j=1, j\neq 1}^{N} \frac{1}{\beta_j}}, \qquad \text{(eq. 2.31)}$$

*R*egra 3: Caso contrário:

$$Y_3 = \frac{\sum_{j=1, j\neq i}^{N} \frac{\mu_{ji}}{\beta_j} I.(X_j)}{\sum_{j=1, j\neq 1}^{N} \frac{\mu_{ji}}{\beta_j}}. \qquad \text{(eq. 2.32)}$$

O método de filtragem dos sistemas que empregam da lógica *fuzzy* baseada no método do mínimo quadrado ponderado cria condições de avaliar a situação dos valores dos pixels vizinhos em relação ao valor do pixel central. O resultado global é obtido através da combinação dos resultados aplicados aos valores de cada pixel, cada resultado contribuindo

para o grau de satisfatibilidade da cláusula antecedente correspondente. O tempo de CPU gasto pelos sistemas de filtragem R_1, R_2 e R_3 para efetuar uma operação de filtragem é muito próximo àquele gasto pelo filtro da mediana.

13. O *filtro L mínimo quadrado médio (LMS)* **[Kotropoulos, 96]** emprega técnicas adaptativas e apresenta a vantagem de ter sido projetado com um critério para estimativa da minimização do erro que geralmente ocorre entre os resultados obtidos no processo de filtragem e no sinal livre de ruídos. O critério de erro considerado é o *LMS*. Como se trata de um processo adaptativo, os coeficientes do filtro são atualizados de acordo com as características da imagem. Neste caso, é desenvolvido um critério para sinais não constantes e corrompidos por ruídos brancos aditivos de média zero.

A janela do filtro é definida em uma vizinhança em torno de cada pixel k, de dimensão $M \times M$, assumindo que M é um número ímpar, tal que $M = 2\xi + 1$,como mostrado na eq.2.33. A janela do filtro desliza sobre a imagem no modo *scan raster*, tal que $N = M^2$.

$$X(K) = \begin{bmatrix} x(i-\varepsilon, j-\varepsilon) & x(i-\varepsilon, j-\varepsilon+1) & \cdots & x(i-\varepsilon, j+\varepsilon) \\ \vdots & \vdots & & \vdots \\ x(i+\varepsilon, j-\varepsilon) & x(i+\varepsilon, j-\varepsilon+1) & \cdots & x(i+\varepsilon, j+\varepsilon) \end{bmatrix} \qquad \text{(eq. 2.33)}$$

Deste modo, o filtro aplicado é baseado num ordenamento dos pixels da amostra. Por exemplo, a janela $x(k)$ pode ser organizada em um vetor $N \times 1$ (representando a janela do filtro em linha por linha), como segue:

$$x(k) = (x(i-\xi, j-\xi), x(i-\xi, j-\xi+1), \ldots, x(i-\xi, j+\xi), \ldots, x(i+\xi, j+\xi))^T$$

Se K e L denotam a linha e a coluna da imagem, então:

$$K = (i\text{-}1)K + j, \text{ com } 1 \le i \le K,\ 1 \le j \le L \qquad \text{(eq. 2.34)}$$

Uma ordenação dos pixels da janela é representada por:

$$x_r(k) = (x_{(1)}(k), x_{(2)}(k), \ldots, x_{(N)}(k))^T \qquad \text{(eq.2.35)}$$

tal que cada $x_{(i)}(k)$ é denotada a i-ésima maior observação da janela de entrada. A saída $y(k)$

é determinada por:

$$y(k) = a^T(k)\, x_r(k) \qquad \text{(eq.2.36)}$$

tal que os coeficientes do vetor do filtro L são obtidos pela seguinte equação:

$$\hat{a}(k+1) = \hat{a}(k) + \mu\varepsilon(k)\, \hat{x}(k) \qquad \text{(eq.2.37)}$$

em que ξ (k) é a estimativa de erro do pixel k, e a eq. 2.37 é recursiva, sendo que a atualização dos coeficientes do filtro são derivadas dos sinais corrompidos por ruído aditivo.

2.2.3 O Método Vetorial e os Filtros para as Imagens Multiespectrais

Nos últimos anos uma grande quantidade de técnicas de filtragem tem sido aplicada no processamento das imagens coloridas, especialmente os filtros não lineares. Esses filtros tendem a preservar bordas e detalhes e remover ruídos gaussianos e impulsivos quando aplicados em imagens coloridas. Entretanto, como o processo de filtragem é realizado em cada componente separadamente, a correlação existente entre as coordenadas das bandas não são consideradas. O processamento em separado de sinais de valores vetoriais de dimensão maior que dois podem causar problemas que não são detectados em sinais escalares **[Astola, 90]**.

Os últimos 10 anos de avanços nas pesquisas realizadas nas áreas de processamento de sinais multicanais (ou multidimensionais) permitiram que surgisse uma nova categoria de técnicas de processamento, "as técnicas de processamento vetorial" para as imagens coloridas. Os filtros espaciais que se utilizam destas técnicas de filtragem para imagens multicanais, usualmente consideram critérios de erro da imagem caracterizados por atributos vetoriais. Os atributos vetoriais geralmente considerados são a distância, a direção ou ambos. Os algoritmos apresentados a seguir empregam essas técnicas nos algoritmos de filtragem.

F. <u>O Método Vetorial e os Filtros para as Imagens Multiespectrais:</u>

14. O *filtro da mediana vetorial VMF* **[Astola, 90]** é baseado nas propriedades do filtro da mediana escalar. O filtro apresenta resposta zero ao impulso, suavizando os dados enquanto retém o formato das bordas e não introduzindo qualquer valor na amostra de saída que não esteja presente na amostra de entrada. A operação de filtragem apresenta resul-

tados similares à operação mediana no caso escalar.

O filtro da mediana vetorial considera dois algoritmos diferentes para processar os sinais de valores vetoriais. A mediana vetorial **[Astola, 90]**, no primeiro caso, é calculada do seguinte modo:

1. Um ordenamento de valor escalar é associado a cada vetor. Através deste ordenamento, é realizada a seguinte classificação:
   - A cada posição da janela W (W é a janela), $\forall \ \underline{x}_j \in W$,calcula-se a distância euclidiana de cada vetor e depois efetua-se a classificação baseada na distancia;
2. No momento a seguir é escolhido o valor mediano da distância:

$$\bar{x}_{Med} = med\left\| x_j \right\|_p \qquad \text{(eq. 2.38)}$$

O outro modo efetua a escolha de um vetor protótipo (geralmente o vetor selecionado é àquele que ocupa a posição central na janela), de modo que a mediana vetorial é calculada da seguinte forma:

1. para cada vetor $\underline{x}_i \in W$ (W é a janela do filtro), calcula-se a distância do vetor protótipo em relação a todos os vetores, usando a norma L_1 ou L_p. Em seguida adiciona-se o resultado:
   - $S_i = \sum_{j=1}^{N} \left\| \underline{x}_i - \underline{x}_j \right\|$ *onde* $i = 1, ..., N$ (eq.2.39)
   - tal que $\underline{x}_i$ é escolhido de acordo com a sua posição na janela;
2. em seguida calcula-se o valor S_{min} ,que é obtido através do min de S_i ou seja, a menor das normas;
3. o vetor mediano é $\underline{x}_{i\,min}$ encontrado no cálculo dos min S_i , ou seja será o vetor associado à menor norma calculada através da (eq. 2.39). O vetor protótipo selecionado pode também ser considerado como o valor da média dos outros vetores, eq. 2.40:

$$(\underline{x}_i = 1/n \sum_{i=1}^{N} \underline{x}_i). \qquad \text{(eq.2.40)}$$

No caso do min dos S_i não ser único, qualquer $\underline{x}_i$ resultante de min S_i pode ser selecionado como o vetor mediano. No caso de existirem diversos candidatos, a decisão deve ser baseada na posição ocupada na janela do filtro.

15. O *filtro mediano vetorial reduzido* **[Regazzoni, 97]** é baseado no algoritmo do

filtro da Mediana Vetorial. É um filtro da mediana multidimensional indicado para remover ruídos impulsivos e preservar bordas. A propriedade interessante deste filtro é que, na etapa anterior a operação de filtragem o algoritmo realiza uma transformação *vetorial-escalar*, ou seja, os valores dos pixels da imagem original, são transformados em valores escalares. Este método de filtragem é estruturado em três etapas, conforme a descrição apresentada nos parágrafos seguintes.

Na primeira etapa é executada a transformação dos dados vetoriais em dados do espaço de uma dimensão (1 D), ou seja, dados escalares. Essa transformação é denominada vetorial-escalar e se baseia no conceito de curvas do preenchimento de espaço de *Peano*. Tais curvas são consideradas curvas discretas contínuas que apresentam a propriedade de poderem transformar todos os pontos de um espaço vetorial de dimensão d em um intervalo escalar. Assim, a cada ponto do espaço de dimensão d é possível associar um valor escalar diretamente proporcional ao comprimento da curva necessária para alcançar cada ponto, a partir das coordenadas da origem. As curvas de preenchimento do espaço permitem a redução na complexidade computacional em relação ao filtro VMF e reduzem o tempo de processamento.

Na segunda etapa, efetua-se a operação de filtragem nos valores escalares obtidos através da transformação vetorial-escalar. Esta operação é realizada de modo semelhante ao filtro da mediana escalar.

Já na terceira etapa é aplicada uma operação inversa nos pixels filtrados, ou seja, é aplicada uma anti-transformada que transforma os pixels de valores escalares em pixels com valores de vetores.

Essas etapas podem ser representadas de tal forma:

Seja um conjunto de vetores $w = \{ \underline{x}_i : i = 1,..., n\}$ então:

1) calcula-se $T(\underline{x}_i)$ para todo $\underline{x}_i \in w$;
2) calcula-se a mediana de $\{ T(\underline{x}_i) \} = T(\underline{x}^*)$;
3) calcula-se $T^{-1}(T(\underline{x}^*))$.

No caso 2-D a curva de preechimento do espaço é reduzida a uma função γ que associa um valor escalar aos vetores 2-D. Então:

$$\gamma : Z \rightarrow K_2 \quad K_2 \subset Z^2 \qquad K_2 = [0, X_{1max}] \times [0, X_{2max}] \qquad \text{(eq.2.41)}$$

$$\gamma(t_k) = (x_{1k}(t_k), x_{2k}(t_k))$$

No caso 3-D a curva de preechimento do espaço é reduzida a uma função γ que associa um valor escalar aos vetores 3-D. Então:

$$\gamma : Z \rightarrow K_3 \quad K_3 \subset Z^3 \quad K_3 = [0, X_{1max}] \times [0, X_{2max}] \times [0, X_{3max}] \qquad \text{(eq.2.42)}$$

$$\gamma(t_k) = (x_{1k}(t_k), x_{2k}(t_k), x_{3k}(t_k))$$

Para o caso 3-D a curva de preenchimento do espaço pode ser imaginada como a expansão de sucessivas camadas ordenadas de acordo com o valor da norma de cada vetor 3-D.

O filtro da mediana vetorial reduzido é interessante por implementar um novo método capaz de estender os procedimentos de filtragem mediana à dados vetoriais de maneira computacionalmente eficiente.

16. Os *filtros adaptativos não lineares para múltiplos canais* **[Venetsanopoulos, 97]** são propostos para filtrar imagens coloridas, baseando-se no fato de não precisarem de informações específicas a *priori* sobre o sinal ou sobre as características do ruído. O processo de filtragem é estruturado em duas etapas. A finalidade deste algoritmo é derivar filtros adaptativos sobre uma certa família de densidade de probabilidade, tais como:

- família da densidade gaussiana;
- família da densidade exponencial dupla;
- família de todas as densidades que surgem em aplicações de processamento de imagens;

Em suma, a idéia é desenvolver técnicas que possibilitem a obtenção de parâmetros através destas distribuições, viabilizando o processo de filtragem. Há 2 métodos destinados à otimização do processo de modelagem destes parâmetros: o procedimento da inferência *Bayesiana* e as técnicas não-paramétricas.

Para modelar a degradação sofrida pela imagem usando o procedimento de inferência *Bayesiana*, estima-se o parâmetro *função perda*. Este parâmetro é usado para imperme-

abilizar os erros que ocorrem durante o processo de filtragem. O valor desta função depende do vetor ruidoso e da estimativa a ser filtrada.

O processo de filtragem ocorre do seguinte modo:

1- Num primeiro momento, determina-se a função densidade de probabilidade. Esta função é condicionada aos vetores da imagem nos dados da imagem ruidosa;

2- A função de densidade tem geralmente uma forma funcional conhecida, porém um conjunto de parâmetros desconhecidos;

3- Tendo em vista que em processamento de imagens encontra-se geralmente uma certa família de modelos de ruído, introduz-se uma distribuição de origem simétrica baseada nestes modelos. Esses modelos são caracterizados pelos parâmetros de localização escalar e um grau de não normalidade da distribuição (γ);

4- Com base nestas distribuições de origem generalizada, concebe-se um *estimador adaptativo,* usando as técnicas de inferência Bayesiana.

Se a função densidade de probabilidade (fdp) é conhecida então o *estimador adaptativo* pode ser facilmente determinado. Caso contrário, se assume que a fdp apresenta uma forma funcional conhecida, mas com um conjunto de parâmetros desconhecidos. Baseado numa distribuição de origem generalizada " *a função Gaussiana Multivariada Generalizada"(f(m/*θ *,*σ *,* γ *)),* é concebido então o *estimador adaptativo,* que pode ser concebido usando a técnica de inferência Bayesiana (eq. 2.43).

$$f(m/\theta,\sigma,\gamma) = K^{M} \exp(-0{,}5\beta\left(\frac{|m-\theta|}{\sigma}\right)^{2/1+\theta}), \qquad \text{(eq. 2.43)}$$

em que M é a dimensão da medida do espaço e a variância (σ) é uma matriz $M \times M$ que pode ser considerada como diagonal com elementos σ_C ,tal que c = 1,...,M, enquanto que:

$$\beta = \frac{\tau(1{,}5(1+\gamma))}{\tau(0{,}5(1+\gamma))}^{1/1+\gamma} \qquad \text{(eq.2.44)}$$

$$K = \left(\frac{(\tau(1{,}5(1+\gamma))^{0{,}5}}{(1+\gamma)\ (\gamma(0{,}5(1+\gamma))^{0{,}5}}\right)\sigma^{-1} \qquad \text{(eq.2.45)}$$

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \qquad \text{(eq.2.46)}$$

para $x > 0$.

Essa é uma densidade simétrica de desvio *2*, que oferece grande flexibilidade, pois consegue-se gerar diferentes distribuições a partir da alteração do parâmetro γ . Podendo ser aplicado a diferentes tipos de ruídos. Se a janela *W* do filtro que desliza sobre a imagem, em uma dada localização, apresenta *n* vetores ruidosos, então o que se deseja é estimar um sinal constante para o valor central desta janela. Esta estimativa se baseia nas *n* observações ruidosas.

O procedimento da inferência estatística Byaesiana fornece meios para a seleção de uma densidade apropriada dentre a família de densidades consideradas.

Por outro lado o método não-paramétrico é considerado um método mais generalizado, pois parte do pressuposto de que a forma funcional da função densidade do ruído presente na imagem é realizada. Assume-se que n pares de vetores da imagem (x_l, y_l) com $l=1, ...,n$, são obtidos através da janela de comprimento *n*, centralizada em torno do ruído observado *y*. Com base nestas amostras, as funções de densidades *f(y)* e *f(x,y)* são aproximadas usando pontos da amostra do estimador adaptativos não-paramétrico.

2.2.4 Filtros para Imagens SAR

Os Sistemas de Radar de Abertura Sintética SAR, são compostos de radares imageadores que apresentam uma antena sintética, antena esta usada para receber sinais de todos os alvos **[Novo, 93]**. Uma imagem SAR é gerada pela produção de diferentes visadas de um mesmo objeto, e são obtidas por sensores ativos que operam nas faixas de microondas **[Mascarenhas, 94]**. Os pixels que compõem a imagem de radar são formados por dois atributos do campo elétrico, amplitude e fase, representados por um número complexo (ponto flutuante) com componente real (amplitude) e imaginária (fase) **[Paradella, 96]**. As imagens de radar são sensíveis a rugosidade ou texturas da superfície, sendo excelentes para definição de diferenças na superfície terrestre, tais como espécies vegetais e tipos de rochas **[Conway, 96]**.

As imagens SAR são corrompidas por um tipo de ruído que depende do sinal, o *ruído speckle,* que é um ruído multiplicativo descrito pela distribuição de *Rayleigh*, a qual varia de coordenada para coordenada sob as condições de uma visada e deteção linear **[Mascarenhas,96].** O problema de filtrar uma imagem de radar pode ser encarado como um processo de estimar parâmetros dentro da janela do filtro sobre a imagem. Esses estimadores apresentam um bom comportamento quando as observações dentro das amostras consideradas são conseqüência das variáveis aleatórias de *Rayleigh* independentes identicamente e coletivamente distribuída.

O ruído *speckle* tende a dificultar a interpretação das informações contidas na imagem, motivo pelo qual diversos algoritmos têm sido propostos para eliminar este ruído, sem corrupção dos detalhes finos da imagem. Geralmente os algoritmos dos filtros propostos para redução do ruído *speckle* assumem que todas as amostras provêm da mesma distribuição (por exemplo sobre áreas homogêneas, com deteção linear e uma visada).

A seguir é apresentada uma descrição dos algoritmos usualmente encontrados nas ferramentas de SIG, assim como alguns dos algoritmos recentemente propostos na literatura especializada.

1. O *filtro de Estatística local* **[Smith, 96]** é aplicado em imagens SAR de multivisada. Na imagem observada, o valor do pixel Z, está relacionado com o valor do pixel sem ruído, representado pela seguinte equação:

$$Z = X.V \qquad \text{(eq. 2.47)}$$

com desvio padrão $\sigma_Z = \sigma_v \ \bar{z}$, tal que σ_Z representa o desvio padrão da imagem observada, $\bar{Z}$ uma área homogênea da imagem e σ_V é o desvio padrão do ruído *speckle*;

O filtro tende a minimizar o erro causado pela presença do ruído através de um estimador $\hat{x}_c$, determinado do seguinte modo:

$$\hat{x}_c = .. \bar{z} + K\,(Z_C - .. \bar{z}) \qquad \text{(eq. 2.48)}$$

em que

$$K = \frac{\text{var}(x)}{\overline{z}^2 \sigma_v^2 + \text{var}(x)} \qquad \text{(eq. 2.49)}$$

e

$$\text{var}(x) = \frac{\text{var}(x) + \overline{Z^2}}{\sigma_v^2 + 1} \qquad \text{(eq. 2.50)}$$

Este filtro preserva bordas em regiões de alto contraste (onde var (x) é grande), ou seja:

$$K \rightarrow 1 \text{ e } \hat{x}_c \rightarrow Z_C$$

O problema deste filtro é que como o valor do estimador $\hat{x}_c$ e o valor de Z_C, tornam-se aproximadamente iguais, resulta então que nas regiões de alto contraste o ruído tende a ser preservado.

2. No *Filtro Sigma* **[Smith, 96]**, o pixel central da janela do filtro é considerado a média da distribuição gaussiana. Como 95,5% das amostras aleatórias de uma distribuição normal gaussiana aceitam valores dentro de 2 desvios padrões, então tem-se que os valores dos pixels da janela pertencem a um dos 2 valores médios, enquanto que os valores dos pixels fora da faixa de 2 sigmas são provavelmente considerados pixels de outra distribuição. O filtro sigma só utiliza para cálculo os pixels da janela de filtragem, que estão dentro do alcance de 2 sigmas do valor do pixel central.

As bordas são preservadas, pois os pixels que não pertencem à mesma distribuição que o pixel central são excluídos do processo de cálculo. Na faixa de dois sigmas, se assume que o pixel central é a média da distribuição, ou seja:

$$Z_{min} = (1 - 2\,\sigma_V)\, Zc_1 \qquad \text{(eq. 2.51)}$$

$$Z_{max} = (1 + 2\sigma_V) Z_c \text{ então } Z = (Z_{min} + Z_{max}) / 2 \qquad \text{(eq. 2.52)}$$

O problema surge quando dentro da janela do filtro não existe nenhum pixel dentro da faixa de dois sigmas. Semelhante ruído de mancha aguda, é negociado com a introdução de um limiar K_s. Se o número de pixels dentro da faixa de dois sigmas é menor ou igual a K_s, então o pixel central é substituído pelo cálculo dos quatro vizinhos mais próximos.

3. No *Filtro Sigma Ponderado* **[Smith, 96]**, se considera o pixel central como pertencente à faixa da distribuição gaussiana. Para reduzir ao máximo o ruído *speckle*, inclui-se no processo do cálculo todos os pixels que pertencem possivelmente à mesma distribuição que o pixel central, excluindo aqueles que não pertencerem a tal distribuição.

Para cada pixel Z_i da janela, w, do filtro, deve-se determinar se Z_i e Z_c estão ambos dentro da faixa de dois sigmas da mesma distribuição gaussiana. Em caso afirmativo, deve-se incluir Z_i no cálculo.

O pixel Z_i da janela é incluso no processo de cálculo se a distribuição gaussiana centrada em Z_i com desvio padrão $\sigma_{v} Z_i$ apresentar o valor do pixel central dentro da faixa de dois sigmas. O filtro ponderado efetua o cálculo para todos os pixels que estão entre Z_{min} e Z_{max} do seguinte modo:

$$Z_{min} = \frac{Z_c}{1+2\sigma_v} \qquad \text{(eq. 2.53)}$$

$$Z_{max} = \frac{Z_c}{1-2\sigma_v} \qquad \text{(eq. 2.54)}$$

O problema é que, com base na distribuição gaussiana, cada pixel da janela do filtro exclui possíveis pixels de valores baixos que estão dentro do alcance de dois sigmas do valor do pixel central. Alguns detalhes que o filtro sigma preserva são aniquilados pelo filtro sigma ponderado.

4. *O Filtro Sigma Modificado* **[Smith, 96]** considera que a média da distribuição gaussiana à qual o pixel central pertence, $\hat{z}$, pode ser tomada como uma combinação linear de Z_i e Z_c, da seguinte forma:

$$\hat{z} = f.Z_i + (1-f).Z_c, \qquad \text{(eq.2.55)}$$

em que f é uma constante a ser especificada;

Para que Z_i *e* Z_c estejam dentro da faixa de *2* sigmas de $\hat{z}$ e, portanto, pertençam a mesma distribuição, as inequações abaixo devem ser satisfeitas:

$$(1 + 2\,\sigma_v)\ \hat{z} \geq Z_c (1 + 2\,\sigma_v)\ \hat{z} \geq Z_i \qquad \textit{(eq. 2.56)}$$

$$Z_{min}{}^{2} = \frac{(1-(1-f)(1+2\sigma_v))}{f(1+2\sigma_v)} Z_c \qquad \text{(eq.2.59)}$$

$$Z_{max}{}^{3} = \frac{(1-f)(2+\sigma_v)}{(1-f)(1+2\sigma_v)} Z_c \qquad \text{(eq.2.60)}$$

$$Z_{max}{}^{4} = \frac{(1-(1-f)(1-2\sigma_v))}{f(1-2\sigma_v)} Z_c \qquad \text{(eq.2.61)}$$

Para $f = 0$ e $f = 1$, o filtro 2 sigma torna-se o filtro sigma e o filtro ponderado, respectivamente recuperados. Para incluir todos os pixels que pertencem à mesma distribuição que o pixel central, f deve ser escolhido de maneira tal que Z_{min} seja minimizado e Z_{max} maximizado. Ou seja:

$$Z_{min} = \frac{(1-2\sigma_v)}{(1+2\sigma_v)} Z_c \qquad Z_{max} = \frac{(1+2\sigma_v)}{(1-2\sigma_v)} Z_c \qquad \text{(eq.2.62)}$$

O valor mínimo de Z_{min} ocorre quando:

$$Z^1_{min} = Z^2_{max}$$

e Z_{max} é maximizado quando:

$$Z^1_{max} = Z^2_{max} \qquad \text{(eq. 2.63)}$$

O alcance de dois sigmas inclui todos os pixels que pertencem à mesma distribuição que o pixel central, ocorre quando f=0,5 e é definida por:

$$Z_{min} = \frac{1-2\sigma_v}{1+2\sigma_v} Z_c \qquad Z_{max} = \frac{1+2\sigma_v}{1-2\sigma_v} Z_c \qquad \text{(eq. 2.64)}$$

O artigo de Mascarenhas **[Mascarenhas, 96]**, apresenta uma discussão mais detalhada sobre vários algoritmos de filtragem espacial para atenuar o ruído *speclke* em imagens SAR.

2.3 Filtros Espaciais em Ferramentas Comerciais de Sistemas de Informações Geográficas

Diversas ferramentas de Sistemas de Informações Geográficas disponíveis no mercado foram investigadas para verificar quais as técnicas de filtragem espacial presentes. Para facilitar a compreensão dos resultados obtidos foi elaborada uma representação tabular, ilustrada na Tabela 2.1. A relação das ferramentas investigadas esta especificada abaixo.

A. SPRING versão 2.0.3;
B. IDRISI for WIN versão 4.1;
C. MGE Advanced Imager (MAI);
D. MGE Base Imager ;
E. GMT-3;
F. V-Image Remote Sensing Software;
G. MAP II;
H. GIS by Genasys;
I. EASI/PACE versão 6.1;
J. ER MAPPER;
L. PCI Remote Sensing ;
M. ENVI versão 2.6;

A ferramenta *Spring* versão 2.0.3 apresenta opções de máscaras pré-definidas, um editor de máscaras para filtros lineares e não lineares permitindo assim que o usuário crie, atualize e aplique suas próprias máscaras [INPE, 96].

A ferramenta *IDRISI for WIN* oferece ao usuário o núcleo dos filtros da média, moda e mediana pronto e este mesmo núcleo pode ser usado para criar filtros direcionais **[IDRISI, 95]**.

O *software MGE Advancer Imager* (MAI) é um ambiente funcional para manipular imagens multiespectrais, recomendado para operação em sistemas de coordenadas com acesso a base de consulta e gerenciamento para projeto em sistemas de informações geográficas **[Intergraph's, 95]**.

O *software GTM-3* é um software de domínio público, composto de aproximadamente 50 ferramentas de *UNIX*. A função do filtro *filter 1D* é filtrar dados de 1 dimensão no domínio do tempo, e a do *filter 2D* é filtrar dados de 2 dimensões no domínio do espaço **[CRSP, 96].**

O *software V-IMAGE REMOTE SENSING FOR WINDOWS* produzido pela *VYSOR*

Integration Inc. trabalha em plataformas *Windows 95 NT 4.0*, além dos filtros convencionais permite que o usuário crie seus próprios filtros através de uma máscara que alcança a ordem de até *15 x 15* **[CRSP, 96].**

No *software MAP II* as operações de filtragem são especificadas pelas operações: Filter <map> [Mask <map>] [LPF] [HPF1] [HPF2] [Sobel] [LAE <value. <value>] {maskSize<value>] [Laplacian] [DiffHoriz] [DiffVet] [DiffDiag]; onde o filtro LPF especifica um filtro passa baixa, o LAE indica que uma máscara menor que 12% das linhas ou colunas de entrada será usada ou então o tamanho será especificado por MaskSize<value> **[Thinkspace, 96].**

O *software GIS by GENASYS* dispõe de opções de filtragem, tal que a operação *FILTER* pode ser usada para detectar bordas, propriedades lineares de imagens de sensoriamento remoto ou de modelos de terrenos. A operação *FILTER* é um operador passa baixa e pode ser usado com outros operadores de janelas móveis tais como *SCAN* e *NINE. SCAN* é uma operação de convolução que se move sobre a imagem do terreno, produzindo uma nova imagem tal que cada pixel da nova imagem contém um valor que é determinado pelo valor do pixel anterior pertencente ao centro da matriz. *NINE* é uma operação de convolução que permite remover ruídos do tipo *salt and pepper*, e que permite também definir um filtro específico **[Consult, 96].**

O *software EASI/PACE* dispõe também de filtros para remover o ruído speckle provenientes de imagens orbitais dos satélites como *RADARSAT, ERS-1* e *JERS-1* **[Threetek, 96].**

O software *ER MAPPER* dispõe de filtros passa baixa, direcionais, geofísicos, gradientes e máscaras que permitem que o usuário defina seus próprios algoritmos de filtragem, armazenados em formato *ASCII*. Os filtros são criados ou editados através de ícones que compõe cada algoritmo **[planetek, 95].**

Os algoritmos de filtragem espacial encontrados nas ferramentas SIG investigadas foram os seguintes:

1. Adobe PlugIn;
2. DIF HORIZ;
3. DIF VERT;
16. Filtro Morfológico Erosão;
17. Filtro Nine;
18. Filtro Procedimento Adaptativo;

4. DIFDIAG;
5. Filtro 1D;
6. Filtro da média;
7. Filtro da Mediana;
8. Filtro da Moda;
9. Filtro de Lee;
10. Filtro Frost;
11. Filtro Gamma MAP;
12. Filtro Gaussiano;
13. Filtro Gradiente
14. Filtro ICM;
15. Filtro Kuan;
19. Filtro Sigma Local;
20. Filtros Morfológicos Dilatação;
21. GRDFilter;
22. HPF1 e HPF2;
23. LAE;
24. LPF;
25. Mediana Máxima e Mínimo;
26. Operadores de Prewitt;
27. Operadores de Roberts;
28. Operadores de Sobel;
29. Operadores Laplacianos;
30. SCAN;

A representação tabular abaixo indica a relação ferramenta SIG versus algoritmo de filtragem espacial presente nas ferramentas investigadas.

Tabela 2.1: Ferramentas Investigadas

Filtros espaciais	Ferramentas investigadas											
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**L**	**M**
Adobe PlugIn						•						
DIF HORIZ							•		•			
DIF VERT							•		•			
DIFDIAG							•		•			
Filtro 1D					•							
Filtro 2D					•							
Filtro da média	•	•			•				•	•	•	
Filtro da Mediana	•	•	•	•	•				•	•	•	•
Filtro da Moda		•							•		•	
Filtro de Lee	•		•								•	•
Filtro Frost	•		•								•	•
Filtro Gamma MAP												•
Filtro Gaussiano									•		•	•
Filtro Gradiente										•	•	
Filtro ICM												•
Filtro Kuan	•										•	•
Filtro Morf. Erosão	•											•
Filtro Nine								•				
Filtro Proc. Adaptativo	•											
Filtros Morf. Dilatação	•											•
GRDFilter								•				
HPF1 e HPF2							•					
LAE							•					
LPF							•					
Mediana Máx. e Mín.						•						
Operad. Laplacianos							•			•	•	•
Operadores de Prewitt	•	•		•							•	•
Operadores de Roberts	•	•		•								•
Operadores de Sobel	•	•		•			•		•		•	•
Editor de Máscaras	•	•		•	•	•					•	
SCAN								•				

Ferramentas: SPRING (A); IDRISI (B); MAI (C); MGE (D); GMT – 3 (E); V – REMOTING SENSING (F); MAP II (G); GIS BY GENASIS (H); EASI PACE (I); ER MAPPER (J); PCI REMOTE SENSING (L); ENVI 2.6 (M).

2.4 Conclusão

Uma análise dos algoritmos de filtragem espacial disponíveis nas ferramentas SIG comerciais investigadas atesta que eles são algoritmos lineares ou não lineares relativamente simples. Esses algoritmos são indicados para executarem o processo de filtragem das imagens multiespectrais em cada banda separadamente.

Este procedimento de filtragem torna o processamento das tarefas de filtragem espacial altamente trabalhoso computacionalmente, principalmente se considerarmos a necessidade de filtrar as imagens dos novos satélites.

Além do mais as máscaras de filtragem associadas a esses algoritmos geralmente são pré-determinadas, o que reflete a baixa adaptatividade de implementação.

Incrementar as ferramentas comerciais SIG com novas alternativas de implementação e novos métodos de filtragem espacial as tornam mais preparadas para acompanhar os recentes avanços da tecnologia de Sensoriamento Remoto, e mais especificamente indicadas para tratar as imagens de alta resolução.

Além disso, a nova geração de Sistemas de Informações Geográficas (OpenGIS) leva em consideração a interoperabilidade de dados e funções de processamento digital de imagens de Sensoriamento Remoto. Isto significa que funções e dados relativos à tarefa de filtragem espacial de imagens deverão ser compartilhadas, oferecendo uma resposta razoável em termos de velocidade de processamento. Neste contexto, a alta densidade dos dados das imagens dos novos sensores orbitais é uma justificativa adicional para o desenvolvimento de novas ferramentas, de alto desempenho, para o processamento de imagens e particularmente de filtragem espacial [Barros98].

O objetivo de realizar este estudo comparativo foi o de selecionar um algoritmo de filtragem espacial para ser implementado em *FPGA* através da metodologia *hardware/software codesign*, visando o seu emprego nas ferramentas SIG comerciais.

Capítulo 3
Análise Computacional dos Filtros Espaciais

O propósito deste capítulo é apresentar uma análise computacional sobre o processamento dos algoritmos de filtragem espacial apresentados no capítulo 2 com o objetivo de ressaltar o esforço computacional envolvido no processamento de uma tarefa de filtragem espacial.

3.1 A Análise da Quantidade de Operações Aritméticas dos Filtros Espaciais Estudados

Em muitas das aplicações das tarefas de filtragem espacial há necessidade de se efetuar grande quantidade de cálculos exigindo geralmente o emprego de avançadas técnicas de processamento. O tempo de resposta que o sistema computacional alcança para realizar uma determinada tarefa de filtragem é uma das principais preocupações deste trabalho. Estamos interessados em aplicar os algoritmos de filtragem espacial, em imagens que manipulam grande quantidade de informações tais como aquelas que manipulam dados da ordem de *terabytes* (como foi apresentado nas Tabelas 1.1 e 1.2 do Capítulo 1).

A análise da quantidade de operações realizada, nesta seção, sobre os algoritmos de filtragem espacial estudados no Capítulo 2 tem o propósito de demonstrar o esforço computacional necessário para o processamento destas tarefas.

3.2 Critérios Empregados na Análise Computacional

O procedimento adotado para efetuar esta análise foi o de investigação dos critérios, abaixo relacionados:

1. Identificação das operações básicas:

   - adições, subtrações, multiplicações e divisões, etc;

2. Avaliação do número de vezes em que estas operações ocorrem;

3. Cálculo da quantidade de operações envolvidas em janelas de tamanho 3×3, 5×5 e 7×7.

Uma vez estipulados esses parâmetros foi então identificado, avaliado e calculada a quantidade de operações existentes nos algoritmos de filtragem espacial, e em seguida, construídas as tabelas comparativas com a finalidade de apresentar os resultados práticos sobre a relação *número de operações/algoritmo/tamanho da imagem.*

Estas análises foram realizadas considerando imagens compostas por pixels de 8 bits. As imagens analisadas são imagens dos satélites comerciais convencionais *GOES/METEOSAT, AVHRR – NOAA, LANDSAT TM.* Estas imagens são usadas como ilustração para referenciar os algoritmos de filtragem espacial **[INPE, 96]**.

3.3 Resultados Obtidos

Os resultados apresentados nesta seção (número de adição/multiplicação) foram obtidos pela aplicação do algoritmo de filtragem espacial em cada banda do sensor do satélite, haja visto que atualmente ainda é comunmente empregado o método de filtragem "por banda".

Esses resultados foram colocados em tabelas. Essas tabelas mostram o resultado da quantidade de operações obtidas através de cálculos efetuados sobre os algoritmos dos filtros espaciais, e o método usado para tratar os elementos da borda da imagem foi o de repeti-los. A tabela 3.1 apresenta as características das imagens usadas como referência nos cálculos.

Tabela 3.1: Características das Imagens

Tamanho da imagem	Número de bits	Nome do satélite	Número de Pixels
512 x 512	8	GOES/METEOSAT	262.144
1024 X 1024	8	AVHRR – NOAA	1 Milhão
2286 x 3600	8	LANDSAT MMS	32 Milhões
8000 x 8000	8	---	64 Milhões

3.3.1 Quantidade de Operações Envolvidas

Na Tabela 3.2 é apresentada a média da quantidade de operações envolvidas nos algoritmos estudados no Capítulo 2. É interessante notar que o número de operações indi-

cada nesta Tabela pode variar, esta variação depende do critério adotado na implementação dos algoritmos.

Os resultados obtidos foram calculados considerando uma janela de tamanho *3 x 3* e considerou-se que as operações de logaritmo, seno, cos-seno, exponencial, e operações estatísticas (ex. mediana) como operações complexas.

Tabela 3.2: Quantidade de Operações dos Filtros Espaciais (por pixel)

Algoritmo	**Número de Adições e Subtrações**	**Número de Multiplicações e Divisões**	**Comparações**	**Número de Operações Complexas**
Mediana	-	-	9	1
Média	8	1	-	-
Permutação Extendida	25	15	9	-
ROM	21	1	8	-
Racional	21	-	8	-
Filtro para Realce usando lógica fuzzy	20	25	-	-
Filtro VMF	18	-	9	9
Filtro RVMF	15	12	18	-
Filtro LMS	9	18	9	-
Estatística Local	70	16	-	-
Sigma	35	12	-	-
Sigma Ponderado	27	7	-	-
Sigma Modificado	45	37	-	-

Uma análise dos resultados obtidos na Tabela 3.3 atesta que a quantidade de operações envolvidas nos algoritmos dos filtros espaciais tendem a crescer com o tamanho da janela usada pelo filtro assim como com o tamanho da imagem.

Para se ter uma idéia da quantidade de operações envolvidas no processo de filtragem foram elaboradas as tabelas 3.3 e 3.4. Os algoritmos selecionados foram um algoritmos linear e um não linear, presentes nas ferramentas comerciais investigadas. A tabela 3.3 e 3.4 indicam a quantidade de operações envolvidas nos algoritmos do filtro da Média e Mediana escalar.

Tabela 3.3: Filtros da Média Janela de Tamanho 7 x 7

operações	512 × 512	1024 × 1024	2600 x 3600	8000 × 8000
+	12 milhões	25 milhões	790 milhões	1.536 milhões
÷	262 mil	1 milhão	32 milhões	64 milhões
	262 mil	1 milhão	32 milhões	64 milhões
TOTAL DE OPERAÇÕES	13 milhões	53 milhões	167 milhões	3264 milhões

Tabela 3.4: Filtro da Mediana Janela de Tamanho 7 x 7

operações	512 × 512	1024 × 1024	2600 × 3600	8000 × 8000
Mediana	256 mil	1 milhão	32 milhões	64 milhões
Comparação	12 milhões	49 milhões	1 trilhão	3 trilhões

3.4 Esforço Computacional dos Algoritmos de Filtragem Espacial

O esforço computacional envolvido no processamento de uma tarefa de filtragem espacial varia significativamente de algoritmo para algoritmo. Para estimar o quanto um computador gasta para realizar um cálculo é necessário saber quanto tempo gasta-se para obter a resposta de cada um dos comandos básicos. Genericamente falando, exceto para instruções de entrada e de impressão, cada comando envolvido obtém acesso a uma ou mais posição na memória, dependendo se a maquina usa comandos de endereço único ou múltiplos, para ler o conteúdo ou armazenar o resultado, e realizar alguma operação de classificação na unidade de aritmética e lógica.

O processamento das tarefas de filtragem espacial envolve muitas operações aritméticas, tais como desvio padrão, mediana, somatórios, etc.., exigindo muitos cálculos da arquitetura usada. Para se obter uma estimativa do tempo empregado para realizar o processamento das tarefas de filtragem espacial faz-se necessário estimar o tempo gasto para cada operação.

Através da análise quantitativa, realizada na Seção 3.4 e 3.5, pode-se verificar que o processamento das operações básicas, devido a quantidade e tipo de operação, limitam o uso destes filtros, mas existem também outros aspectos que contribuem para limitar o desempenho de processamento e em qualquer tecnologia empregada é necessário tratar esses fatores.

- A presença de *loops*, que são procedimentos comuns na implementação dos algoritmos de filtragem espacial, e que tendem a ocupar um tempo significativo da percentagem total do tempo de processamento (nas arquiteturas dedicadas, os *loops* de números de iterações fixas podem ser eliminadas);
- A manutenção do fluxo de entrada e saída dos dados;
- O tratamento dado para as operações de ponto flutuante e o uso de memória para manter tabelas das funções usuais empregadas nestes algoritmos e facilidades para acessá-las;

3.5 Critérios Usados para a Seleção de Um Algoritmo de Filtragem Espacial

Através da análise apresentada neste Capítulo pode-se considerar que os algoritmos de filtragem espacial são computacionalmente complexos, sendo portanto indicado o emprego de arquiteturas especializadas no processamento destas tarefas.

A tendência dos algoritmos de filtragem espacial para as imagens orbitais adquiridas pelos futuros satélites aponta para uma maior sofisticação e complexidade. Estas imagens são compostas por um grande número de pixels e contém um grande número de bandas que fazem com que o processamento de uma dada imagem leve horas ou dias. Isso é valido também para o processamento de imagens em tempo real por causa do tempo limitado dentro do qual o resultado da filtragem tem que estar disponível, assim como é também valido para o processamento de imagens de satélites.

Tendo em vista esta proposição, foi selecionado um algoritmo de filtragem espacial para ser implementado em uma arquitetura reconfigurável. Os parâmetros selecionados para a escolha deste algoritmo de filtragem espacial foram:

- Esforço computacional determinado pelo número de operações, tipos de operações e forma dos operadores envolvidos;
- Nível de aplicação, considerando a grande aplicabilidade do filtro, tais como os tipos de ruídos eliminados e tipos de detalhes preservados;
- Estimativa do tempo de execução para todo o processo de filtragem;
- Custo computacional.

O algoritmo *RVMF* foi selecionado por apresentar todas as características mencionadas acima, acrescido ainda o fato de ser um algoritmo que apresenta propriedades similares ao do filtro da mediana escalar com a vantagem de poder ser aplicado em imagens de multicanais.

3.6 Características do Algoritmo de Filtragem Espacial Selecionado

Como já foi afirmado no Capítulo 1 o presente trabalho visa mostrar a viabilidade e a facilidade de implementar as técnicas de filtragem espacial para as imagens multicanais nas ferramentas SIG empregando arquiteturas reconfiguráveis com FPGA. Tendo este parâmetro em vista e os resultados obtidos na análise computacional, o algoritmo RVMF foi selecionado para ser implementado em FPGA. O motivo desta escolha é apresentado a seguir.

No método vetorial os pixels das janelas do filtro são processados como vetores, em contrapartida ao método de filtragem escalar em que os pixels são processados como escalares. O filtro da mediana Vetorial Reduzido - RVMF propõe uma nova metodologia de filtragem para os sinais vetoriais. A filtragem é realizada simultaneamente nas bandas que compõem a imagem, diferentemente do que ocorre nos métodos usuais, onde o processo de filtragem é aplicado em cada banda da imagem.

O algoritmo do filtro RVMF, realiza inicialmente um mapeamento dos vetores que compõem a imagem em um intervalo escalar, através de uma transformação escalar aplicada em cada vetor que compõe a janela de filtragem. O algoritmo RVMF usa as curvas de preenchimento do espaço para realizar a operação de classificação dos vetores. A primeira operação então é transformar os pixels vetoriais em pixels escalares.

Depois que os pixels são transformados em escalares é realizada a operação de filtragem escalar (o filtro da mediana já foi estudado na Tabela 3.3). Logo após é realizada a operação de anti-transformada dos pixels com valores escalares em pixels na forma de vetores de três coordenadas (*RGB*).

O filtro RVMF apresenta propriedade similar ao filtro da mediana escalar tais como: resposta zero ao impulso, não introduz no processo de filtragem qualquer pixel que não esteja presente na janela de entrada, apresenta bons resultados na suavização do ruído

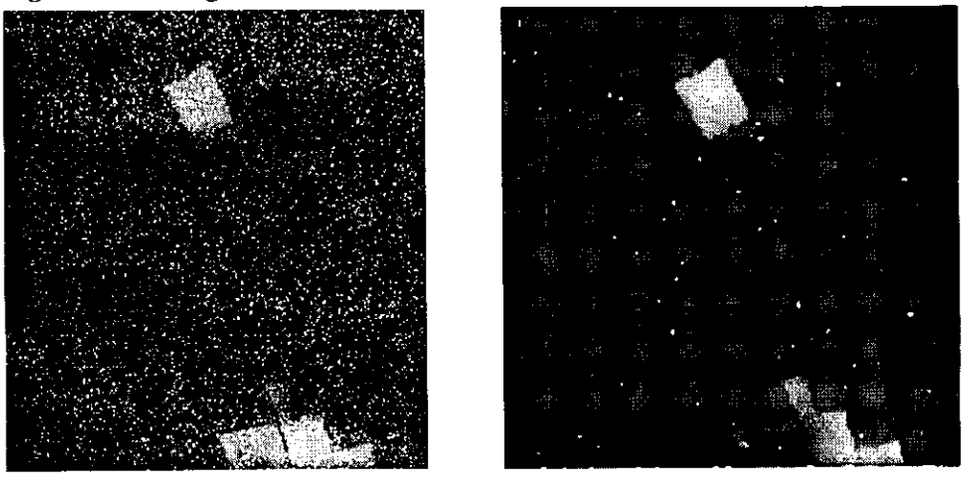
enquanto preserva as bordas **[Astola, 90]** **[Regazzoni, 97]**, sendo portanto adequado para tratar as imagens de Sensoriamento Remoto. Apresenta também a vantagem de se poder trabalhar com o filtro da mediana para o processamento das imagens multicanais usando para o cálculo da mediana multiespectral a mediana monoespectral, acrescentando ao algoritmo um tratamento bem fácil de lidar pois trata pixel por pixel nas janelas de filtragem.

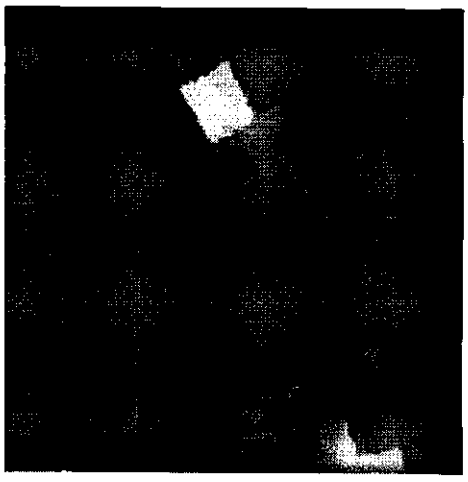
Considerou-se ainda o fato de que atualmente é de grande interesse dispor de novos métodos de filtragem empregando os métodos vetoriais para tratar as imagens multicanais.

Outro critério adotado foi o de considerar que na implementação em FPGA da parte do algoritmo RVMF que executa a transformação do valor escalar em vetor pode eventualmente ser aproveitado para a implementação de filtros de ordem estatística e/ou operadores morfológicos para o processamento de sinais vetoriais.

Os resultados obtidos no processo de filtragem podem ser analisados nas Figura 3.1 à 3.3. O filtro RVMF pode ser aplicado em imagens multiespectaris de qualquer satélite, eg., NOAA, GOES, SPOT. Neste trabalho a imagem selecionada consistiu da cena da região norte do Brasil obtida pelo satélite LANSAT TM compostas das bandas 3, 4 e 5. A simulação do ruído impulsivo foi feita através da ferramenta Koros.

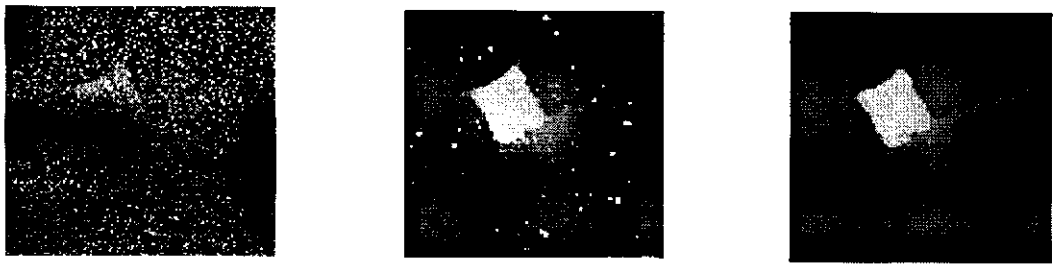
Figura 3.1: Imagem do satélite LANDSAT 229 X 235





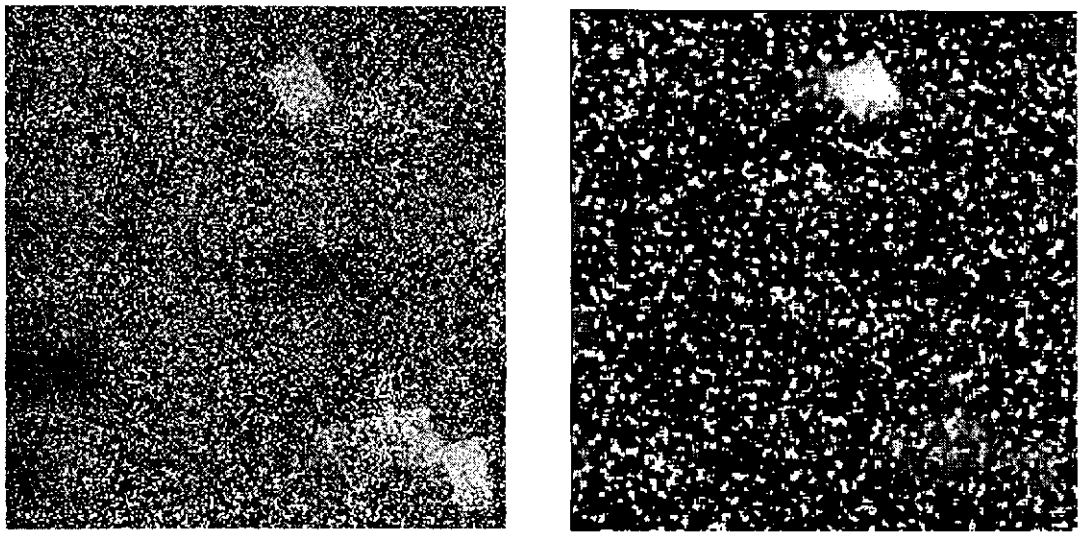
a) imagem com 6% de ruído impulsivo;
b) imagem filtrada com o filtro RVMF primeira iteração;
c) imagem filtrada com o filtro RVMF terceira iteração;

Figura 3.2 Imagem do satélite LANDSAT região norte tamanho 130 x 130



a) imagem com 8% de ruído impulsivo;
b) imagem filtrada com o filtro RVMF primeira iteração;
c) imagem filtrada com o filtro RVMF sexta iteração;

Figura 3.3: Imagem do satélite LANDSAT região norte tamanho 253 x 259



a) imagem corrompida com 20% de ruído impulsivo;
b) imagem filtrada primeira iteração;
c) imagem filtrada após 40 iterações;

3.7 Taxa Média de Processamento das Operações básicas no FPGA da Altera

Motivados pelo fato de que as arquiteturas reconfiguráveis são uma solução altamente viável para serem empregadas nas tarefas de filtragem espacial, foi medido o tempo gasto para realizar as operações básicas, adição e multiplicação, em um *FPGA ALTERA EPF 10K10TC144-3* [**Data Book, 96**]. A tabela 3.5 apresenta os resultados obtidos da medição destas operações no *Timing Analyzer* da ferramenta *MAX + Plus II* versão 8.3 da Altera Corporation.

Tabela 3.5: Operações Avaliadas

Operações	Número de bits	Tempo de Resposta do Operador
Adição	8	18.2 ns
Multiplicação	8	51.6 ns

Foi também calculado por esta ferramenta o tempo estimado, em segundos, para realizar as operações de convolução, e filtragem espacial da média, nas janelas de tamanho 3 × 3, 5 × 5 e 7 × 7. As Tabelas 3.6 e 3.7 indicam os resultados obtidos.

Operação de Convolução

Tabela 3.6: Tempo de Processamento em segundos para diferentes tamanhos de Imagens e Janelas

Tamanho da Imagem	Janela 3 x 3	Janela 5 x 5	Janela 7 x 7
512 x 512	0,12	0,4	0,8
1024 x 1024	0,5	1,4	3,2
2286 x 3600	18,5	52	70
8000 x 8000	37	70	191

Filtro da Média

Tabela 3.7: Tempo de Processamento em segundos da operação de adição no filtro da média para diferentes tamanhos de imagens e janelas

Tamanho da Imagem	Janela 3 x 3	Janela 5 x 5	Janela 7 x 7
512 x 512	0,03	0,1	0,2
1024 x 1024	0,15	0,4	0,9
2286 x 3600	4,8	14	29
8000 x 8000	9,3	28	56

3.8 Conclusão

Na revisão bibliográfica realizada foi averiguado que:

- os algoritmos de filtragem espacial necessitam de grande esforço computacional;
- as imagens provenientes dos futuros satélites são imagens de alta resolução formadas pela composição de várias bandas (por exemplo o sensor GER tem 63 bandas);
- os algoritmos de filtragem espacial presentes nas ferramentas SIG comerciais investigadas usam o método de filtrar uma imagem de satélite banda por banda;
- existem novos métodos de filtragem espacial que empregam o método vetorial (como é o caso do filtro RVMF).

A proposição de técnicas para otimizar o desempenho das tarefas de filtragem espacial nas aplicações de processamento de imagens de alta resolução é de contribuição ímpar para os futuros avanços das técnicas de Sensoriamento Remoto.

No Capítulo 4 a seguir, é apresentada uma descrição da implementação do algoritmo de filtragem espacial RVMF empregando arquitetura reconfigurável a fim de ilustrar a possibilidade de contornar os tópicos acima citados com muito mais eficiência a um custo razoável e com acréscimo no desempenho e na flexibilidade.

Capítulo 4
Hardware/Software Codesign do Filtro da Mediana Vetorial Reduzido

Neste capítulo serão descritos os procedimentos empregados para implementar o algoritmo do filtro da Mediana Vetorial Reduzido, RVMF de filtragem espacial, segundo a metodologia de *Hardware/Software Codesign* e a tecnologia de circuitos FPGA.

4.1 Hardware/Software Codesign

Hardware/Software Codesign significa satisfazer os requerimentos de uma aplicação a ser implementada através da exploração da ação conjunta de componentes de hardware e componentes de software [Micheli, 97]. As componentes de *hardware* são executadas por circuitos digitais específicos, enquanto que as componentes de software são executadas em um processador de propósito geral.

4.2 A Metodologia Hardware/Software Codesign e o Filtro da Mediana Vetorial Reduzido

As etapas adotadas pela metodologia *Hardware/Software Codesign* para implementar o algoritmo RVMF em FPGA são apresentadas a seguir.

4.2.1 Especificação do Algoritmo

Nesta etapa, o algoritmo do filtro RVMF foi amplamente estudado e analisado. Foram determinados os parâmetros de entrada, as funções necessárias e as saídas desejadas. A principal preocupação nesta etapa foi a geração de um fluxuograma geral para o desenvolvimento da implementação. A figura 4.1 apresenta a especificação do algoritmo RVMF.

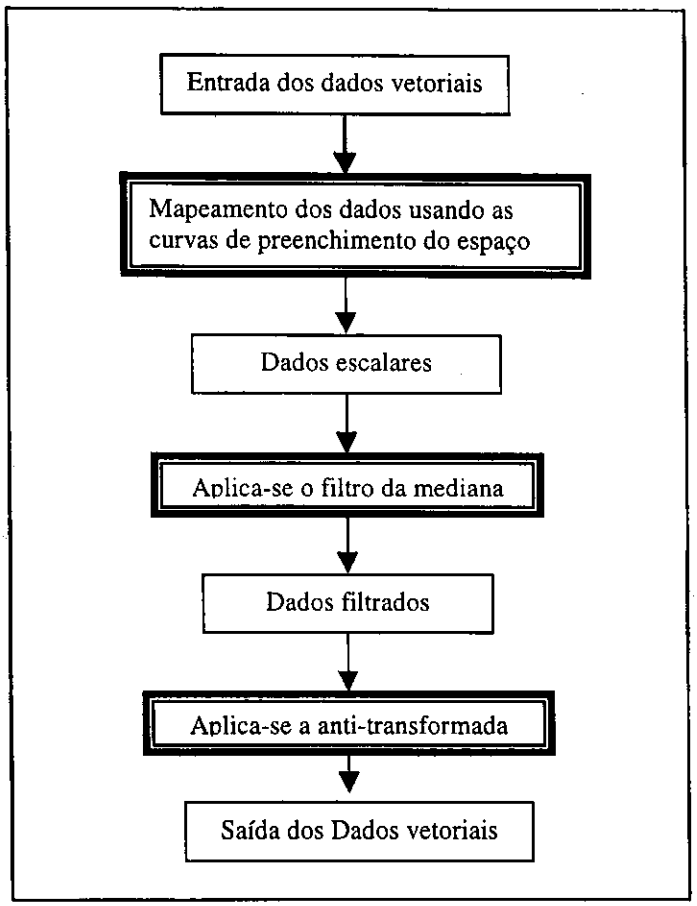


Figura 4.1: Especificação do Algoritmo RVMF

4.2.2 Algoritmo e Implementação

Depois de identificada as propriedades do algoritmo e estabelecidas as relações com os métodos de filtragem, partiu-se para a fase de implementação. O algoritmo foi implementado em linguagem *C*. Primeiramente, foi implementada a transformação da curva de preenchimento do *espaço* para 2 e 3 dimensões. Estas etapas receberam o nome de `gama_12D` e `gama_13D`, transformando os vetores em escalares. Depois que os vetores são tratados como escalares, é aplicada a operação de filtragem da mediana escalar. Posteriormente, é implementada a função inversa, `gama2D` e `gama3D`, que transforma os escalares em vetores.

4.2.3 Partição em hardware/software

Nesta etapa, foi decidida quais as partes do algoritmo que seriam implementadas através de *hardware* e quais as funções que seriam executadas por *software*. Para a seleção das funções a serem implementadas em hardware o critério adotado foi observar

o tempo de processamento em C das funções que compõem o algoritmo RVMF. Este tempo de processamento foi medido num *Pentium II 300 MHz com 128 Mb de RAM*, usando a imagem da região norte do Brasil do satélite LANDSAT cujo tamanho é 560 x 384. Vale observar que neste tempo não está incluído o tempo gasto no acesso do arquivo de dados. A Tabela 4.1 apresenta os resultados obtidos. Vale ressaltar que o tempo de processamento da função `gama_12D` esta incluído na função `gama_13D` (para maiores detalhes ver o Apêndice).

Tabela 4.1 Tempo de Processamento das funções implementadas em C

gama_13D (*Seg*)	mediana (*Seg*)	Gama3D (Seg)
0.6	1.3	0,9

As funções a serem implementadas em Verilog foram escolhidas pelos seguintes critérios:

- viabilidade quanto ao tempo de processamento observado na implementação em C;
- facilidade de implementação em FPGA;
- disponibilidade de blocos funcionais já implementados em Verilog.

A plataforma disponível na UFPB para implementação em FPGA consiste de uma placa inserida no barramento ISA de um PC. Esta placa foi desenvolvida para pré-processamento e filtragem de imagens em tempo real no trabalho de **[Morais, 98]**. Ela pode contar com até dois FPGA's da família 10K da Altera acoplados ao barramento ISA e a barramentos externos para o processamento de imagens digitais.

As funções selecionadas para serem implementadas em *hardware* foram aquelas associadas ás da transformadas `gama_12D` e `gama_13D`, pelos seguintes motivos:

- a função gasta um tempo razoável mesmo não sendo a que gasta maior tempo comparadas as outras funções;
- esta função não necessita de recursos de armazenamento interno dos pixels que estão sendo tratados (cada vetor é transformado em escalar independentemente dos pixels vizinhos);

- estima-se que as funções `gama_12D` e `gama_13D`, cabem facilmente no dispositivo *FPGA ALTERA EPF10K40RC240-4* [Data Book, 96], pois este *FPGA* está montado na placa disponível em [Morais, 98]. Além disso para implementar as funções de módulo, quadrado e cubo necessárias este dispositivo contém células RAM que podem ser usadas como tabelas;

- não encontramos na literatura investigada implementações em FPGA desta função ou de funções parecidas.

Os circuitos de triagem necessárias para a parte da mediana já existem implementados em [Naviner, 97] usando arquiteturas conhecidas [Galisson, 92], e as funções `gama2D` e `gama3D` geram tabelas grandes á qual estima-se que não caiba no *FPGA* disponível no ambiente de trabalho da UFPB.

4.2.4 A Implementação das funções selecionadas

As funções gama_12D e gama_13D foram implementadas em Verilog, resultando em 4 etapas de *pipeline*. Considerou-se ainda o fato de que o algoritmo irá tratar imagens coloridas, ou seja, foram consideradas que as janelas de filtragem serão compostas por pixels com vetores de 3 componentes. As funções gama_12D e gama_13D do programa original C são apresentadas no Apêndice A.1 enquanto que sua descrição em Verilog é apresentada no Apêndice A.2.

4.2.5 Etapas de *Pipeline*

Para cada etapa do *pipeline* as operações usadas foram balanceadas para aproveitar a estrutura física do FPGA. Na Figura 4.2 é apresentada uma estrutura simplificada do elemento lógico básico da família 10K da Altera. Ele contém um bloco combinacional que pode ser programado através de uma tabela verdade para implementar qualquer função lógica de quatro entradas.

A saída deste bloco combinacional é ligada a entrada de um registrador que pode ser contornado por um atalho. Um pipeline otimizado aproveitaria sempre o registrador e não levaria no caso de nosso algoritmo a um grande número de níveis de pipeline, o que tornaria o projeto muito mais complicado. Foi escolhido um limite de 5 etapas de pipeline. Estas etapas foram balanceadas de acordo com as operações usadas na função

`gama_13D`.

Na Tabela 4.2 são listadas as operações executadas em cada etapa do pipelining. São representadas as operações executadas em cadeia e as operações executadas em paralelo dentro de cada etapa. Sendo que as operações em paralelo são listadas separadas por colunas dentro da linha pertencente à respectiva etapa.

Pode-se ver que as operações estão bem distribuídas entre as etapas, com exceção das operações do quadrado (Aq) e do cubo (Ac) na etapa b. Entretanto pensou-se em implementar estas operações em forma de tabelas nas memórias RAM embutidas no FPGA almejado.

Tabela 4.2: Operações Aritméticas usadas nas etapas de pipeline

Operação da etapa de Pipeline	Operações Aritméticas Envolvidas		
a	3 Comparações;		
b	Quadrado	Cubo	- 1 subtração de 8 bits - multiplexação de 18 possibilidades para 4 variáveis
c	- 2 multiplicações com constante 3 - adições de 16 bits e 24 bits, respectivamente		2 comparações de 8 bits
d	1 comparação, 2 multiplicações, 1 multiplicação por constante, 1 adição;		
e	2 adições, 1 multiplicação;		

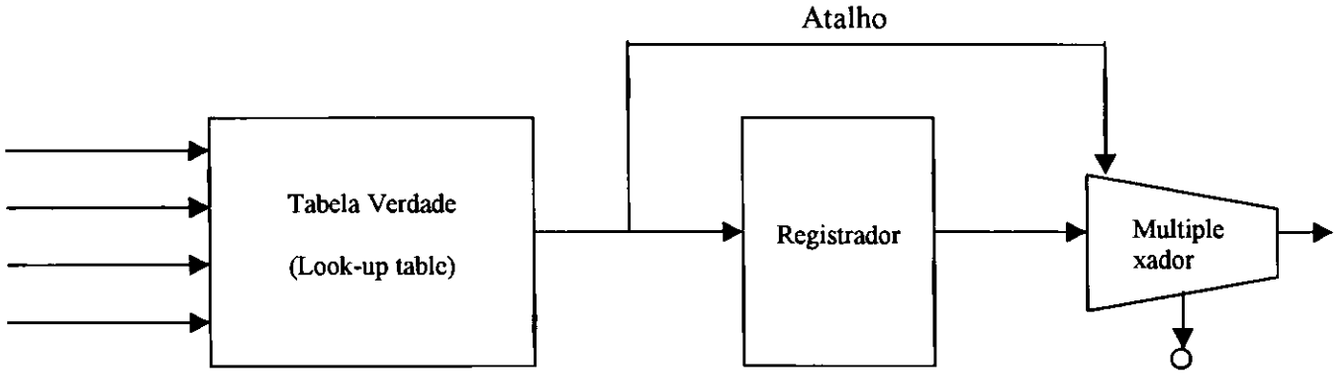


Figura 4.2: Estrutura Simplificada do elemento Lógico Básico da Família 10K da Altera **[Data book, 96]**

4.2.6 Descrição do Algoritmo em Verilog

A descrição em Verilog das funções `gama_12D` e `gama_13D` correspondentes ao comportamento que elas assumem no FPGA é apresentada no Apêndice A.3. As declarações dadas às duas funções em Verilog correspondem às declarações originais usadas em C, o que pode ser comparado analisando o Apêndice A.2 e A.3. Na descrição do comportamento destas funções em Verilog, as funções `gama_12D` e `gama_13D` são transformadas em módulos. Como pode-se observar, no Apêndice A.2, as funções `gama_12D` e `gama_13D` recebem um ponto com 3 coordenadas na função `gama_13D` e 2 coordenadas na função `gama_12D` e determinam o maior valor destas coordenadas. Este valor é usado para calcular o valor inteiro tk, que é o valor de retorno do módulo gama_13D.

As fases do relógio em relação as operações efetuadas em cada etapa do pipelining são representadas na Tabela 4.3.

Tabela 4.3: Etapas do Pipeline e Fases do Relógio

	Relógio fase m	Relógio fase m + 1	Relógio fase m + 2	Relógio fase m + 3	fase m + 4
Etapa (a)	Pixel n (p_x,p_y,p_z) A ← max (p_x,p_y,p_z) P_x1 ← p_x P_y1 ← p_y P_z1 ← p_z	Pixel n + 1	Pixel n + 2	Pixel n + 3	Pixel n + 4
Etapa (b)	Pixel n - 1	Pixel n P1x ← f_a (p_x1,p_y1,p_z1,A) P1y ← f_b (p_x1,p_y1,p_z1,A) Lado Sn Aq ← A^2 Ac ← A^3	Pixel n + 1	Pixel n + 2	Pixel n + 3
Etapa (c)	Pixel n - 2	Pixel n - 1	Pixel n hist ← f_c (Aq, Ac, A1, lado) Sn1 ← sn	Pixel n + 1	Pixel n + 2
Etapa (d)	Pixel n - 3	Pixel n - 2	Pixel n - 1	Pixel n Sn2 ← sn1 Hist1 ← hist T ← gama_2D	Pixel n + 1
Operação (e)	Pixel n - 4	Pixel n - 3	Pixel n - 2	Pixel n - 1	Pixel n K ← f_d (hist1, t, sn2)

Como pode ser observado na tabela acima o processo foi dividido em quatro

etapas. Na fase m é realizada a leitura do vetor que é transferida para a fase seguinte para o estágio de execução. No estágio de execução o valor do vetor (p_x,p_y,p_z), é usado no cálculo de A que é o maior valor das três coordenadas. Os retardos de p_x, p_y, p_z, A e de hist são necessários para conseguir o sincronismo do relógio:

- p_x1 recebe o valor de p_x (na fase m + 1);
- p_y1 recebe o valor de p_y (na fase m + 1);
- p_z1 recebe o valor de p_z (na fase m + 1);
- A1 recebe o valor de A (na fase m + 2);
- hist1 recebe o valor de hist (na fase m + 3).

Os valores Aq e Ac são dados em função de A e refletem a lei do cubo. O operador Aq possui 16 bits, e o operador Ac 24 bits. O valor de p1x e p1y determinam as coordenadas do vetor em 2D na função gama_13D e são calculados em função de p_x1, p_y1, p_z1 e A . O valor do lado é determinado pelo lado do cubo pertencente aos eixos cartesianos XZ, XY ou YZ.

Na fase m + 2 o valor de hist é calculado em função de Ac, Aq, A1 e do lado. Na fase m + 4 o valor de K é dado em função de hist1, sn2 e t, o valor de t é dado em função de gama_12D na fase m + 3. A Figura 4.3 apresenta o fluxo de dados apresentando todos as etapas usadas para determinar o valor do operador K (valor escalar dado em função dos vetores).

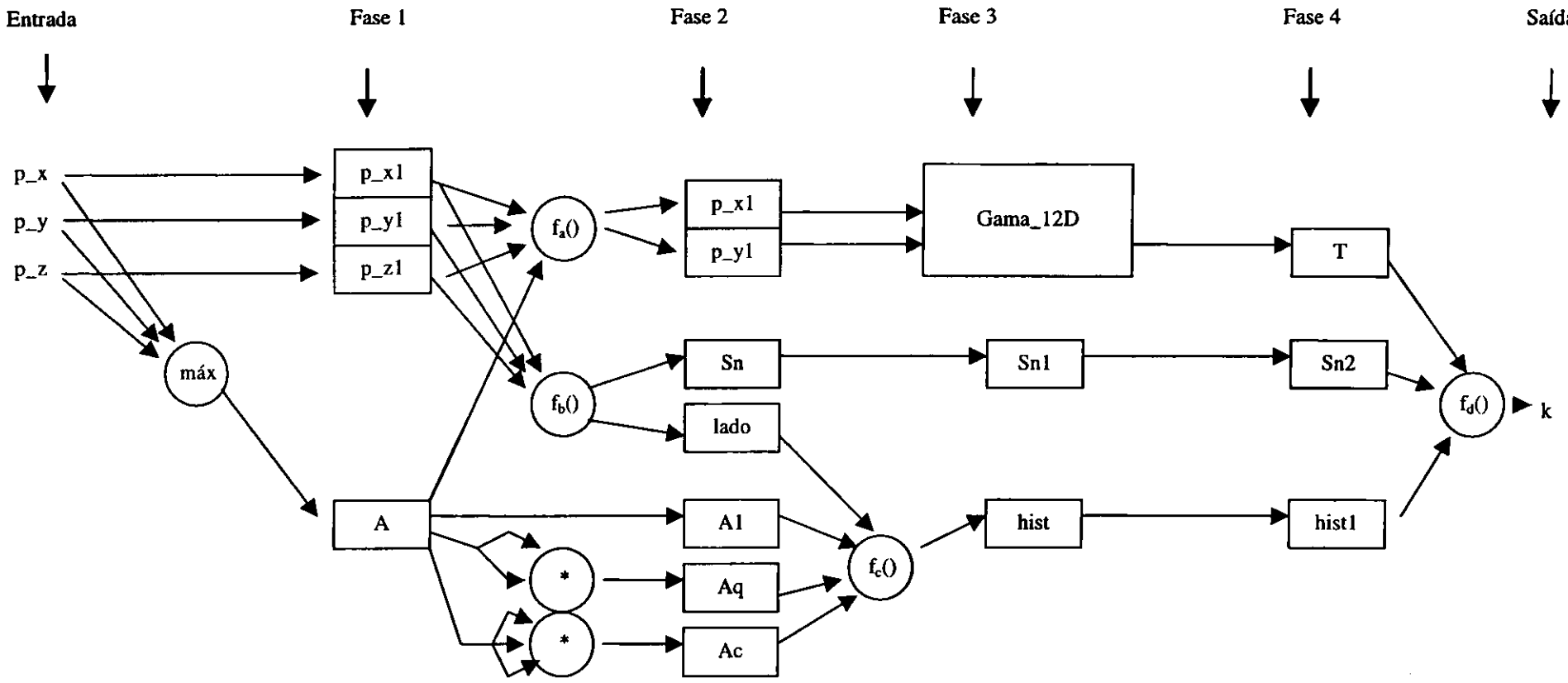


Figura 4.3: Diagrama de Fluxo de Dados para o Cálculo de k

4.2.7 Comportamento dos módulos `gama_12D` e `gama_13D` no `Testbench`

Os módulos `gama_12D` e `gama_13D` foram testados através do *testbench*, `t_gama12D` e `t_gama13D`, apresentados no Apêndice A.4, simulados no simulador Cadence Verilog-XL, e os resultados obtidos estão representados na Figura 4.5.

Na figura 4.2 pode ser observado o seguinte:

Num primeiro momento os dados são trazidos da entrada e através de um retardo no sincronismo do relógio é determinado o valor de A, usando a operação de comparação. O valor de A é determinado como o valor da maior coordenada do vetor (p_x, p_y, p_z). É necessário um retardo no sincronismo do relógio para que as operações de comparação sejam sincronizadas com a etapa seguinte, resultando nos valores p_x1, p_y1, p_z1.

O valor de A é usado para o cálculo de Aq e Ac e do lado. Estes dados são determinados em cada ciclo do relógio e refletem a lei do cubo de preenchimento do espaço. É necessário novamente um retardo no sincronismo do relógio para que as operações de multiplicação, adição e subtração sejam recebidas no tempo exato para o cálculo de cada `p1x` e `p1y`.

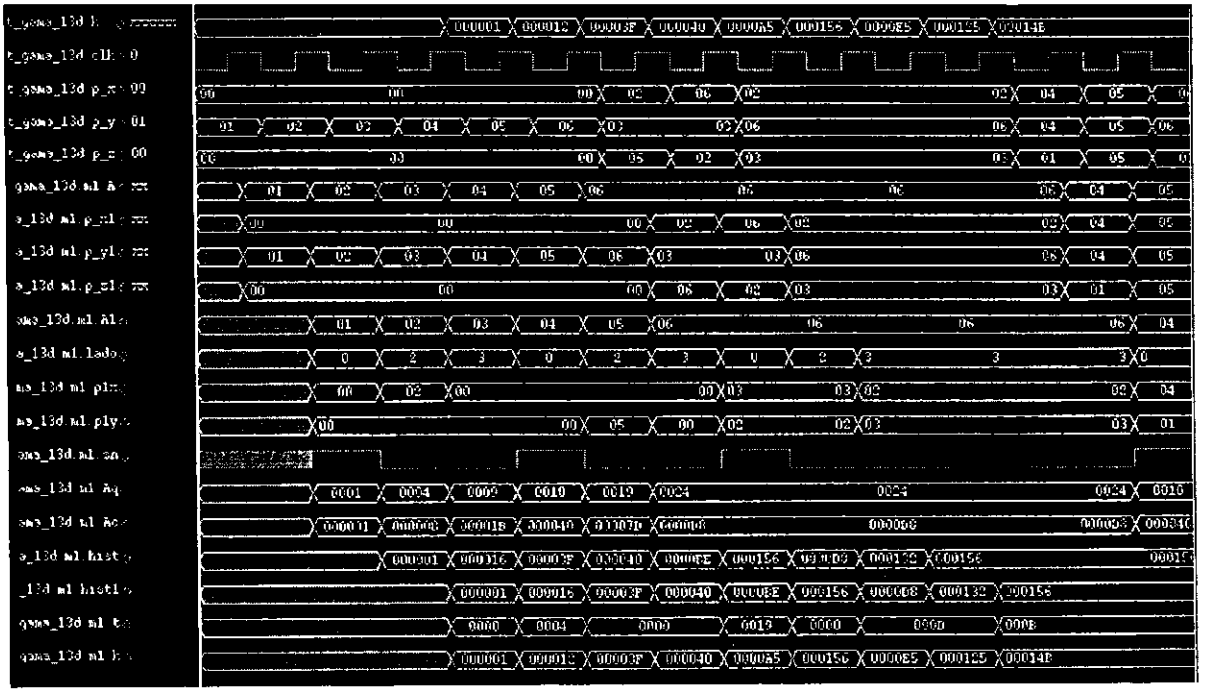


Figura 4.4: Representação das formas de onda dos módulos `gama_12D` e `gama_13D` simulados no *testbench* [Simulador Cadence Verilog – XL]

Os valores de Aq e Ac e do lado são usados para determinar o valor da variável hist. Nesta etapa também é necessário um retardo no sincronismo do relógio para que as operações de multiplicação por escalar sejam recebidas no tempo exato.

Depois é realizado um retardo no sincronismo do relógio para que os valores de hist sejam sincronizados e nesta etapa ocorre uma espera para que o algoritmo examine o que resultou de `gama_12D` para verificar as operações realizadas.

O valor recebido da função `gama_12D` é usado para determinar o valor de k (que é o resultado final). Nesta etapa também é necessário um retardo no relógio para que ocorra o sincronismo dos valores que vão chegando das etapas anteriores.

Depois foi realizada a etapa de validação. Nesta etapa de validação, foram realizados comparações exaustivas entre os resultados fornecidos pelo algoritmo implementado em C e pela implementação em Verilog. Estes testes foram realizados através da simulação de todos os vetores de entrada possíveis (16.777.216 possibilidades) já testados em C, a fim de detectar as possíveis falhas produzidas nos resultados obtidos a partir da observação dos valores de saída.

4.2.8 Mapeamento do FPGA e o Ambiente de Projeto MAX Plus II

Nesta etapa os módulos `gama_12D` e `gama_13D` foram mapeados no *FPGA ALTERA EPF 10K40RC240-4* [Data Book, 96] através da ferramenta *MAX + plus II versão 8.3* da *Altera Corporation*. Esta ferramenta efetua o mapeamento após os módulos terem sido compilados através das seguintes etapas:

1. *Compiler Netlist Extractor*
2. *Database Builder*
3. *Logic Synthesizer*
4. *Partitioner*
5. *Filter*
6. *Timing SNF Extractor*
7. *Assembler*

A etapa 1 gera os arquivos *.cnf*, e se for encontrado erro de sintaxe ocorre uma parada no projeto. Na etapa 2 o sistema para se ocorrer problema na pinagem dos sub-blocos. Na etapa 3 o sistema para se for usado um esquema de relógio incorreto. A etapa 5 gera os arquivos *.rpt* onde obtém-se informações pertinentes ao projeto, tais como recursos utilizados, tamanho da memória usada, etc. A etapa 7 gera os arquivos

.binários para programação. Para obter-se o ciclo e a freqüência do *clock* utiliza-se a função *Timing Analyser.*

4.3 Validação

Depois de verificado que não existia erro, os módulos `gama_12D` e `gama_13D` foram transportados do *Verilog XL da Cadence* para o PC onde a ferramenta *MAX + plus II versão 8.3* da *Altera Corporation* estava instalada.

Nesta etapa foram observadas determinadas características da ferramenta MAX – Plus II que fizeram necessárias adaptações no projeto. Por exemplo os nomes dos módulos devem ser iguais aos nomes dados aos arquivos.

A função módulo 2 e módulo 3 usadas na expressão (A%2*10 + A%3) tiveram que ser mudadas, por que o sintetizador do MAX Plus - II não aceita operador de módulo. O módulo 2 foi então substituído pelo bit menos significativo, A[0], o módulo 3 foi substituído por uma tabela para ser mapeada em bloco de memória RAM embutido no FPGA do tamanho de 256 palavras de 2 bits.

Na fase de elaboração do projeto houve uma preocupação sobre a quantidade de recursos necessários no cálculo de Aq e Ac, mas na fase de mapeamento não houve problemas devida a boa otimização realizada pela ferramenta MAX Plus - II.

4.4 Custos

O projeto do filtro RVMF foi executado por dois projetistas (um com acentuada experiência e o outro estudante) e o tempo gasto para a sua elaboração, incluindo a implementação em linguagem C, foi em torno de seis meses.

Para este projeto pode ser usado um circuito de FPGA, conforme Figura 4.4, ao custo aproximado de 200 dólares. Em um lote de 10 placas o preço unitário de fabricação de cada placa de circuito impresso fica em torno de 130 reais.

4.5 Resultados do Mapeamento no FPGA

Os resultados obtidos no processo de síntese, com relação ao *timing* e a porcentagem de ocupação, *do FPGA EPF 10K40RC240-4* utilizados são mostrados na Tabela 4.5 e 4.6.

Tabela 4.4: Timing obtidos na síntese

Função	Período mínimo do relógio	Freqüência máxima do relógio
Gama_12D	89.6 ns	11.16 Mhz
Gama_13D	136.6ns	7.32 Mhz

Tabela 4.5: Recursos Utilizados obtidos na síntese

Função	Recursos Utilizados
Gama_12D	12%
Gama_13D	64%

A freqüência máxima de funcionamento da função gama_13d é menor que aquela da função gama_12D. Este fato pode ser contornado melhorando a freqüência da função gama_13D para que fique mais próximo ao da função gama_12D. Isto pode ser feito usando tabelas em *RAM* dentro do *FPGA*, para calcular a função do quadrado (A*A) e do cubo (A*A*A) conforme explicada na Seção 4.2.6.

Toda a função gama_12D e gama_13D cabe dentro de um *FPGA* sendo disponível portanto o outro *FPGA* da placa para executar a mediana.

4.6 Conclusão

As arquiteturas reconfiguráveis implementadas através da metodologia *hardware/software codesign* podem ser consideradas como uma tecnologia emergente tida como solução para o processamento de algoritmos que exigem altas demandas de processamento como é o caso do filtro RVMF. O emprego dessas arquiteturas surge como uma nova alternativa para vencer as limitações dos processadores de uso geral. Tendo em vista que as tarefas de Filtragem Espacial são notoriamente tarefas que exigem intenso processamento, tornam-se excelentes candidatas para implantação neste tipo de arquitetura.

Os resultados obtidos neste capítulo servem de diretrizes para a compreensão de como proceder quando se deseja usar arquiteturas reconfiguráveis através da aplicação da metodologia *hardware/software codesign* na implementação das tarefas de filtragem espacial.

Capítulo 5

Conclusão

O presente trabalho mostra uma nova implementação segundo a metodologia *hardware/software codesign* de um algoritmo de filtragem para imagens multiespectrais.

Através dos resultados obtidos demonstrou-se que as arquiteturas reconfiguráveis são uma excelente alternativa para o problema da complexidade computacional da filtragem espacial de imagens de alta resolução.

Mostrou-se ainda a viabilidade da implementação de um procedimento vetorial para filtrar imagens multicanais usando arquiteturas reconfiguráveis

Foram necessários para a elaboração deste projeto conhecimentos de microeletrônica, informática e processamento de imagens. A linguagem de programação usada para a parte do projeto que funciona em software foi C, e a usada para a implementação em *hardware* foi Verilog.

Como resultado final observa-se que este método de implementação para as tarefas de filtragem espacial apresenta meios para se obter maior velocidade, maior flexibilidade e menor custo em relação a métodos tradicionais de programação orientada a instruções usando estações de trabalho ou supercomputadores. Indicada portanto como solução o seu emprego para os novos avanços da tecnologia de Sensoriamento Remoto visando a sua aplicabilidade nas ferramentas de Sistemas de Informações Geográficas.

5.1 Perspectivas para Trabalhos Futuros

Para complementar o trabalho de mestrado descrito e chegar a uma implementação diretamente utilizável por profissionais da área de processamento de imagens, precisam ser realizadas as seguintes tarefas:

- construir uma interface de comunicação entre o ambiente de prototipagem (ferramenta MAX + plus II e a placa de prototipagem usada no trabalho de **[Morais, 98]**) e o PC hospedeiro, inclusive com programação automática dos FPGAs ao iniciar o programa. Este ambiente de prototipagem permitirá a implementação de arquiteturas dedicadas, tais como aquela desenvolvida neste trabalho;
- interface com o usuário aceitando imagens em diferentes formatos padrão;
- inserir o programa dentro de uma ferramenta SIG.

Como trabalho futuro pode-se sugerir a implementação de outros algoritmos de filtragem espacial empregando arquiteturas reconfiguráveis. Pode-se também viabilizar a aplicação de novos métodos de filtragem vetoriais de alta complexidade computacional nas ferramentas de sistemas de informações Geográficas. Ainda podem-se criar novos algoritmos de filtragem espacial que atendam as necessidades dos novos satélites.

Referências Bibliográficas

[Abreu, 96] Eduardo Abreu, Michael Ligstone, Sanjit K. Mitra, Kaoru Arakawa, A New Approach for the Removal of Impulse Noise from Highly Corrupted Images, IEEE Trans. On Image Processing, v.5, Jun 1012-1025, 1996;

[Arnaud, 1992] Arnaud Galisson, Arquitectures VLSI pour algoritmes de TRI, tese de doutorado, Ecole Nationale Supérieure des Telecommunications, Paris, France 1992;

[Athanas, 95] Peter M. Athanas, A. Lynn Abbot, Real Time Image Processing on a Custon Computing Plataform, IEEE Computer, pg 16-24, 1995;

[Aylor 93] Sanjaya Kumar, James H. Aylor, B. W. Johnson, W. A. Wulf, A framework for Hardware/ Software Codesign IEEE Computer, pg 39-45, 1993;

[Araújo, 87] Arnaldo de Albuquerque Araújo, Tese de doutorado em Filtros Espaciais Estudo Comparativo e Aplicação em Classificação e Segmentação de Imagens, Campina Grande, Departamento de Engenharia Elétrica – UFPB, 1987;

[Astola, 90] Jaakko Astola, Petri Haavisto, Vector Median Filters, Proceedings of the IEEE, vol. 78, n.04, pg. 678-689, 1990;

[Barros, 90] Marcelo A. Barros, Filtros Espaciais: Implementação, Estudo Comparativo e Aplicação em Sensoriamento Remoto. Dissertação de Mestrado. UFPB - Campina Grande. 1990;

[Barros, 92a] Marcelo Alves de Barros, Mohamed AKIL. "Study and implementation of a real time programmable convolver with reconfigurable technology", EUROASIC-92, Juin 1992, Paris, France, 192-195;

[Barros, 92b] Marcelo Alves de Barros, Mohamed AKIL. "Vers la synthèse d'architectures pour le traitement bas niveau d'images: un exemple de réalisation sur technologie reconfigurable". Annales de la 1 ère Conference Adequation Algorithmes-Architectures", Lannion, France, Septembre 1992, 67-72;

[Barros, 93a] Marcelo Alves de Barros, Mohamed AKIL. "Circuits Reconfigurables et Traitement Bas Niveau d'Images en Temps Réel". Annales 14ème GRETSI, Juan les Pins, France, Septembre 1993;

[Barros, 93b] Marcelo Alves de Barros, Mohamed AKIL and R. NATOVWICZ. « A reconfigurable architecture for real time segmentation of image se-

quences using self-organizing feature maps ». Proceedings of the IJCNN'93 Internation Joint Conference on Neural Networks Nagoya, Japan, October 1993, pp 69-74;

[Barros, 94a] Marcelo Alves de Barros. « *Traitement Bas Niveau d'Images en Temps Réel et Circuits Reconfigurables* ». Tese de Doutorado em *Architectures de Machines Informatiques Nouvelles*. Université PARIS SUD ORSAY (PARIS XI), Setembro de 1994.

[Barros, 94b] Marcelo Alves de Barros Mohamed AKIL,. « Implementation and Performance Evaluation of an Image Preprocessing Chain on FPGA:». Proc. of the FPL'94 Workshop on Field Programmable Logic and Applications, Prague, Tchecoslovaque, September, 1994.

[Barros, 94c] Marcelo Alves de Barros, Mohamed AKIL. « Low Level Image Processing Operators on FPGA: Implementation Examples and Performance Evaluation ». Proceedings of the 12th International Conference on Pattern Recognition, Jerusalem, Israel, October, 1994;

[Leite, 94] Neucimar Jeronimo Leite, Marcelo Alves de Barros « A Highly Reconfigurable Neighborhood Image Processor based on Functional Programming ». Proceedings of the 1st International Conference on Image Processing, Austin, USA, October 1994;

[Barros, 95] Marcelo Alves de Barros. « A High Level Approach to Design Real Time Image Operators». MIDWEST International Conference on Circuits and Systems, Rio de Janeiro, Brasil, Julho 1995;

[Barros, 96] Marcelo Alves de Barros. Uma Metodologia de Projeto e Implementação de Operadores para Processamento Digital de Imagens em Tempo Real usando Field Programmable Gate Arrays (FPGA). Anais do V Simpósio Brasileiro de Computação Gráfica e Processamento Digital de Imagens, Caxambú, 1996, pp 307-316;

[Barros98] Marcelo A. barros and Milcíades A. almeida and. Crítical factors do spatial data processing in Open Geographic Information Systems. Proceedings of the GISPLANET'98 International Conference and Exhibition on Geographic Information. Lisbon, September, 1998. pp 127,134;

[Barros, 97] Marcelo Alves de Barros, Sistemas de Informações Geográficas. Associação Brasileira de Ensino Superior. Curso de Especialização em Sensoriamento Remoto e SIG, Campina Grande, 1997;

[Berstein, 87] R. Berstein, Adaptative Nonlinear Filters for Simultaneous Removal of Diferrent Kinds of Noise in Images, IEEE Transactions on Circuits and Systems, pg. 1275-1291, 1987;

[Brown, 92] Steplen D. Brown, Robert J. F., J. Rose, Z. G. Vranesic, FPGA, Kluwer Academic Publishes, 1992;

[Camara, 96] Gilberto Câmar, Marco ª Casanova, Andrea S. H., Geovane C. M. e Claudia Maria B. M., Anatomia de Sistemas de Informação Geográfica, IC- UNICAMPI, 8 a 13 julho 1996;

[Camera, 94] Gilberto Camara "Anatomia de um SIG", Fator Gis, n° 04, pg 11 - 15, jan/fev/mar, 1994;

[Candeias, 97] Ana Lúcia Bezerra Candeias, Tese de doutorado em Aplicação da Morfologia Matemática a Análise de Imagens de Sensoriamento Remoto, São Jose dos Campos, INPE, 1997;

[Child 96] Jef Child, Image Processing Architetures take new twists, Computer Design, pg 53-66, 1996;

[Choi, 97] Young Sik Choi, R. Krishnapuram, A Robust Approach to Image Enhancement Based on Fuzzy Logic , IEEE Trans. On Image Processing, v. 6, pg 808-825, 1997;

[Conway, 96] J. Conway, H. Eva, G. D`Souza, "Comparation of deflorested areas using ERS-1 ATRS and the NOAA - 11 AVHRR with reference to ERS-1 SAR data: a case study in the Brazilian Amazon " , Remote Sensing, vol 17, pg 3419 - 3440, 1996;

[Coyle, 89] Edward J. Coyle, Jean-Hsang Lin, Moncef Gabbouj, Optimal Stack Filtering and Estimation and Structural Approach to Image Processing, IEEE Trans. On Acoust, Speech and Signal Processing, v.37, n.12, dez 89;

[Crosta 96] A. P. Crosta. Século XXI em Alta Resolução. Fator GIS, Setembro/Outubro, 1996, pp 14-17;

[CRSP, 97] Endereço eletrônico: http:// www.crsp.ssp.nasa.gov/ssti/welcome.htm, 1997;

[Data Book, 96] Altera Data Book, Altera Corporation, Jun 96;

[Edward 96] M.D. Edwards, Software Acceleration using programable hardware devices;

[Earth Watch, 97] George R. Carlson, Beni Patel, A New Era Draws for Geospatial Imagery, GIS World, Endereço Eletrônico: http//www.geoplace.com;

[Facon 93] Jaques Facon. Processamento e Análise de Imagens. VI Escola Brasileiro- Argentina de Informática. Editora VI EBAI; 1993.

[Filho, 96] Jugurta Lisboa Filho, Cirano Iochpe. Introdução a Sistemas de Informações Geográficas com Ênfase em Banco de Dados. XV Jornada de Atualização em Informática. 1996.

[Foley, 94] James D. Foley, Andries V. Dam, Steven K. Feiner, John F. Hugles, Richard L. Phillips. Introduction to Computer Graphics. Editora Addison- Wesley; 1994.

[Galisson, 92] Arnaud Galisson, Arquitetures VLSI pour algoritmes de Tri, tese de doutorado, Ecole Nationale Superieure des Télácommunications, Paris, França, 1992;

[Garcia, 95] Francilene Garcia e Vladimir Alencar. Sistemas de Informações Geográficas. Fundação Parque Tecnológico Da Paraíba. Nov. 1995;

[Gomes, 94] Jonas Gomes, Luiz Velho, Computação Gráfica: Imagem, IMPA-SBM, editora Grafitex, 1994;

[Gonzales, 82] Rafael C. Gonzales, Paul Wintz. Digital Image Processing. Editora Addison - Wesley, 1982;

[Guccione, 95] Steven Anthony Guccione, Dissertation in Programmin Fine-Grained Reconfigurable Architetures, Facult of the Graduate School of The University at Austin, 1995;

[Hardie, 96] Russel C. Hardie, Kenneth E.Barner, " Extended Permutation Filters and Their Application to Edge Enhancement ", IEEE Transactions on Image Processing, vol.5 , n° 6, jun 1996;

[Heygster, 82] G. Heygster, Rank Filters in Digital Image Processing, Computer Graphics, pg 148-164, 1982;

[IDRISI, 96] Endereço eletrônico: http:// www.clark.university;

[INPE, 96] Endereço eletrônico http:// www. ltid.inpe.br/html/obt.html;

[INPE] Endereço eletrônico http:// www.dgi.inpe.br/html/landsat.htm;

[Inc, 94] Cadence Design Systems INC, "Verilog – XL Reference Manual 2.0", 1994;

[Ismail 94] Tarek Ben Ismail, A. Jerraya, Tutorial: Hardware/ Software Codesign, Tima/ INPC Laboratory, France;

[Kotropoulos, 96] Constantine Kotropoulos, Ioannis Pitas, Adaptative LMS L-Filters for Noise Suppression in Images, IEEE Trans. On Image Processing, v.5, 1996;

[Louis 90] J. E. Louis, Jr Galbiati, Machine Vision and Digital Image Processing Fundamentals Editora; 1990;

[Lulich, 96] Dan W. Hammerstrom Daneil P. Lulich, Processamento de Imagens Usando Processador Array de 1 dimenssão, 1996;

[Maragos, 96] Petros Maragos, Differential Morphology and Image Processing, vol. 5, pg. 922-936, junho, 1996;

[Mascarenhas, 96] Nelson D. A Mascarenhas, An Overview of Speckle noise Filtering in SAR Images, Buenos Aires, dezembro 2/4, 1996;

[Morais, 98] Marcos Ricardo Alcântara Morais, Sistema de Prototipagem Rápida Dirigido ao Processamento de Imagens, Dissertação de mestrado, Campina Grande, Pos Graduação em Engenharia Elétrica – UFPB, outubro 1998;

[Micheli, 97] Giovanni De Michelli, R. K. Gupta, *Hardware/Software Codesign*, Proceedings of IEEE, vol. 85, n. 3, pg 349 - 365, 1997;

[Naviner, 97] Elmar Melcher, Lírida Naviner, João Marques de Carvalho, Jean François Naviner, Ricardo ª S. Moreira, Yuri de Mello Villar, Marco

Morais, VLSI Implementation of Contourn Extration from Real Time Image Sequences VLSI, 1997;

[Novo, 93] Evyn M.L. de Moraes Novo. Sensoriamento Remoto Princípio e Aplicações. Editora Edgard Blücher Ltda; 1993;

[Paradella, 96] Waldir Renato Paradella, "Imagens de Radar Fundamentos e Experiências com o SAR na Amazonia" , Fator GIS, n° 14, jun/jul 1996;

[Paredes, 94] Evaristo A. Paredes. Sistema de Informação Geográfica - Princípios e Aplicações. Editora Èrica, 1994;

[Pratt, 78] W. K. Pratt, Digital Image Processing , John Wiley & Sons, 1978;

[Ramponi, 96] Giovanni Ramponi, The Rational Filter for Smoothing, IEEE Signal Processing Letters, v. 3, pg 63-65, março, 1996;

[Ramponi, 97] G. Ramponi, C. Moloney, Smoothing Speckled Images Using na Adaptative Rational Operator, IEEE Signal Processing Letters, v04, pg 68-71, março, 1997;

[Rencher, 96] Michael A. Rencher , A Comparison of FPGA Platforms Through SAR/ATR Algoritm Implementation, Brigham Yong University, dez. 1996;

[Regazzoni, 97] Carlo S. Regazzoni, Andrea Teschioni, A New Approach to Vector Median Filtering Based on Space Filling Curves, IEEE Transactions on Image Processing, vol. 6, n.7, pg. 1025-1037,july, 1997;

[Schulze, 95] Mark A Schulze, Aing X. Wu, Noise Reduction in Synthetic Aperture Radar Imagery Using a Morphology - Based Nonlinear Filter, Digital Image Computing Tchinics and Applications, Conference of the Australian Pattern Recognition Society, 6/8 Dezembro, 1995;

[Smith, 96] D. Smith, Speckle reduction and Segmentation of Synthetic Aperture Radar Images, International Journal of Remote Sensing, vol. 17, n.11, pg. 2043-20-58,july 1996;

[Tanenbaum, 92] Andrew S. Tanenbaum , Organização Estruturada de Computadores. Editora Prentice Hall do Brasil, 1992;

[Tei, 95] "Qual a Melhor Definição de SIG", Fator Gis, n.11, pp. 20- 24, out/nov/dez 1995.

[Venetsanopoulos, 97] K. N. Plataniotis, D. Androutsos, S. Vinayagamoorthy and ª N. Venetsanopoulos, Color Image Processing Using Adaptative Multichannel Filters , IEEE Transactions on Image Processing, vol. 6, n. 7, pg. 933-949, july 1997;

[Vuilleman96] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. H. Touati and P. Boucard. « Programmable Active Memories: Reconfigurable Systems Come of Age » . IEEE Trans. on VLSI Systems, Vol. 4, N° 1, March 1996;

[Villar, 97] Yuri de Mello Villar, Projeto de um ASIC para extração de vértices de imagens em tempo real, Dissertação de mestrado, Campina Grande, Departamento de Engenharia Elétrica – UFPB, dezembro de 1997;

[Weinhardt 95] Markus Weinhardt Integer Programming for Partioning in Software oriented Codesign, Field Logic and Applications: 5 th Intern. Workshop aug/set 1995;

[Weinhardt 95] Markus Weinhardt Portable Pipiline Syntheis for FCCMs, Field Logic and Applications: 5 th Intern. Workshop and apllications 1996;

Apêndice
Algoritmos Implementados

A . 1 Programa das Funções gama_12D e gama_13D em C

```
/*#define max(a,b)      (a)>(b)?(a):(b)*/
 int gama_12D(struct ponto p)
            int p1;
            int A       = max(p.x,p.y);
            int hist= A*A;

            if ( A%2 == 0 )
                  if ( A == p.x )
                        p1 =  p.y;
                  else
                        p1 = 2*A-p.x;
            else
                  if ( A == p.y )
                        p1 =  p.x;
                  else
                        p1 = 2*A-p.y;

            return hist+p1;

 int gama_13D(struct ponto p)
{
      int   A     = max(p.x,max(p.y,p.z));
      int   hist= A*A*A;
      int   sn    = -1;
      int         n     = A+1;
      struct ponto p1;
      switch( A%2*10+A%3 )
      {
            case  0:
                  if ( A == p.z )
                  {
                              p1.x = p.y, p1.y = p.x;
                              sn   = 1;
                  } else
                  if ( A == p.x )
                  {
                              p1.x = p.z, p1.y = A - p.y;
                              hist+= 2*n*n - n -1;
                  } else
                  if ( A == p.y )
                  {
                              p1.x = p.z, p1.y = p.x;
```

```
                    hist+= 3*n*n - 3*n ;
        }
        break;
case    1:
        if ( A == p.y )
        {
                    p1.x = p.x, p1.y = p.z;
                    sn   = 1;
        } else
        if ( A == p.z )
        {
                    p1.x = A - p.x, p1.y = p.y;
                    hist+= 2*n*n - n -1;
        } else
        if ( A == p.x )
        {
                    p1.x = p.z, p1.y = p.y;
                    hist+= 3*n*n - 3*n ;
        }
        break;
case    2:
        if ( A == p.x )
        {
                    p1.x = p.z, p1.y = p.y;
                    sn   = 1;
        } else
        if ( A == p.y )
        {
                    p1.x = A - p.z, p1.y = p.x;
                    hist+= 2*n*n - n -1;
        } else
        if ( A == p.z )
        {
                    p1.x = p.y, p1.y = p.x;
                    hist+= 3*n*n - 3*n ;
        }
        break;
case    10:
        if ( A == p.z )
        {
                    p1.y = p.y, p1.x = p.x;
                    sn   = 1;
        } else
        if ( A == p.x )
        {
                    p1.y = p.z, p1.x = A - p.y;
                    hist+= 2*n*n - n -1;
        } else
        if ( A == p.y )
        {
                    p1.y = p.z, p1.x = p.x;
                    hist+= 3*n*n - 3*n ;
        }
        break;
case    11:
        if ( A == p.y )
        {
```

```
                        p1.y = p.x, p1.x = p.z;
                        sn   = 1;
                } else
                if ( A == p.z )
                {
                        p1.y = A - p.x, p1.x = p.y;
                        hist+= 2*n*n - n -1;
                } else
                if ( A == p.x )
                {
                        p1.y = p.z, p1.x = p.y;
                        hist+= 3*n*n - 3*n ;
                }
                break;
        case  12:
                if ( A == p.x )
                {
                        p1.y = p.z, p1.x = p.y;
                        sn   = 1;
                } else
                if ( A == p.y )
                {
                        p1.y = A - p.z, p1.x = p.x;
                        hist+= 2*n*n - n -1;
                } else
                if ( A == p.z )
                {
                        p1.y = p.y, p1.x = p.x;
                        hist+= 3*n*n - 3*n ;
                }
                break;
    }
    return hist + sn*gama_1(p1) ;

}
```

A . 2 Implementação das Funções gama_12D e gama_13D em Verilog

```
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// Modulo de descricao da funcao que representa a transformada gama_12D
// junho 1998;

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

module gama_12d(t,clk,p_x,p_y); // transforma  um vetor de 2D
em um escalar

      input  [7:0]p_x,p_y; // o ponto e formado de 2 coordenadas
      input  clk;
      output t;
      reg [15:0] t;
      reg [7:0] A;
      reg [7:0] p_xr,p_yr;
      reg xmaior;



      always @(posedge clk)
       begin

             if (p_x > p_y)
               begin

                  A <= #2 p_x;
                  xmaior <= #2 1;

               end
             else
               begin

                  A <= #2 p_y;
                  xmaior <= #2 0;

               end


             p_xr <= #2 p_x;
                   p_yr <= #2 p_y;


          end

        wire [15:0] #2 hist = A*A;



      always @ (posedge clk)
        begin

             //se o resto da divisao de A por 2 for zero entao...

             if (A%2 ==0)
```

```
                begin
                  if (xmaior)

                    t <= #2 (hist + p_yr); // retorna um valor escalar;

                 else

                      t <= #2 (hist + 2*A - p_xr);  // retorna um valor
escalar;
                 end


// se o resto da divisao de A por 2 for  diferente de zero entao

              else if (!xmaior)
                  t <= #2 (hist + p_xr);

                else

                  t <= #2 (hist + 2*A - p_yr);   // retorna um valor
escalar;

            end


endmodule
//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
// Modulo 3D para transformar um vetor em escalar;
// junho 1998;
// o modulo gama3D  transforma 1 vetor de 3D em um escalar baseado nas
//propriedades da curva de preenchimento do espaco;


`include "gama_12d.v"

// o comportamento desta curva em 3D pode ser visto como uma curva que
//atravessa todos os pontos que possuem o mesmo valor da norma do
//max(distancia do vetor) de modo continuo atravessando todos os lados do
//cubo. Entao o modulo  abai//xo calcula o valor escalar cobrindo os 3
//lados do cubo, tendo a norma do max
//o valor de A.

// p_x, p_y e p_z sao as coordenadas do  ponto e k o valor do  escalar :

module gama_13D(k,clk,p_x,p_y,p_z);
// declaracao das entradas, saidas e argumentos necessarios:


      input [7:0] p_x,p_y,p_z;
      input clk;
      output [23:0] k;
      reg [7:0] A,A1,p_x1, p_y1, p_z1, p1x,p1y;
      reg [15:0] Aq;
      reg[23:0] Ac,hist,hist1;
      wire [15:0] t;
// o valor de t e determinado pela funcao de gama2D
      reg sn,sn1,sn2;
```

```
      reg xmaior,ymaior,zmaior;
// o lado do cubo e obtido em funcao dos eixos
// das coordenadas x,y e z;
      reg [1:0] lado; // o lado do cubo - pode ser 0, 2, ou 3



      assign  k = sn2 ? hist1+t : hist1-t;


// a funcao gama2D e referenciada pois associa um valor escalar( que
//representa o comprimento do arco da curva do espaco filling)a um vetor



      gama2D U1(t,clk,p1x,p1y);



// o valor de A e determinado como o max de p_x,p_y,p_z,pois o maior
//valor de k esta associado com o vetor que apresenta maior distancia da
//origem (dada pela norma do max). Para calcula-la usa-se a seguinte
//funcao:

// os valores encontrados aqui devem casar com a escolha do ponto;

// a relacao usada para encontrar o maior valor foi considerar as 8
//situacoes que podem ocorrer :

// xmaior   ymaior            zmaior
//   0              0            0          ----------------------
//   0              0            1          p_x == p_y  && p_y <= p_z
//   0              1            0          p_x != p_y     p_x == p_z  p_y >
p_z
//   0              1            1          p_x != p_y     p_x != p_z
//   1              0            0          p_x != p_y     p_x != p_z  p_x >
p_z
//   1              0            1          p_x != p_y     p_x == p_z  p_y <
p_z
//   1              1            0          p_x == p_y     p_y > p_z
//   1              1            1          p_x == p_y     p_y == p_z

         always @ (posedge clk)

                  if( p_x == p_y )

                      begin  // 111, 110, 001:

                        if (p_y == p_z)
                               begin   // 111:
                              A <= #2 p_x;
                             xmaior <=#2 1;
                              ymaior <=#2 1;
                              zmaior <=#2 1;
                            end

                        else  // 110, 001:
```

```
                if (p_y > p_z)
                  begin
                A <= #2 p_y;
                xmaior <=#2 1;
                ymaior <=#2 1;
                zmaior <=#2 0;
                   end

                 else
                   begin   // 001:
                      A <= #2 p_z;
                xmaior <=#2 0;
                ymaior <=#2 0;
                zmaior <=#2 1;
                   end

            end

        else   // 010, 011, 100, 101,001:

              if (p_x==p_z )

                begin  // 010, 101:

                    if (p_y > p_z)
                      begin   // 010:
                        A <= #2 p_y;
                        xmaior <=#2 0;
                        ymaior <=#2 1;
                        zmaior <=#2 0;
                      end

                    else
                      begin  // 101:
                         A <= #2 p_z;
                         xmaior <=#2 1;
                         ymaior <=#2 0;
                         zmaior <=#2 1;
                      end
                    end

                  else

                    begin   // 011, 100, 010, 001:

                     if( p_y == p_z)
                       begin   // 011, 100:
                            if (p_y > p_x)
                              begin
                               A <= #2 p_y;
                               xmaior <=#2 0;
                               ymaior <=#2 1;
                               zmaior <=#2 1;
                            end

                          else
```

```
                                    begin  // 100:
                                      A <= #2 p_x;
                                      xmaior <=#2 1;
                                      ymaior <=#2 0;
                                      zmaior <=#2 0;
                                    end
                                  end
                                else
                                  begin
                                    if (p_x > p_y & p_x > p_z)
                                         begin  // 100:
                                        A <= #2 p_x;
                                        xmaior <=#2 1;
                                        ymaior <=#2 0;
                                        zmaior <=#2 0;
                                         end
                                    else
                                      if (p_y > p_x & p_y > p_z)
                                           begin  // 100:
                                          A <= #2 p_y;
                                          xmaior <=#2 0;
                                          ymaior <=#2 1;
                                          zmaior <=#2 0;
                                           end

                                        else
                                             begin  // 001:
                                            A <= #2 p_z;
                                            xmaior <=#2 0;
                                            ymaior <=#2 0;
                                            zmaior <=#2 1;
                                             end

                                    end


                            end



// retardos de sn, A, p_x, p_y, p_z para conseguir sincronismo

      always @(posedge clk)
         begin

               p_x1 <= #2 p_x;
               p_y1 <= #2 p_y;
               p_z1 <= #2 p_z;
           sn1 <= #2 sn;
           sn2 <= #2 sn1;


          end
```

```
// os parametros abaixo sao dados em funcao de A que reflete a lei do
cubo:
      always @(posedge clk)
         begin


            A1 <= #2 A;
            Aq <= #2 (A*A);
                  Ac <= #2 (A*A*A);
                  hist1 <= #2 hist;

//O termo hist aumenta de acordo com a norma do max. O valor de hist e
determi//nado assim:

                if (lado==0)
                   hist <= #2 Ac;
                 else if (lado==2)
                    hist <= #2 (Ac+2*Aq+3*A1);
                 else if (lado==3)
                    hist <= #2 (Ac+3*Aq+3*A1);



           end

// No Verilog-XL, usamos o operador de modulo
      wire [1:0] Amod3 = A%3;


// o valor escalar k e obtido da relacao que associa a cada vetor  sua
//norma do max. Esta relacao e composta pela historia da curva acima da
//camada a qual ela se situa e o deslocamento da curva dentro da propria
//camada.
// As relacoes abaixo correspondem a trajetorias seguidas pela curva em
//diferentes lados do cubo. Para calcular o valor de k usa-se a relacao
//do modulo. O comportamento da curva em cada lado do cubo e considerado
//o mesmo seguido pela curva no espaco de 2D:
       always @ (posedge clk)
          begin

            case (A[0]*10 + Amod3)
// quando o valor de A e um numero que e divisivel por 2 e
3(0,6,12,18...)
            0:

              if (zmaior)
                 begin

//p1x e p1y determinam as coordenadas do vetor em 2D
                       // $display("caso 0  1o if");
                          p1x <= #2 p_y1;
                        p1y <= #2 p_x1;
                        lado <= #2 0;
                          sn  <= #2  1;
                   end

                   else if (xmaior)
```

```
                        begin

                         // $display("caso 0  2o if");
                           p1x <= #2 (A - p_y1);
                           p1y <= #2 p_z1;
                           lado <= #2 2;
                           sn <= #2  0;

                        end

                 else              //  caso  A==p_y;
                 begin

                        // $display("caso 0  3o if");
                     p1x <= #2 p_x1;
                     p1y <= #2 p_z1;
                     lado <= #2 3;
                     sn <= #2  0;

                 end
// quando o valor de A e 4,10,16...

              1:

                if (ymaior)
                begin

                     // $display("caso 1  1o if");
                     p1x <= #2 p_x1;
                     p1y <= #2 p_z1;
                     lado <= #2 0;
                     sn <= #2  1;

                  end

               else if (zmaior)
                          begin

                           // $display("caso 1  2o if");
                             p1x <= #2 (A - p_x1);
                             p1y <= #2 p_y1;
                             lado <= #2 2;
                             sn <= #2 0;

                           end

               else                // caso (xmaior)
                 begin

                          // $display("caso 1  3o if");
                    p1x <= #2 p_z1;
                      p1y <= #2 p_y1;
                    lado <= #2 3;
                    sn <= #2  0;

                 end
```

```
// quando o valor de A e um numero que e divisivel por 2 e quando
//dividido por 3 reste 2 ( 8,2...)

              2:

               if (xmaior)
                 begin


                      // $display("caso 2  1o if");
                      p1x <= #2 p_z1;
                      p1y <= #2 p_y1;
                      lado <= #2 0;
                      sn <= #2  1;

                    end

                else if (ymaior)
                           begin

                             // $display("caso 2  2o if");
                              p1x <= #2 (A - p_z1);
                              p1y <= #2 p_x1;
                              lado <= #2 2;
                              sn <= #2  0;

                           end

                else                          //caso  (zmaior)
                  begin

                           // $display("caso 2  3o if");
                      p1x <= #2 p_y1;
                      p1y <= #2 p_x1;
                      lado <= #2 3;
                      sn <= #2  0;

                  end
// quando o valor de A e 3,9...

                10:

                  if (zmaior)
                 begin

                      // $display("caso 10  1o if",A);
                      p1x <= #2 p_x1;
                      p1y <= #2 p_y1;
                      lado <= #2 0;
                      sn <= #2  1;

                    end

                else if (xmaior)
                           begin

                             // $display("caso 10  2o if");
```

```
                        p1x <= #2 p_z1;
                        p1y <= #2 (A - p_y1);
                        lado <= #2 2;
                        sn <= #2  0;

                    end

            else                        // caso (ymaior)
              begin

                    // $display("caso 10  3o if");
                p1x <= #2 p_z1;
                p1y <= #2 p_x1;
                lado <= #2 3;
                sn <= #2  0;

              end

// quando o valor de A e

           11:

              if (ymaior)
               begin

              // $display("caso 11  1o if");
                  p1x <= #2 p_z1;
                  p1y <= #2 p_x1;
                  lado <= #2 0;
                  sn <= #2  1;

               end

             else if (zmaior)
                       begin

                           // $display("caso 11  2o if");
                     #1p1x = p_y1;
                           #1p1y = (A - p_x1);
                           lado <= #2 2;
                           sn <= #2  0;

                       end

             else                              // caso  (xmaior)
               begin

                    // $display("caso 11  3o if");
                    p1x <= #2 p_y1;
                    p1y <= #2 p_z1;
                    lado <= #2 3;
                    sn <= #2  0;

               end

// quando o valor de A e
           12:
```

```
                if (xmaior)
               begin

                     // $display("caso 12  1o if");
                    p1x <= #2 p_y1;
                    p1y <= #2 p_z1;
                    lado <= #2 0;
                    sn <= #2  1;


                 end

              else if (ymaior)
                         begin

                       // $display("caso 12  2o if");
                             p1x <= #2 p_x1;
                             p1y <= #2 (A - p_z1);
                             lado <= #2 2;
                             sn <= #2  0;

                         end

              else                              // caso (zmaior)
                begin

                     // $display("caso 12  3o if");
                    p1x <= #2 p_x1;
                    p1y <= #2 p_y1;
                    lado <= #2 3;
                    sn <= #2  0;
                end
             endcase


        end

endmodule
```

A.3 Implementação dos Módulos de Teste das Funções gama_12D e gama_13D em Verilog

```
      // o modulo gama3D transforma 1 vetor de 3D em um escalar;


// testebench
// testa a funcao gama para vetores de 3 dimensoes:



`include "t3dd.v"


module test_gama;


// declaracao das portas do projeto gama


 reg clk;
 reg [7:0] p_x,p_y,p_z;
 reg [7:0] p1x,p1y;
 wire [31:0] k;



      //`define debug
$display(m1.p_x,m1.p_y,m1.p_z,m1.A,m1.Aq,m1.Ac,m1.lado,//m1.p1x,m1.p1y,m1
.t,m1.k);
        `define debug
       // `define dpl  $display(p_x,p_y,p_z,k);
       ` define t_gama_13D.v $display(t_gama_13D.v)
      `define dpl  $display(k);
// instanciando o modulo gama3D

      gama_13D m1(k,clk,p_x,p_y,p_z);

      always
        #5 clk = ~clk;



      always @(posedge clk);
      initial
        begin
        $cw_display(k,clk,p_x,p_y,p_z);
        $cw_display(m1.A,m1.p_x1,m1.p_y1,m1.p_z1);
        $cw_display(m1.A1,m1.lado,m1.p1x,m1.p1y, m1.sn);
        $cw_display(m1.Aq,m1.Ac,m1.hist);
        $cw_display(m1.hist1,m1.t,m1.k);
```

```
clk = 0;

  p_x=2; p_y=3; p_z=6; #5 'dpl 'debug #5
  p_x=6; p_y=3; p_z=2; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5

  p_x=4; p_y=4; p_z=1; #5 'dpl 'debug #5
  p_x=5; p_y=5; p_z=5; #5 'dpl 'debug #5
  p_x=6; p_y=6; p_z=1; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5
  p_x=2; p_y=6; p_z=3; #5 'dpl 'debug #5

  p_x=4; p_y=2; p_z=4; #5 'dpl 'debug #5
  p_x=5; p_y=3; p_z=5; #5 'dpl 'debug #5
  p_x=6; p_y=6; p_z=6; #5 'dpl 'debug #5
  p_x=1; p_y=0; p_z=1; #5 'dpl 'debug #5
  p_x=9; p_y=2; p_z=9; #5 'dpl 'debug #5
  p_x=3; p_y=3; p_z=3; #5 'dpl 'debug #5

  p_x=6; p_y=3; p_z=6; #5 'dpl 'debug #5
  p_x=1; p_y=0; p_z=1; #5 'dpl 'debug #5
  p_x=9; p_y=2; p_z=9; #5 'dpl 'debug #5
  p_x=9; p_y=9; p_z=9; #5 'dpl 'debug #5



p_x=4; p_y=4; p_z=0; #5 'dpl 'debug #5
p_x=7; p_y=7; p_z=0; #5 'dpl 'debug #5
p_x=9; p_y=0; p_z=9; #5 'dpl 'debug #5
p_x=3; p_y=2; p_z=3; #5 'dpl 'debug #5
  $display();
  p_x=1; p_y=1; p_z=0; #5 'dpl 'debug #5
p_x=8; p_y=8; p_z=8; #5 'dpl 'debug #5
p_x=2; p_y=4; p_z=1; #5 'dpl 'debug #5
p_x=4; p_y=2; p_z=1; #5 'dpl 'debug #5
  $display();
p_x=1; p_y=2; p_z=4; #5 'dpl 'debug #5
p_x=2; p_y=4; p_z=3; #5 'dpl 'debug #5
p_x=2; p_y=3; p_z=4; #5 'dpl 'debug #5
p_x=5; p_y=3; p_z=1; #5 'dpl 'debug #5
  $display();
p_x=2; p_y=5; p_z=3; #5 'dpl 'debug #5
p_x=2; p_y=3; p_z=5; #5 'dpl 'debug #5
p_x=1; p_y=2; p_z=0; #5 'dpl 'debug #5
p_x=2; p_y=1; p_z=0; #5 'dpl 'debug #5
  $display();
  p_x=0; p_y=1; p_z=2; #5 'dpl 'debug #5
p_x=2; p_y=3; p_z=1; #5 'dpl 'debug #5
p_x=1; p_y=2; p_z=3; #5 'dpl 'debug #5
p_x=3; p_y=1; p_z=0; #5 'dpl 'debug #5
   $display();
  p_x=0; p_y=6; p_z=3; #5 'dpl 'debug #5
```

```
        p_x=2; p_y=3; p_z=6; #5 `dpl `debug #5
        p_x=6; p_y=2; p_z=0; #5 `dpl `debug #5
        p_x=2; p_y=7; p_z=0; #5 `dpl `debug #5
          $display();
          p_x=2; p_y=5; p_z=7; #5 `dpl `debug #5
        p_x=7; p_y=6; p_z=5; #5 `dpl `debug #5
        p_x=1; p_y=8; p_z=8; #5 `dpl `debug #5
        p_x=8; p_y=1; p_z=8; #5 `dpl `debug #5
          $display();

          p_x=2; p_y=5; p_z=3; #5 `dpl `debug #5
        p_x=2; p_y=3; p_z=5; #5 `dpl `debug #5
        p_x=1; p_y=2; p_z=0; #5 `dpl `debug #5
        p_x=2; p_y=1; p_z=0; #5 `dpl `debug #5
        $display();
          p_x=2; p_y=5; p_z=3; #5 `dpl `debug #5
        p_x=2; p_y=3; p_z=5; #5 `dpl `debug #5
        p_x=1; p_y=2; p_z=0; #5 `dpl `debug #5
        p_x=2; p_y=1; p_z=0; #5 `dpl `debug #5
        $display();
          p_x=1; p_y=0; p_z=1; #5 `dpl `debug #5
        p_x=1; p_y=1; p_z=0; #5 `dpl `debug #5
        p_x=0; p_y=1; p_z=1; #5 `dpl `debug #5
        p_x=4; p_y=4; p_z=4; #5 `dpl `debug #5
          $display();
          p_x=4; p_y=4; p_z=0; #5 `dpl `debug #5
        p_x=2; p_y=4; p_z=4; #5 `dpl `debug #5
        p_x=2; p_y=0; p_z=2; #5 `dpl `debug #5
        p_x=3; p_y=0; p_z=3; #5 `dpl `debug #5
          $display();
          p_x=1; p_y=1; p_z=0; #5 `dpl `debug #5
        p_x=8; p_y=8; p_z=8; #5 `dpl `debug #5
        p_x=2; p_y=4; p_z=1; #5 `dpl `debug #5
        p_x=4; p_y=2; p_z=1; #5 `dpl `debug #5
          $display();
          p_x=1; p_y=1; p_z=1; #5 `dpl `debug #5
        p_x=2; p_y=2; p_z=2; #5 `dpl `debug #5
        p_x=3; p_y=3; p_z=3; #5 `dpl `debug #5
        p_x=5; p_y=5; p_z=5; #5 `dpl `debug #5
          $display();
          p_x=6; p_y=6; p_z=6; #5 `dpl `debug #5
        p_x=7; p_y=7; p_z=7; #5 `dpl `debug #5
        p_x=9; p_y=9; p_z=9; #5 `dpl `debug #5
        p_x=9; p_y=2; p_z=9; #5 `dpl `debug #5
          $display();
          p_x=1; p_y=1; p_z=0; #5 `dpl `debug #5
        p_x=8; p_y=8; p_z=8; #5 `dpl `debug #5
        p_x=2; p_y=4; p_z=1; #5 `dpl `debug #5
        p_x=4; p_y=2; p_z=1; #5 `dpl `debug #5
          $display();

        //$stop;
        $finish;
        end


endmodule
```