UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ BORIN SOARES

Exploração do Paralelismo em Arquiteturas para Processamento de Imagens e Vídeo

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Altamiro Amadeu Susin

Porto Alegre, janeiro de 2007.

CIP-CATALOGAÇÃO NA PUBLICAÇÃO

Soares, André Borin

Exploração do Paralelismo em Arquiteturas para Processamento de Imagens e Vídeo / André Borin Soares. - Porto Alegre : PPGC da UFRGS, 2007.

134 f. : il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2007. Orientador: Altamiro Amadeu Susin.

1.Processamento de Imagem. 2. NoC. 3. Arquiteturas para processamento de imagens. 4. Hardware para processamento de imagens. I. Susin, Altamiro Amadeu. II . Título.

Agradecimentos

Gostaria de deixar registrado neste trabalho, meus sinceros agradecimentos àquelas pessoas que me ajudaram de alguma forma, em especial:

aos meus pais, por todo o apoio até aqui;

à minha esposa, pelo apoio e paciência em inúmeros momentos que deixamos de estar juntos para que um dia este trabalho fosse completado;

ao Prof. Dr. Altamiro Amadeu Susin, pelos constantes ensinamentos, orientação e apoio sem os quais este trabalho não teria sido realizado;

aos professores do PPGC pelos ensinamentos ministrados, em particular aos professores Luigi Carro e Sérgio Bampi, pelo exemplo de competência e capacidade de trabalho;

aos colegas do LaPSI (Laboratório de Processamento de Sinais e Imagens), em particular à Letícia, Negreiros, Thiago, Davi, Viviane, Bonatto e Vinícius;

aos colegas do GME, pela convivência durante o decorrer do doutorado;

à prof. Vitória Kessler, que com seu magnífico talento ajudou-me a enfrentar a árdua tarefa de aprender a língua francesa em um curtíssimo espaço de tempo;

ao Daniel Mesquita e a Maria Augusta Nunes, que através de sua amizade tornaram mais fácil a vida em Montpellier;

ao prof. Lionel Torres, que me recebeu com boa vontade no LIRMM para o estágio sanduíche;

à UFRGS, ao CNPq e à CAPES pelo apoio financeiro.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AGP	Advanced Graphics Port
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-Set Processor
CAD	Computer Aided Design
CAM	Content addressable memory
CC-NUMA	Cache-Coherent NUMA
CISC	Complex Instruction Set Computer
COMA	Cache-Only Memory Architecture
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
Fps	frames per second
FSBMA	Full Search Block Matching Algorithm
Gops	Giga operations per second
HDL	Hardware Description Language
IP	Intellectual Property
ISA	Instruction Set Architecture
ISP	Image Signal Processor
LUT	Lookup table
MAC	Multiply-accumulate
ME	Motion Estimation
MIMD	Multiple Instruction Flows, Multiple Data Flows
MISD	Multiple Instruction Flows, Single Data Flow
MMX	Multi Media eXtension
MPEG	Motion Picture Experts Group

MRBMA	Multi-Resolution Block Matching Algorithm
MSB	Most-Significant Bit
MVP	Multimedia Video Processor
NCC-NUMA	Non-Cache-Coherent NUMA
NoC	Network-on-Chip
NTSC	National Television System Committee
NUMA	Non-Uniform Memory Access
OCP	Open Core Protocol
PAL	Phase Alternating Line
PA	Saída do filtro passa - altas
PB	Saída do filtro passa - baixas
PCI	Peripheral Component Interface
P.E.	Processing Element
RISC	Reduced Instruction Set Computer
SAD	Sum of Absolute Differences
SC-NUMA	Software-Coherent NUMA
SISD	single instruction flow, single data flow
SIMD	Single Instruction Multiple Data
SoC	System-on-Chip
SoCIN	System-on-Chip Interconnection Network
SoCINfp	System-on-Chip Interconnection Network Fully Parameterizable
SPE	Synergistic Processor Element
ULA	Unidade Lógico Aritmética
UMA	Uniform Memory Access
VCI	Virtual Component Interface
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

a_i	i-ésima entrada da imagem 1 para o cálculo da SAD
b_i	i-ésima entrada da imagem 2 para o cálculo da SAD
c	Correlação entre uma imagem e uma sub-imagem
e_1	Entrada do circuito de cálculo de média de 4 pixels
f	Intensidade luminosa, como função das coordenadas na imagem
I	Intensidade da imagem, na escala de tons de cinza
J	Dimensão horizontal de uma sub-imagem, em pixels
K	Dimensão vertical de uma sub-imagem, em pixels
m	Dimensão horizontal de uma região, em blocos de 4x4 pixels
n	Dimensão vertical de uma região, em blocos de 4x4 pixels
P_x	Número de processadores na horizontal que atuam sobre uma imagem
P_y	Número de processadores na vertical que atuam sobre uma imagem
s	Deslocamento de uma sub-imagem na horizontal
s_1	Saída do circuito de cálculo de média de 4 pixels
t	Deslocamento de uma sub-imagem na vertical
T_{ix}	Dimensão horizontal de uma imagem, em pixels
T_{iy}	Dimensão vertical de uma imagem, em pixels
T_{rx}	Dimensão horizontal de uma região, em pixels
T_{ry}	Dimensão vertical de uma região, em pixels
w	Intensidade luminosa em uma sub-imagem, na escala de tons de cinza

RESUMO

O processamento de vídeo e imagens é uma área de pesquisa de grande importância atualmente devido ao incremento de utilização de imagens nas mais variadas áreas de atividades: entretenimento, vigilância, supervisão e controle, medicina, e outras. Os algoritmos utilizados para reconhecimento, compressão, descompressão, filtragem, restauração e melhoramento de imagens apresentam freqüentemente uma demanda computacional superior àquela que os processadores convencionais podem oferecer, exigindo muitas vezes o desenvolvimento de arquiteturas dedicadas. Este documento descreve o trabalho realizado na exploração do espaço de projeto de arquiteturas para processamento de imagem e de vídeo, utilizando processamento paralelo. Várias características particulares deste tipo de arquitetura são apontadas. Uma nova técnica é apresentada, na qual Processadores Elementares (P.E.s) especializados trabalham de forma cooperativa sobre uma estrutura de comunicação em rede intra-chip

Palavras-Chave: Processamento de imagem, NOC, arquiteturas para processamento de imagem, hardware para processamento de imagem.

Parallelism Exploration in Architectures for Video and Image Processing

ABSTRACT

Nowadays video and image processing is a very important research area, because of its widespread use in a broad class of applications like entertainment, surveillance, control, medicine and many others. Some of the used algorithms to perform recognition, compression, decompression, filtering, restoration and enhancement of the images, require a computational power higher than the one available in conventional processors, requiring the development of dedicated architectures.

This document presents the work developed in the design space exploration in the field of video and image processing architectures by the use of parallel processing. Many characteristics of this kind of architecture are pointed out. A novel technique is presented in which customized Processing Elements work in a cooperative way over a communication structure using a network on chip.

Keywords: Image processing, NOC, image processing architectures, image processing hardware

1 INTRODUÇÃO

O processamento de vídeo e imagens é alvo de muita pesquisa atualmente, pois já é empregado nas mais diversas áreas, da indústria ao entretenimento, e novos algoritmos continuam a exigir cada vez mais poder de processamento. O desenvolvimento das tecnologias de aquisição de imagem, processamento, armazenamento e comunicação de informações possibilitou a disseminação do uso de imagens e vídeo digitais. O processamento é obtido freqüentemente através de arquiteturas dedicadas, que atendem a necessidades muito específicas. Contudo, o campo das arquiteturas onde se tem uma maior flexibilidade de uso e desempenho é uma área mais interessante, pois o reuso destas arquiteturas reduz significativamente o tempo de projeto, que é fator decisivo para empresas que enfrentam um mercado extremamente competitivo.

Até o momento, o desenvolvimento das arquiteturas mostra que os problemas com grande demanda computacional não podem ser resolvidos sem a utilização massiva de processamento paralelo para atender à necessidade computacional dos algoritmos. A opção de aumento do desempenho pela elevação da freqüência do relógio do processador atualmente limita-se em torno de 4 GHz. Neste caso o limite na operação do silício torna mais prático dobrar o desempenho através da utilização do processamento paralelo do que através da elevação da frequência de operação.

Alguns trabalhos vêm buscando explorar o paralelismo de forma massiva em problemas de processamento de imagem, predominantemente na direção dos chips de visão (*vision chips*) (Lindgren et al., 2005; Komuro et al., 2004). Em (Henessy, 1998) é dito que um sistema com processamento massiçamente paralelo é um sistema que emprega mais de 100 processadores. Em (Kawai et al., 2002) cita-se que a informação de imagem vem do mundo real de forma paralela e que é transformada em serial a partir do sensor, criando um gargalo e que restringe o processamento. O processamento massivamente paralelo ligado ao sensor vem para aliviar este gargalo. Dentre os chips de visão, uma parte destas pesquisas utiliza elementos de processamento analógicos (Dudek, 2005) e outra parte, elementos digitais (Komuro et al., 2004).

Simultaneamente, outro ponto de grande importância é a exploração do espaço de projeto no desenvolvimento de arquiteturas para aplicações de imagens e vídeo, onde são identificados os principais fatores que se apresentam como pontos críticos e possivelmente limitadores do desempenho. O processamento de imagens e de vídeo são tarefas que empregam algoritmos que possuem uma dependência de dados relativamente baixa. Assim, as possibilidades de aceleração destes algoritmos são grandes, pois as dimensões das imagens utilizadas permite o particionamento do problema em um grande número de unidades, geralmente grupos de pixels, e que no caso limite podem

ser compostas por apenas um pixel. Contudo, em sistemas reais, são diversos os fatores que limitam a implementação do paralelismo, pois é muito difícil ou impossível na maioria dos casos realizar o sistema considerando área, desempenho, potência e custo simultaneamente. Podem-se citar como principais limitações dos sistemas reais: a área utilizada, a capacidade das memórias, a banda de comunicação com memórias e entre elementos de processamento e freqüência de operação.

Neste trabalho foram verificados alguns fatores que limitam o desempenho de um sistema de processamento de imagens e vídeo através de estudos de caso considerando diferentes aplicações e plataformas e a busca de formas de estender estes limites, seja através de modificações no algoritmo ou de modificações em arquiteturas já existentes ou mesmo no desenvolvimento de novas arquiteturas. Algumas avaliações foram realizadas em mais alto nível e outras através de simulação ou implementação em placa de prototipação, com o objetivo de avaliar o desempenho das diferentes arquiteturas em alto e baixo nível, respectivamente.

1.1 Motivação

A disseminação de aplicações que utilizam imagens e vídeo está exigindo capacidades cada vez maiores dos processadores, dos canais de comunicação e dos meios de armazenamento. Algoritmos eficientes de compressão com perdas, por exemplo, conseguem reduzir em até duas ordens de grandeza o volume de dados associado às imagens, explorando redundâncias espaciais e temporais nos dados. Isto é feito a um custo computacional elevado, principalmente no caso de aplicações de tempo real e de alta definição.

Este trabalho explora o paralelismo massivo para a implementação de algoritmos de alto desempenho em plataformas de hardware dedicadas e ao mesmo tempo flexíveis. São identificados alguns dos algoritmos que são utilizados comumente em problemas de processamento de imagem que tenham grande demanda computacional e seus requisitos de desempenho. Foram selecionados algoritmos que permitam a exploração de diferentes aspectos do paralelismo de uma arquitetura: paralelismo de dados, com o processamento de grupos de dados pelas mesmas instruções e por instruções diferentes (embora constituam o mesmo programa) e paralelismo entre tarefas, através da execução de funções consecutivas de processamento sobre o mesmo conjunto de dados.

Com base nesta identificação, é proposta uma plataforma composta por elementos de processamento e uma infra-estrutura de comunicação para a execução destes algoritmos através da exploração do paralelismo. Diferentes graus de exploração do paralelismo são avaliados para os algoritmos identificados, através da combinação da exploração do paralelismo entre os diferentes elementos de processamento (P.E.s-*Processing Elements*) e dentro de um determinado P.E. através da relação entre a área (custo) do sistema e a aceleração obtida.

Um dos eixos da exploração do espaço de projeto é a reengenharia dos algoritmos visando uma melhor adaptação aos recursos de hardware. Contudo, neste trabalho foram realizados experimentos explorando apenas o aspecto do particionamento de algoritmos e posicionamento das tarefas em diferentes elementos de processamento com o objetivo de aproveitar melhor os recursos de comunicação.

Finalmente, a plataforma proposta é desenvolvida sob a forma de um núcleo IP parametrizável, para ser adaptado conforme as necessidades da aplicação.

1.2 Objetivos

Os objetivos deste trabalho são:

- Identificar algoritmos de processamento de imagens que possibilitem explorar diferentes formas de paralelismo na plataforma desenvolvida, e seus requisitos temporais;
- Definir uma plataforma composta por elementos de processamento e uma infra-estrutura de comunicação para a execução de algoritmos de processamento de imagens através da exploração do paralelismo;
- Definir uma arquitetura parametrizável para a execução dos algoritmos;
- Avaliar o desempenho do sistema para os algoritmos particionados;
- Avaliar o ganho de desempenho que pode ser proporcionado através da exploração do paralelismo na infra-estrutura de comunicação escolhida.

1.3 Organização do texto

Este texto está organizado da seguinte forma: o capítulo 2 contém uma revisão bibliográfica das arquiteturas de maior relevância que são utilizadas atualmente para o processamento de imagens e de vídeo; o capítulo 3 traz uma descrição dos algorimos de processamento de imagens e vídeo que serão utilizados para a exploração do paralelismo neste trabalho; no capítulo 4 uma arquitetura que permite a execução dos algoritmos de processamento de imagens e vídeo, utilizando processamento paralelo é proposta; no capítulo 5 são descritos os experimentos realizados e os resultados obtidos; no capítulo 6 finalmente são apresentadas as conclusões sobre a exploração de diferentes formas de paralelismo na arquitetura.

2 PRINCIPAIS ARQUITETURAS UTILIZADAS PARA APLICAÇÕES DE PROCESSAMENTO DE VÍDEO E IMAGENS

2.1 Classificação das arquiteturas paralelas

As aplicações de processamento de imagens, quando considerada a sua demanda computacional, situam-se dentro de um amplo espectro, com uma predominância das aplicações com grande demanda computacional. Por isso, antes da sua apresentação, é necessário buscar uma classificação de forma a fornecer uma idéia do seu funcionamento.

A classificação mais utilizada para máquinas paralelas é a proposta por Flynn (Flynn, 1966), a qual é baseada na maneira como os fluxos de dados e de instruções são organizados. Nesta classificação, as arquiteturas podem ser agrupadas basicamente nas seguintes classes:

- SISD (*single instruction flow, single data flow*)
- SIMD (*single instruction flow, multiple data flows*)
- MISD (*multiple instruction flows, single data flow*)
- MIMD (*multiple instruction flows, multiple data flows*)

Uma segunda maneira de classificar as máquinas paralelas é quanto ao compartilhamento da memória. Em geral uma classificação adotada para sistemas multi-processados é:

- UMA (*Uniform Memory Access*): O tempo de acesso à memória de cada processador é o mesmo.
- NUMA (*Non-Uniform Memory Access*): a memória é distribuída, implementada como múltiplos nodos que são associados a cada elemento de processamento. O espaço de endereçamento é único. Dependendo da forma como a coerência de cache tenha sido tratada, esta arquitetura pode ser subdividida em CC-NUMA (*cache-coherent* NUMA), NCC-NUMA (*non-cache-coherent* NUMA), SC-NUMA (*software-coherent* NUMA).
- COMA (*Cache-Only Memory Architecture*): todas as memórias estão estruturadas na forma de cache (este cache tem mais capacidade que um cache tradicional). Há suporte de hardware para replicação de um bloco de cache em múltiplos nós,

enquanto que nas anteriores os blocos são normalmente invalidados e não são atualizados (Navaux, 2003).

Esta classificação pode ser vista na Figura 2.1.

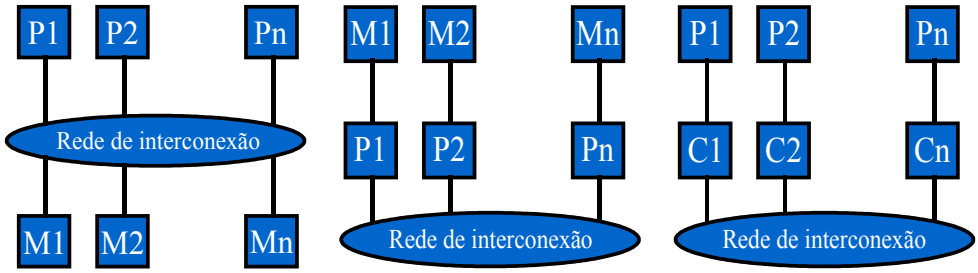


Figura 2.1: Classificação das arquiteturas quanto ao compartilhamento da memória. (a) arquitetura UMA; (b) arquitetura NUMA; (c) arquitetura COMA.

Uma classificação mais específica para arquiteturas de processamento de imagens foi elaborada por (Dasu et al., 2002) e encontra-se reproduzida na Figura 2.2. Esta classificação é o resultado da combinação de duas classificações, uma que é baseada na evolução da arquitetura e outra que é baseada funcionalidade.

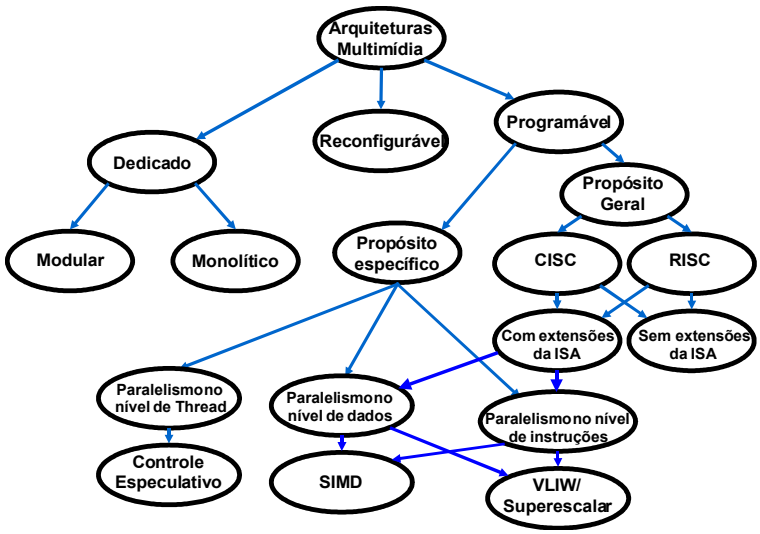


Figura 2.2: Classificação das arquiteturas multimídia (Dasu et al., 2002).

Uma descrição das arquiteturas mais relevantes para processamento de imagens e vídeo é apresentada abaixo.

2.2 Arquiteturas dedicadas

Os ASICs (*Application Specific Integrated Circuits*) são os que apresentam o melhor desempenho dentre todas as arquiteturas para processamento de imagem. São circuitos

projetados com uma finalidade única e que têm em geral a melhor relação custo/benefício para um dado problema. Contudo, apresentam muito pouca ou nenhuma flexibilidade para adaptação para outros problemas, exigindo muitas vezes o reprojeto para uso em outras aplicações. Exemplos de arquiteturas dedicadas são relatados em (Fujiyoshi, 2005; Enomoto, 2003; Nakamura, 2004).

2.3 Arquiteturas programáveis

As arquiteturas que podem ser reutilizadas para diferentes aplicações, através de programação, são enquadradas nesta classe.

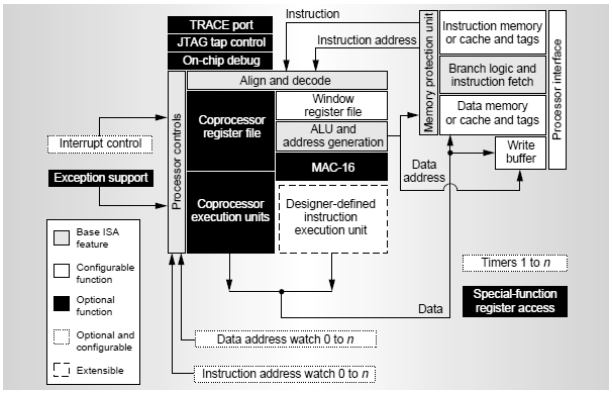
2.3.1 Processadores de propósito geral

Exemplos bem conhecidos desta classe de arquitetura são os processadores da família Pentium™, processadores MIPS, ARM™ e Spark™. Em geral, este tipo de processador apresenta um desempenho bom para operações simples sobre imagens estáticas (desde que programado adequadamente), onde o usuário pode esperar poucos segundos para o término de uma operação. Operações relativamente simples sobre imagens podem ser realizadas na taxa do sinal de vídeo (30 quadros por segundo), como por exemplo filtragem de imagens com uma vizinhança muito pequena, binarização, descompressão de vídeo em resoluções não muito elevadas e algumas operações sobre histograma. Entretanto, operações complexas, como por exemplo compressão de vídeo empregando o padrão MPEG-4 (*Motion Picture Experts Group*), ainda não podem ser realizadas em tempo real neste tipo de arquitetura. Seu desempenho é melhorado através de extensões do conjunto de instruções (ISA - *Instruction Set Architecture*), como por exemplo o uso de instruções MMX, SIMD e SSE. Contudo, como o paralelismo oferecido por estas instruções é relativamente baixo, pois os dados devem ser empacotados antes de ser processados e o número de unidades funcionais que atuam em paralelo é pequeno, resultando numa aceleração relativamente moderada.

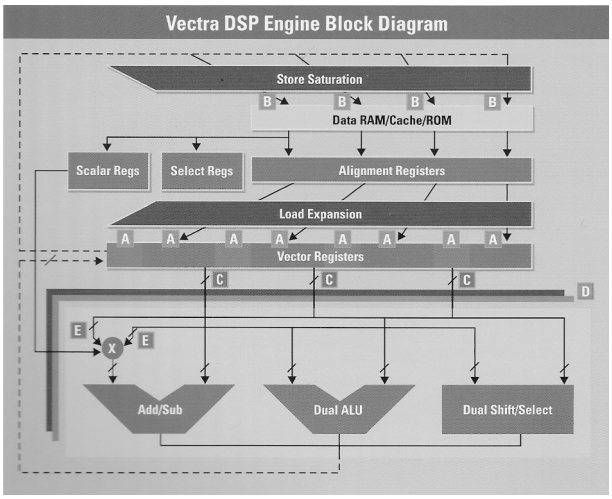
2.3.2 ASIP(Application Specific Instruction-Set Processor)

Esta arquitetura consiste em um processador no qual pode-se realizar a customização de instruções. Um exemplo deste tipo de arquitetura é o processador Xtensa™ (Gonzales, 2000; Ezer, 2000), que é um núcleo de um processador parametrizável. Na Figura 2.3(a) pode-se observar uma visão geral da arquitetura deste processador, e em (b) detalhes do bloco dedicado a DSP.

Neste processador é possível configurar blocos, tais como o banco de registradores e a unidade de gerenciamento de memória, escolher a utilização ou não de outros blocos, tais como MAC (multiplica e acumula) e FPU(unidade de ponto flutuante) e multiplicadores, ou escolher e configurar a unidade de DSP. Contudo, somente a otimização de instruções não permite que se obtenha um ganho de desempenho muito grande, pois em geral este ganho está atrelado à arquitetura do processador. O processador configurado é disponibilizado sob a forma de uma descrição de *hardware* em nível de transferência de registradores (RTL - *Register Transfer Level*), e se pode construir um SoC com mais de um núcleo operando em paralelo em um mesmo chip.



(a)



(b)

Figura 2.3: (a) Visão geral da arquitetura do processador Xtensa (Gonzales, 2000); (b) detalhe da unidade dedicada a DSP (Xtensa product Brief).

2.3.3 Processadores de Sinais Digitais (DSPs) / *Media processors*

O DSP é um tipo de processador específico para processamento de sinais, pois possui operadores especializados específicos para otimizar este tipo de operação, tais como MAC (multiplica e acumula em um único ciclo), *barrel shifter* (unidade de deslocamento em um único ciclo), e controle de laço em *hardware* (o controle de laço não consome ciclos de execução). Existem também versões com múltiplas unidades funcionais, controladas por uma palavra de instrução muito longa (VLIW). Muitos tem arquitetura Harvard, permitindo busca de operandos e instruções simultaneamente. Entretanto, este tipo de processador ainda apresenta limitações no desempenho devido ao pequeno número de unidades funcionais. Assim, operações tais como filtragem em imagens e decodificação de vídeo podem ser realizadas em resolução padrão (720x480, por exemplo) na taxa de 30 quadros por segundo. Já tarefas com maior nível de complexidade, como por exemplo codificação de vídeo digital em alta definição, tem seu número de quadros por segundo reduzido. Na Figura 2.4 pode-se observar uma representação esquemática de um processador DSP VLIW. Este processador conta com

três portas de vídeo configuráveis (portas VP0, VP1 e VP2), que podem servir para captura ou exibição de vídeo. Cada porta consiste em dois canais e um *buffer* de recepção, que pode ser separado entre os dois canais. As portas McASP são utilizadas para comunicação serial (transmissão e recepção). Cada uma possui 8 pinos que podem ser alocados individualmente para a transmissão de áudio. Já a porta EMIF64 permite o acesso à memória externa, síncrona ou assíncrona, sem a utilização de lógica de cola. Há também uma porta de saída analógica interpolada (porta VIC) e portas de acesso à rede Ethernet 10/100 (porta EMAC) e para barramento PCI.

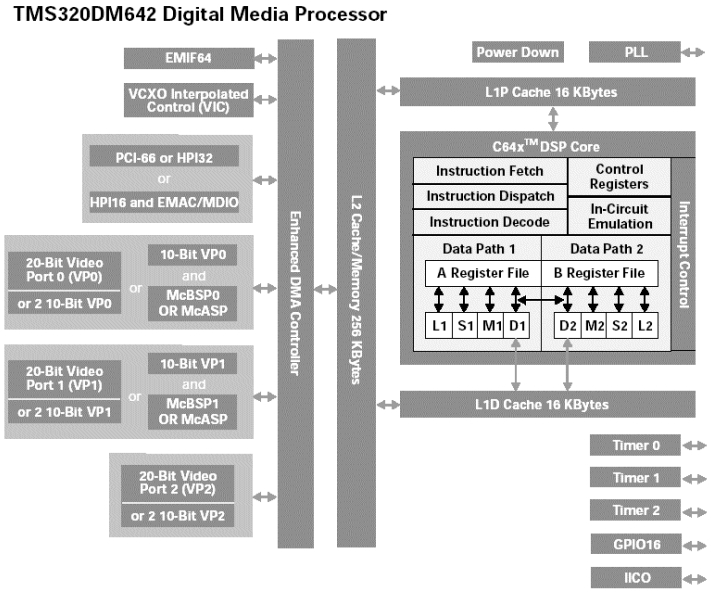


Figura 2.4: Representação esquemática de um processador DSP TMS320DM642 (http://www.ti.com)

Outro exemplo pode ser encontrado em (Yi-Shin et al., 2005), onde é utilizado um DSP Trimedia TM 1300 Media Processor para decodificação de vídeo em múltiplos formatos considerando uma resolução padrão de 720x480. Na Figura 2.5 pode-se observar uma representação esquemática da arquitetura deste Media Processor.

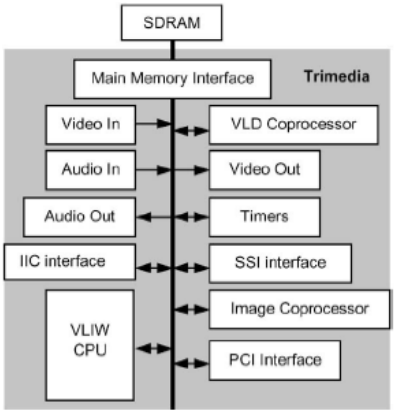
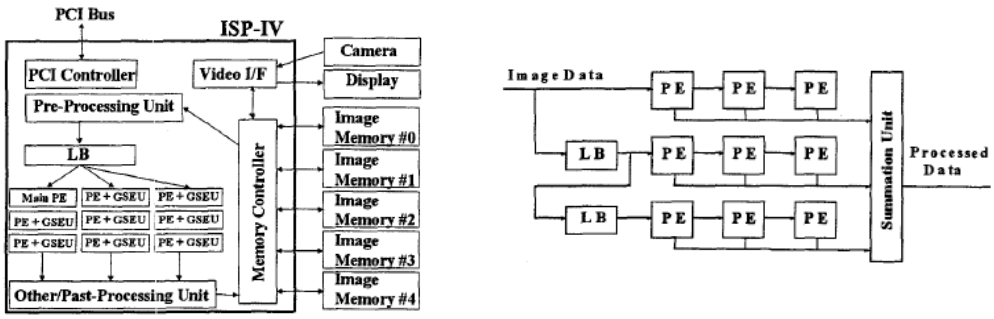


Figura 2.5: Representação esquemática da arquitetura Trimedia (Chalapali et al., 2004).

Esta arquitetura conta com entrada de vídeo digital no formato YUV 4:2:2, saída digital no formato YUV 4:2:2. O bloco co-processador de imagem é capaz de realizar a filtragem de uma imagem presente na SDRAM, reescrevendo a imagem na memória, realizar conversão de cores do formato RGB para YUV, e também realizar sobre-amostragem e sub-amostragem da imagem. Este bloco também permite operações do tipo *chroma keying*, onde uma cor específica da imagem é substituída por uma imagem armazenada. Já o bloco co-processador VLD (*Variable Lenght Decoder*) é capaz de realizar descompressão Huffman para vídeo MPEG1 e MPEG2. A interface SSI (*Synchronous Serial Interface*) é uma interface serial para um modem analógico fora do chip, um terminal de rede ou um conversor AD/DA.

2.3.4 *Image Signal Processors* (ISP)

Em (Muramatsu et al., 1998) uma arquitetura denominada de *Image Signal Processor* é apresentada. Um diagrama básico da arquitetura pode ser visto na Figura 2.6(a). A arquitetura conta com interface de vídeo, controlador PCI, controlador de memória e processador de imagem no mesmo chip. As unidades principais do processamento são uma de pré - processamento, uma de pós - processamento e 9 unidades que são elementos de processamento e de busca em tons de cinza. Finalmente, há *buffers* de linha, que são utilizados para operações de morfologia e convolução, fornecendo vizinhanças de 3x3 e 9x1, realizadas pelos 9 elementos de processamento. Um dos 9 P.E.s tem múltiplas funções e realiza outras funções adicionais entre os pixels. Os outros P.E.s contém ULAs.



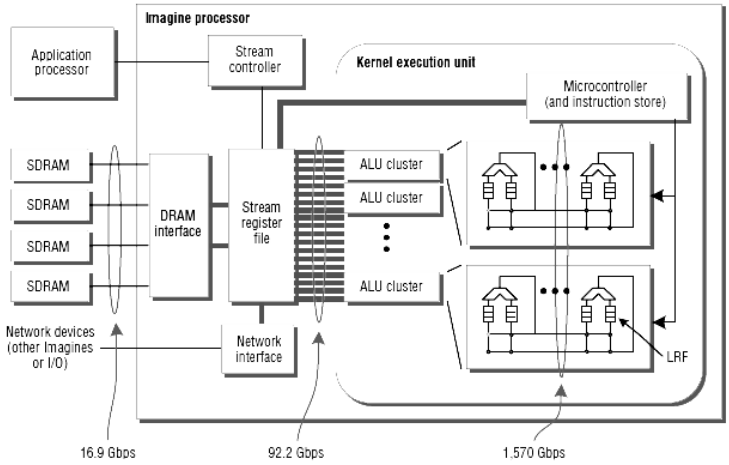
(a) (b)

Figura 2.6: (a) Arquitetura básica de um ISP; (b) detalhe de um bloco dedicado para processamento de vizinhança (Muramatsu et al., 1998).

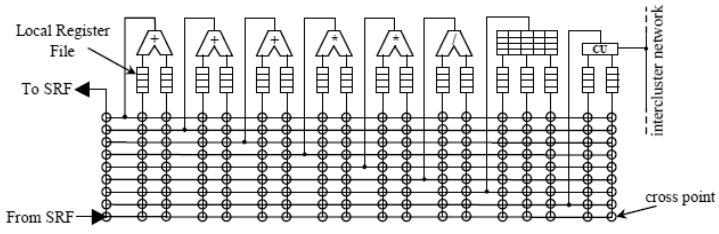
A unidade de pós - processamento realiza funções como valor absoluto, binarização e histograma. Já a unidade de pré - processamento realiza operações de mascaramento e inversão de bits. O controlador de memória permite acesso independente aos blocos de memória. Na Figura 2.6(b) pode-se observar a configuração no caso do processamento de um pixel e sua vizinhança para uma operação de convolução. Outras funções de processamento de imagem básico realizadas são laplaciano, filtragem mediana e suavização (*smoothing)* da imagem, através do *datapath* ligado ao buffer de vizinhança. É possível a realização de acompanhamento das coordenadas de objetos (*tracking*) em alta velocidade devido à presença de 9 unidades de busca em escala de cinza. Extração de cor de imagens YUV em tempo real é realizada por funções de transformação de cor.

2.3.5 Stream Processors

Stream Processors são DSPs projetados para aplicações embarcadas de alta performance (Rajagopal, 2004). Eles contêm *clusters* de unidades funcionais e provêm hierarquia de largura de banda, suportando centenas de unidades aritméticas. O paralelismo no nível de instrução é explorado dentro dos *clusters* e paralelismo no nível de dados entre os *clusters*. O *stream processor* Imagine, apresentado em (Rixner et al., 1998) tem um desempenho de 16 Gflops para operações de precisão simples em 400MHz, com um custo em hardware de 21 milhões de transistores. Na Figura 2.7(a) pode-se observar uma representação de um sistema que utiliza esta arquitetura. O código da aplicação opera em conjunto com o *Stream Register File* (SRF), que possui 64 kb. Operações de *load* e *store* armazenam e recuperam *streams* de dados entre a memória principal e o SRF. Os *clusters* aritméticos contém diversas ULAs e operam sob um controle VLIW. Uma visão mais detalhada do cluster pode ser encontrada na Figura 2.7(b).



(a)



(b)

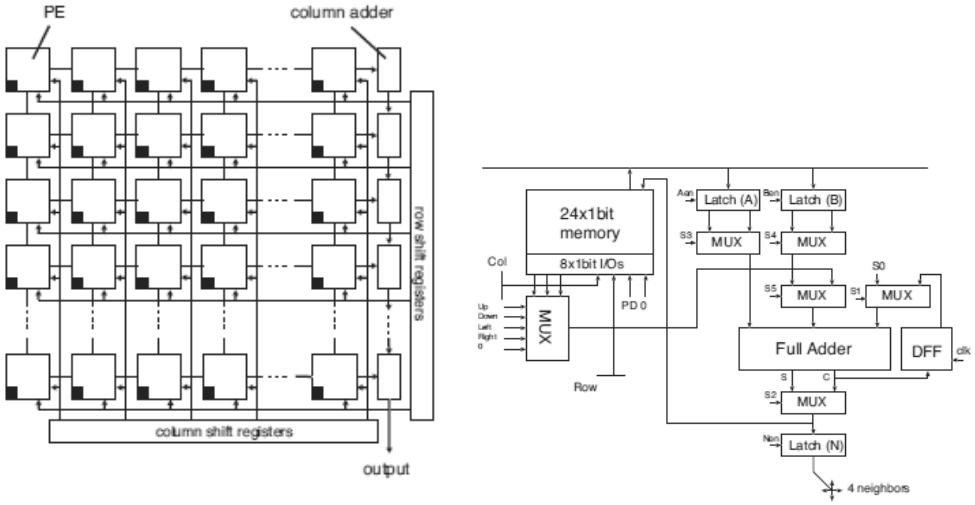
Figura 2.7: (a)Representação esquemática de um *stream processor* (Ujval et al., 2003); (b) detalhe de um *cluster* aritmético (Rixner et al., 1998).

Resultados intermediários da computação são armazenados localmente em registradores e não utilizam o canal de comunicação entre as unidades funcionais e o SRF (o qual possui uma largura de banda menor). Cada *stream* de dados pode ser operado por mais de um núcleo de computação. A comunicação entre diferentes processadores pode ser realizada através de uma interface de rede (fora do chip, customizada), com 4 canais

bidirecionais de 400MB/s para permitir a montagem de diferentes topologias de rede, utilizando funções *send* e *receive* (Rixner et al., 1998).

2.3.6 Chips de visão (Vision Chips)

Em (Akita, 2003) define-se chip de visão como sendo a integração dos circuitos de processamento de sinal com o sensor de aquisição de imagens e que pode realizar processamento de informação paralelamente. Já em (Kagami et al., 2002) define-se que um chip de visão digital é aquele em que um foto – detector, um conversor A/D e um elemento digital de processamento estão conectados em cada pixel. Isto permite realizar algumas operações muito mais rápido do que a taxa de vídeo (NTSC 30 Hz, PAL 25 Hz), chegando em alguns casos a 1000 fps. Existem chips de visão analógicos e digitais. Neste trabalho serão considerados somente os chips de visão digitais, pois em geral é extremamente difícil desenvolver chips de visão analógicos de grande capacidade que possam processar imagens em alta resolução devido a problemas como repetibilidade e estabilidade na fabricação e flexibilidade de programação. Este tipo de arquitetura ainda apresenta limitações tecnológicas e assim o tamanho e o número de processadores acabam ficando limitados. Um exemplo desta arquitetura pode ser visto na Figura 2.8. Em geral, encontram-se soluções com resoluções até cerca de 128x128 pixels com desempenho de 330 Gops/s (analógicos), sendo que os digitais ainda encontram-se com resoluções de 64x64 pixels e têm desempenho na faixa de 50 Gops/s. Outra denominação utilizada para este tipo de processador é Matriz de Processadores do Plano Focal (*Focal Plane Array Processors*).



(a) (b)

Figura 2.8: (a) Estrutura de um chip de visão; (b) Detalhe dos elementos de processamento (Komuro et al., 2004).

Em (Kagawa, 2004) foi desenvolvida uma arquitetura para processamento paralelo no nível de pixel empregando conversores AD de 1 bit digitais ligados aos fotodetectores e elementos de processamento bastante simples. Neste trabalho os sinais têm uma representação por freqüência de pulsos. As operações ocorrem por supressão ou

promoção de pulsos baseadas em interações entre os pulsos do elemento de interesse e dos vizinhos, onde a supressão de pulsos corresponde a um peso negativo e a promoção de pulsos a um peso positivo. Com isto os operadores baseados em correlação, tais como borramento, detecção e melhoria de bordas são implementados. Na Figura 2.9 pode-se observar uma representação esquemática deste processo. Um dos problemas com os chips de visão é que o processamento deve ser simples porque a área é restrita. Um segundo é que a saída de um chip de visão ainda é uma matriz de pixels, enquanto que o que necessitamos é o seu significado (Akita et al., 2003). Em (Akita et al., 2003) é apresentado um sensor de 23x23 pixels que extrai as posições dos objetos no plano focal. Uma pequena parte do processamento é analógico, pois é realizada a comparação do valor de cada pixel com uma tensão de referência, que tem a forma de uma rampa decrescente no tempo. Os autores tomam como centróide o pico da intensidade luminosa de um objeto em uma rede resistiva. Um circuito digital realiza uma comparação de cada elemento com a sua vizinhança em paralelo. Quando o valor do primeiro pixel de um objeto é maior do que a tensão de referência, este valor é tomado como centro e um sinal digital de supressão é lançado para os pixels vizinhos para que somente um pixel seja tomado como centro. O resultado é uma matriz com a posição dos centros dos objetos, e um segundo circuito digital realiza uma varredura nesta matriz e envia somente as coordenadas para fora do sensor.

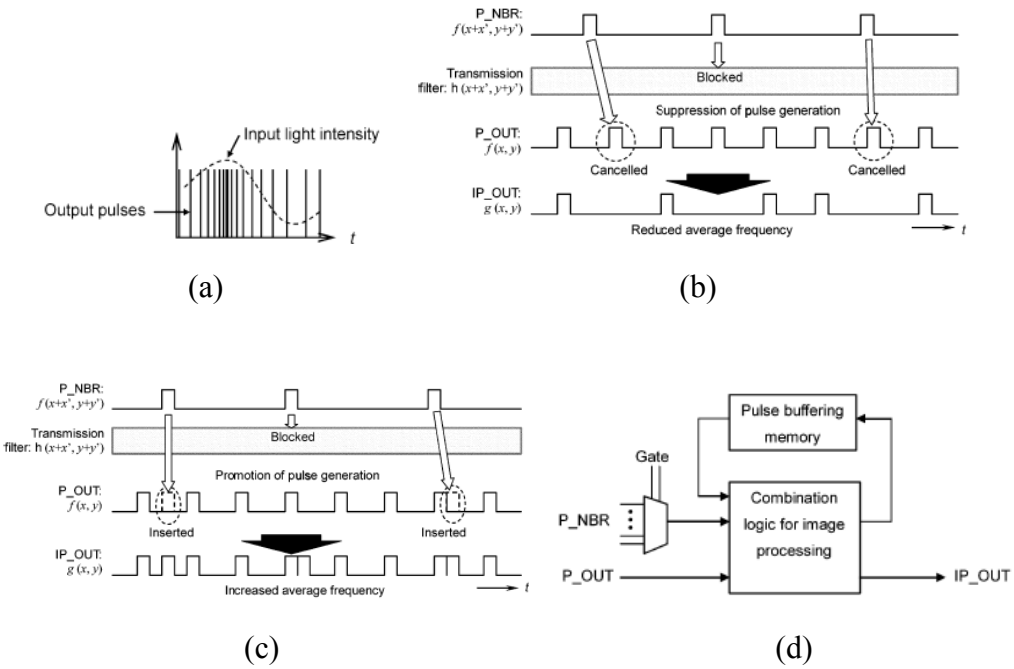


Figura 2.9: (a) Representação da imagem através de freqüência de pulsos;
(b) Multiplicação por um coeficiente negativo através de supressão de pulsos;
(c) Multiplicação por um coeficiente positivo através de promoção de pulsos;
(d) representação esquemática de um elemento de processamento com entrada dos sinais dos elementos vizinhos e sinal de saída (Kagawa, 2004).

Na Figura 2.10 pode-se observar detalhes da arquitetura. Em (a) a intensidade luminosa sobre os foto - detectores gera uma tensão proporcional ao tempo de exposição sobre

uma rede resistiva. Em (b) esta tensão é comparada com uma tensão de referência, onde na ausência de um sinal de inibição dos elementos vizinhos, o pixel de maior intensidade gera uma saída igual a 1 ao ultrapassar a tensão de referência, e produz um sinal de inibição para os elementos vizinhos. Em (c) pode-se observar a leitura dos centróides e o problema de ambigüidade que surge quando há mais de um centróide na matriz, utilizando-se decodificadores para a leitura da posição linha/coluna. Nesta arquitetura optou-se por fazer um mascaramento da saída através de uma *flag* em cada pixel. A máscara vai sendo removida durante a varredura da matriz, e após a leitura de um centróide é recolocada sobre este, produzindo assim somente um centróide por vez.

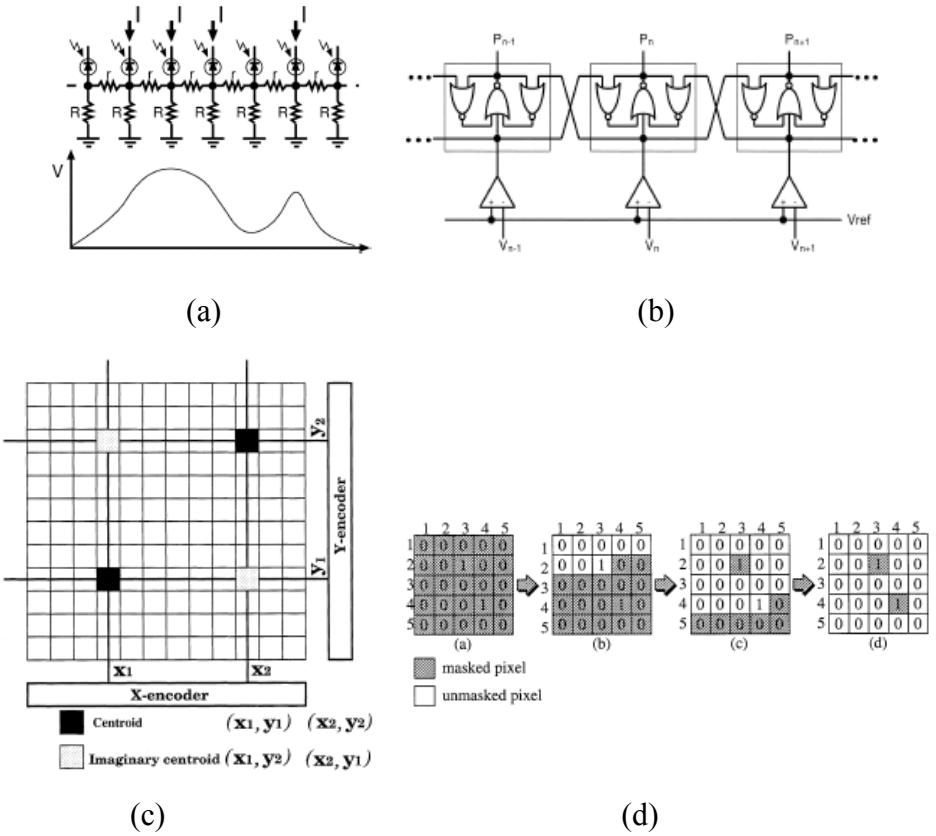
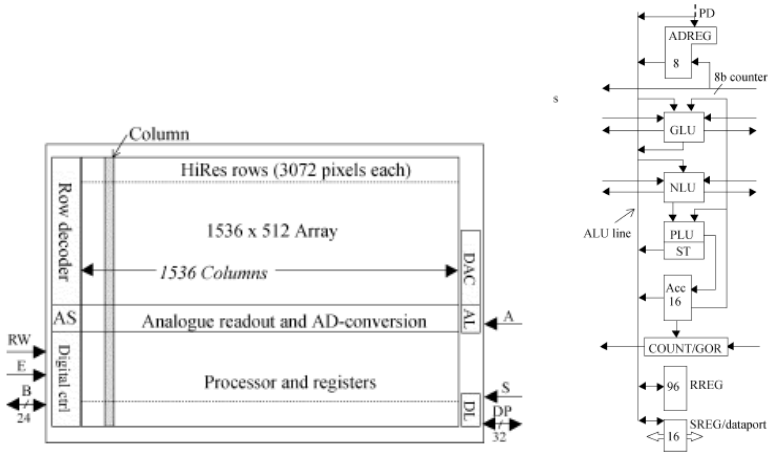


Figura 2.10: (a) Intensidade luminosa sobre uma linha do sensor, modelado como um conjunto de fotodetectores ligados a uma rede resistiva; (b) circuito de comparação do valor dos pixels com uma rampa de referência; (c) pontos de intensidade luminosa máxima detectados (centróide) e problema de leitura; (d) remoção do problema de leitura através de varredura com mascaramento (Akita et al., 2003).

2.3.7 Sensor de visão inteligente (Smart Vision Sensor)

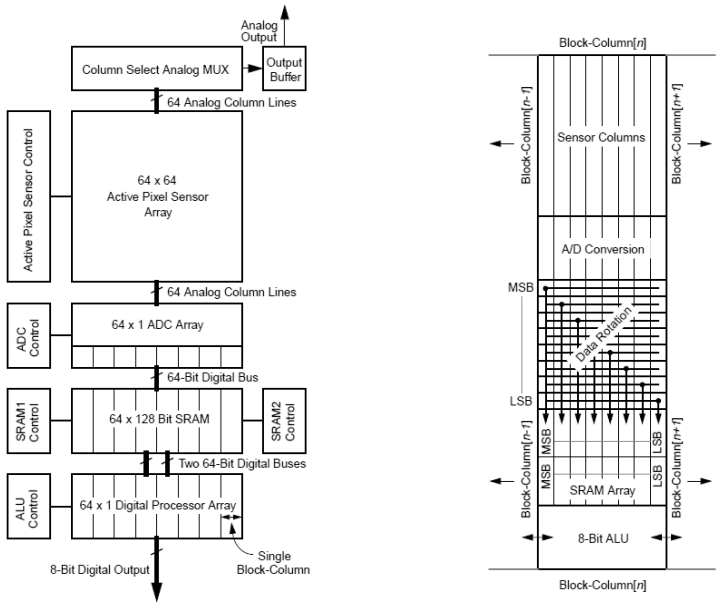
Esta arquitetura consiste em um sensor e múltiplos processadores em um único chip. Em exemplo desta arquitetura pode ser visto na Figura 2.11 (Lindgren et al., 2005), onde o sensor tem uma resolução de 1536x512 pixels e possui 1536 processadores do tipo bit-serial. O aumento na resolução em relação aos chips de visão é possível pois aqui cada processador é responsável por uma coluna da imagem, e não um pixel somente.



(a) (b)

Figura 2.11: (a) Representação esquemática do sensor de visão inteligente; (b) elemento de processamento (Lindgren et al., 2005).

Em (Morris et al., 1999) é apresentada uma arquitetura com o sensor de 64 x 64 pixels ligado a uma coluna de conversores AD/8 bits, uma memória SRAM e uma coluna de processadores digitais. Cada conjunto de 8 colunas é chamado de bloco - colunas *(block-columns*). Este grupo de 8 colunas é tratado por um processador de 8 bits. A arquitetura é organizada em colunas de 1 bit até o registrador que armazena o valor da coluna, momento a partir do qual o valor é rotacionado de 90º para ser processado no P.E., ou armazenado na SRAM (em linha, entre o P.E. e os registradores ligados aos ADs). Na Figura 2.12 pode-se observar a arquitetura do sistema em (a) e do elemento de processamento em (b).

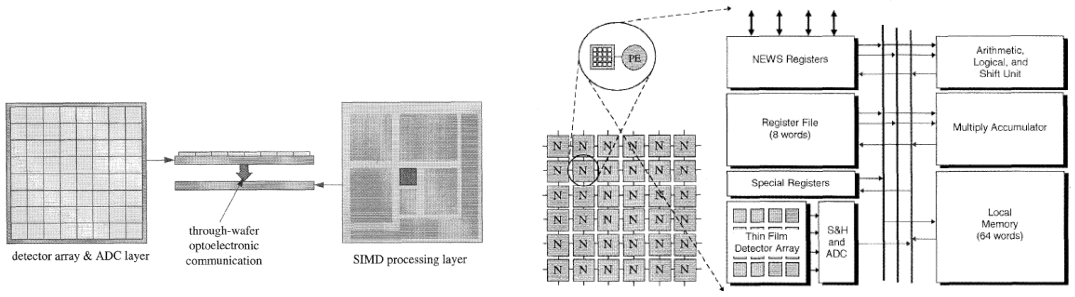


(a) (b)

Figura 2.12: (a) Representação esquemática da arquitetura em coluna; (b) processador do bloco-coluna com matriz de chaveamento para rotação dos dados entre a matriz de ADC e a matriz SRAM (Morris et al., 1999).

2.3.8 Smart pixel architecture/Focal plane image processing system.

Em (Wills et al., 1996; Wills et al., 1996b) aparece o conceito de colocar uma matriz de sensores sobre um circuito de processamento interligados por um feixe paralelo e óptico de dados no lugar de CCD + microprocessador, em duas camadas distintas. Desta forma a matriz de elementos do sensor não é afetada pela área da parte de processamento. Foi desenvolvida uma arquitetura, denominada SIMPil (SIMD *Pixel Processor*). Nesta arquitetura um elemento de processamento ligado está ligado a um grupo de pixels (elementos do sensor) e aos elementos vizinhos, sendo que os autores sugerem o uso de 36 a 64 pixels por nodo para a sua arquitetura. Na Figura 2.13 pode-se observar uma representação esquemática desta arquitetura. Em (a) pode-se observar a matriz de sensores e a camada onde é realizado o processamento. A interconexão ocorre através de dispositivos optoeletrônicos, onde emissores colocados no sensor passam dados para a camada de processamento. Em (b) pode ser vista a matriz de elementos de processamento e mais detalhes de um P.E. O elemento de processamento é composto por conversor AD, registradores de 8 bits, memória de 64 palavras, ULA de 8 bits e MAC de 16 bits.



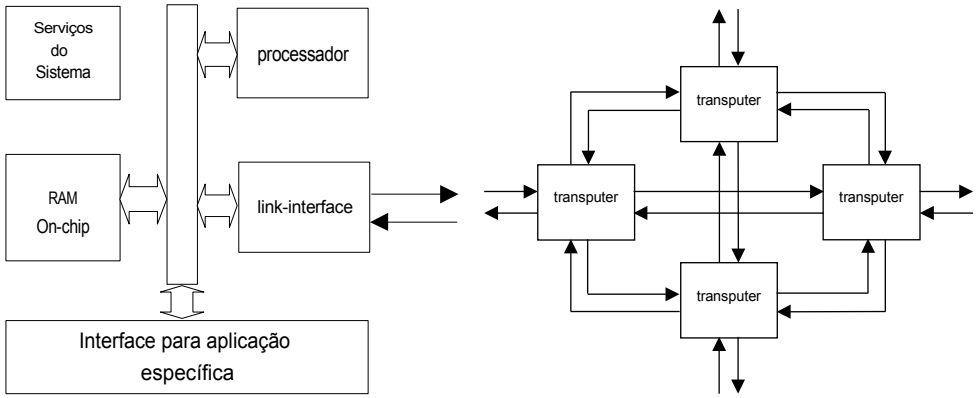
(a) (b)

Figura 2.13: (a) Representação esquemática da arquitetura SIMPil, em duas camadas interconectadas por um feixe óptico de dados; (b) detalhe do elemento de processamento (Wills et al., 1996b).

Em (Chai et al., 1999) são apresentadas aplicações para esta arquitetura (auto-foco, segmentação pelo método *k-means* e compressão) com uma taxa de operações de 500 a 1500 Gops.

2.3.9 Transputers

Este é um circuito integrado com um processador e um módulo de comunicação. Foi inventado nos anos 80, quando a quantidade de transistores que podia ser colocada em um único chip era bastante pequena, para ser uma unidade básica na construção de máquinas paralelas. Uma máquina construída com transputers enquadra-se no modelo MIMD. Na Figura 2.14(a) pode-se ver a arquitetura básica de um *transputer*, e em (b) a interconexão entre 4 *transputers*. Foi utilizado em algumas aplicações tais como reconstrução estéreo (Zatari et al., 1997), detecção de bordas, transformada Hough, transformada rápida de Fourier e convolução (Ehandarkar et al., 1997).



(a) (b)

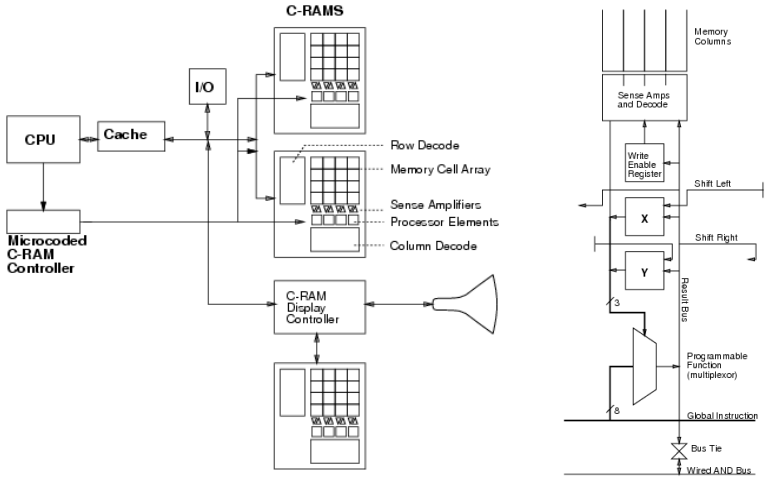
Figura 2.14: (a) Arquitetura de um *transputer*; (b) interconexão entre 4 *transputers* (Pitas, 1993).

2.3.10 Arquiteturas com processamento distribuído com a memória

Estas arquiteturas buscam integrar memória e processamento em uma unidade básica de forma a reduzir o número de pinos e a potência consumida na construção de um sistema maciçamente paralelo.

2.3.10.1 C•RAM (Computational RAM)

Este sistema consiste em uma memória RAM convencional com processadores SIMD adicionados aos *sense amplifiers* (Elliot et al., 1992). Os processadores são do tipo *bit-serial*, programados externamente. Na Figura 2.15 (a) pode-se observar uma representação da arquitetura de todo o sistema empregando C•RAM, e em (b) o elemento de processamento *bit-serial* mais detalhadamente.



(a) (b)

Figura 2.15: (a) Arquitetura de um sistema empregando C•RAM; (b) elemento de processamento (Elliot et al., 1992).

Como um exemplo de aplicação deste sistema é citada a DCT (em blocos 8x8, utilizando o algoritmo de Chan e com 15 bits de precisão) de uma imagem de 1M Pixel, armazenada em 8 C•RAMs, a qual toma 5.7ms. Outro exemplo citado pelos autores é a operação de convolução de uma imagem 1024x1024 com um filtro 3x3 utilizando multiplicação de 8 bits, em 8.6 ms.

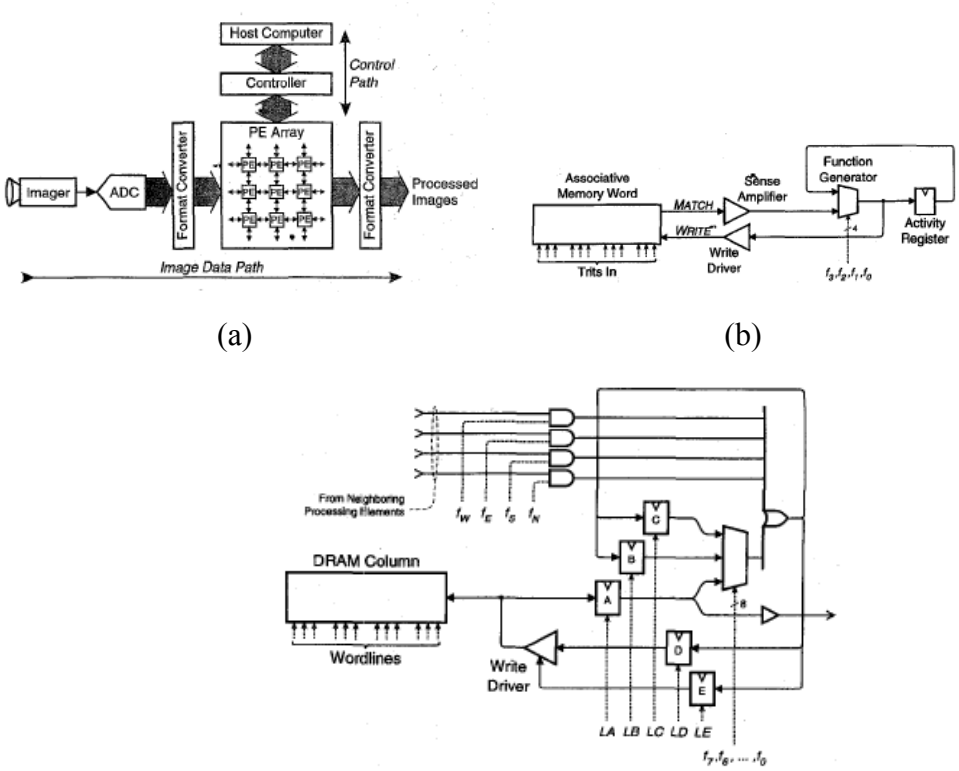
2.3.10.2 CAM (Content addressable memory)

Este tipo de arquitetura utiliza memórias endereçáveis por conteúdo para a realização de processamento massivo de imagens. Esta arquitetura consiste basicamente em uma matriz de elementos de processamento construída em conjunto com uma memória CAM em um mesmo chip. Com os elementos de processamento próximos à memória, a freqüência de operação pode ser elevada, pois não há capacitâncias externas ao chip. Uma vez que este tipo de memória é utilizada (e que ocupa mais recursos), operações de busca são extremamente rápidas.

Em (Ikenaga et al., 1998) é utilizada uma arquitetura com CAM para a realização de autômatos celulares 2D. Como a memória está organizada em uma dimensão neste sistema, a memória é utilizada particionada para acesso em paralelo e em cada parte é mapeada uma parte da matriz bidimensional, em zig-zag. Esta tática reduz o tempo de transferência de dados para os elementos centrais através do paralelismo. Um FPGA é utilizado para realizar o controle da matriz, podendo produzir diferentes seqüências de comandos e mesmo fazer a matriz operar como multicamada.

Em (Ikenaga et al., 1999) são apresentados resultados de performance desta arquitetura, sendo que uma busca de 64 bits é realizada em 25 ns, e um chip é capaz de realizar 640 Gops de busca, em 40MHz. Já operações aritméticas de 8 bits têm uma performance de 9.7 Gops por chip, sendo que cada operação tem duração de 1.7us, correspondente aos 128x128 P.E.s. Operações de detecção de borda empregando morfologia tem uma duração de cerca de 18.6us, o que pressupõe que se tenha 0.9 Gops por chip.

Em (Gealow et al., 1996) é apresentada uma comparação entre uma arquitetura composta por uma matriz de elementos de processamento que operam utilizando uma memória do tipo CAM (processador paralelo associativo) e outra que utiliza elementos de processamento empregando memória DRAM. Na Figura 2.16 pode-se ver mais detalhes da arquitetura. Em uma as operações são baseadas em comparação, enquanto que na outra as operações ocorrem de forma bit-serial utilizando diretamente portas lógicas. A arquitetura não contém os elementos de processamento associados diretamente aos pixels. Há um gargalo na transferência da imagem. Os resultados mostram que em geral operações com memória CAM tem o seu pico de performance para operações baseadas em busca mas esta performance é drásticamente reduzida em operações que envolvem aritmética, pois são necessários vários ciclos para a execução de uma operação. O resultado, apresentado no trabalho, para fluxo óptico (*optical flow*) também mostra que mesmo a solução com DRAM ainda é mais rápida do que a que utiliza CAM. Esta operação foi realizada em uma vizinhança pequena (d=7) e com blocos pequenos (5x5).



(a) (b)

(c)

Figura 2.16: (a) Representação esquemática da arquitetura utilizada em (Gealow et al., 1996); (b) elemento de processamento utilizando CAM; (c) elemento de processamento utilizando DRAM (Gealow et al., 1996)

2.3.11 Matrizes lineares de processadores

Neste tipo de arquitetura os processadores são organizados em linha, realizando o processamento em paralelo. Um exemplo deste tipo de arquitetura é apresentada em (Kurokawa et al., 1996), onde quatro conjuntos 1080 processadores de 1bit são utilizados para operações como conversão de tamanho da imagem e separação de luminância e crominância. Tal arquitetura pode ser observada na Figura 2.17.

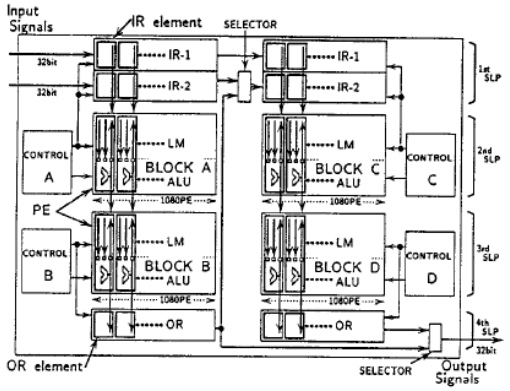


Figura 2.17: Arquitetura que utiliza conjuntos de matrizes lineares de elementos de processamento (Kurokawa et al., 1996).

2.4 Arquiteturas reconfiguráveis

Sistemas de computação reconfiguráveis podem ser definidos como sistemas que combinam hardware reconfigurável com processadores programáveis por software (Singh, 1998). O hardware reconfigurável permite que seja customizável para as necessidades de qualquer problema após a fabricação do dispositivo e que se explore um amplo grau de computação customizada espacialmente (DeHon, 1999). Este tipo de arquitetura pode ser utilizado para processamento de imagens, como pode ser visto em (Boschetti et al., 2004; Singh et al., 1998; Sassatelli et al., 2002). Na Figura 2.18 pode-se observar uma representação esquemática de uma arquitetura reconfigurável, denominada *Systolic Ring*. Nesta arquitetura, os elementos de processamento são dispostos em um laço fechado, formando um anel e existe um laço de realimentação dos dados. Nesta arquitetura, pode-se controlar a interconexão entre os elementos através de chaves.

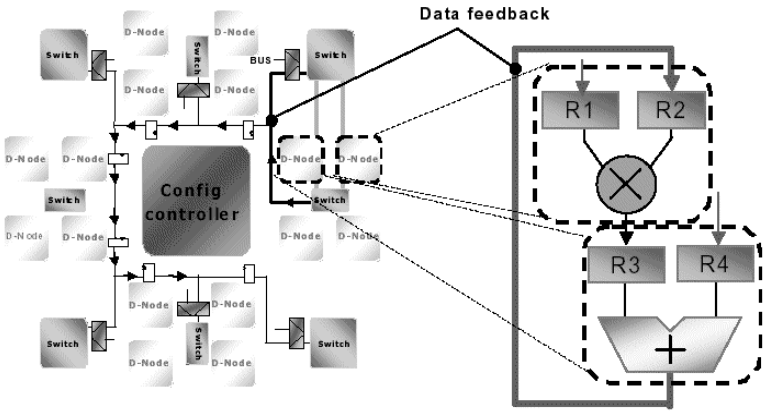


Figura 2.18: Exemplo de arquitetura reconfigurável (Sassatelli, 2002).

As arquiteturas reconfiguráveis podem ser divididas conforme a granularidade, termo que refere-se ao tamanho dos dados para as operações (Singh et al., 1998). Esta granularidade pode ser grossa, média ou fina. Granularidade grossa refere-se a operações sobre dados com o tamanho de uma palavra, empregando ALUs. Já a granularidade fina refere-se a operações em nível de bits, como por exemplo portas lógicas, ou LUTs (*Lookup tables*, para a implementação de funções lógicas). Os FPGAs (*Field Programmable Gate Arrays*) são dispositivos que empregam reconfiguração com granularidade fina, no nível de portas lógicas.

2.4.1 Arquiteturas de Grão Fino

Estas arquiteturas tem o *datapath* geralmente com largura de 1 bit. Contudo, são menos eficientes do que as arquiteturas de grão grosso, devido à maior ocupação da área do dispositivo por elementos para roteamento, o qual também é pobre (Hartenstein, 2001).

2.4.2 Arquiteturas de Grão Grosso

Estas arquiteturas possuem um grau menor de flexibilidade do que as arquiteturas de grão fino, uma vez que já tém seus operadores definidos com uma determinada largura de bits. De acordo com a distribuição espacial dos elementos de processamento, as

arquiteturas de grão grosso geralmente podem ser classificadas nos principais grupos (Hartenstein et al., 2001):

- arquiteturas organizadas em malha (*mesh based*)
- arquiteturas organizadas em matriz linear
- arquiteturas organizadas em anel
- arquiteturas organizadas em *crossbar*

Embora pouco utilizadas, ainda poderiam ser encontradas arquiteturas dos seguintes tipos:

- Barramento
- Estrela
- Toro
- Matriz tridimensional
- Hipercubo
- Totalmente conectada
- *Switch*

2.5 MPSoC (Multi-Processor SoC)

Estes sistemas consistem em múltiplos processadores interligados por uma rede de interconexão. Podem ser homogêneos ou heterogêneos. Estes sistemas têm como vantagem combinar as características de arquiteturas específicas em um mesmo chip, como por exemplo um bloco DSP e um bloco reconfigurável, através de uma rede de interconexão.

2.5.1 Multiprocessor DSP / MVP (Multimedia Video Processors)

Este tipo de processador consiste em um chip com múltiplos processadores DSP e um processador de propósito geral (Kim, 1996; Tremeac, 1998). Na Figura 2.19 (a) pode ser visto um diagrama geral de uma arquitetura deste tipo e em (b) da arquitetura do MVP TMS320C80, que tem uma performance de cerca de 2Gops (Kim, 1996). O controle dos múltiplos processadores é realizado através de uma palavra de instrução muito longa para cada ADSP (*Advanced DSP*) . Em geral estendem um pouco as limitações dos DSPs VLIW. Na Figura 2.19(c) pode-se observar um diagrama de blocos do multiprocessor DSP da Cradle tecnologies. Esta arquitetura atinge cerca de 96 Gmac (inteiro, 8 bits) para uma unidade com 16 DSPs e 8 processadores de propósito geral a 375MHz.

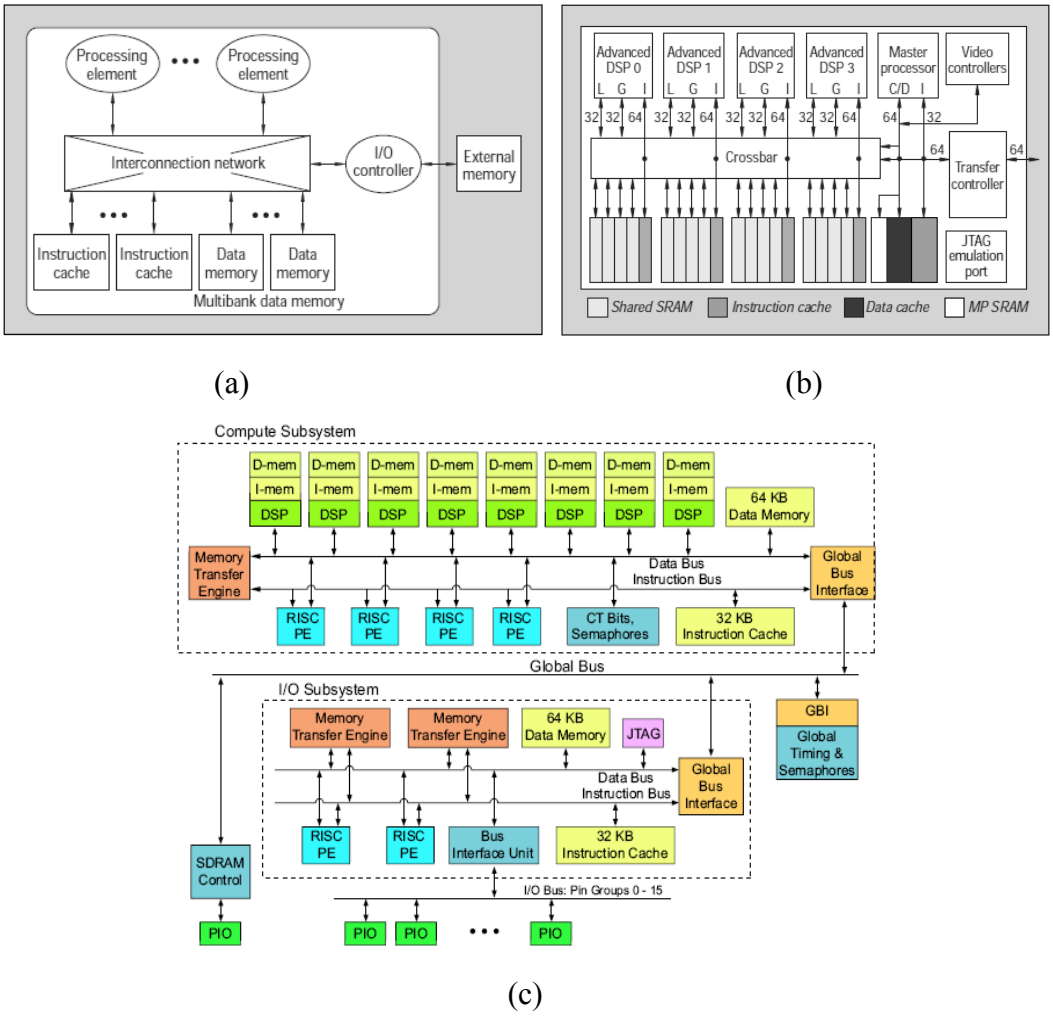
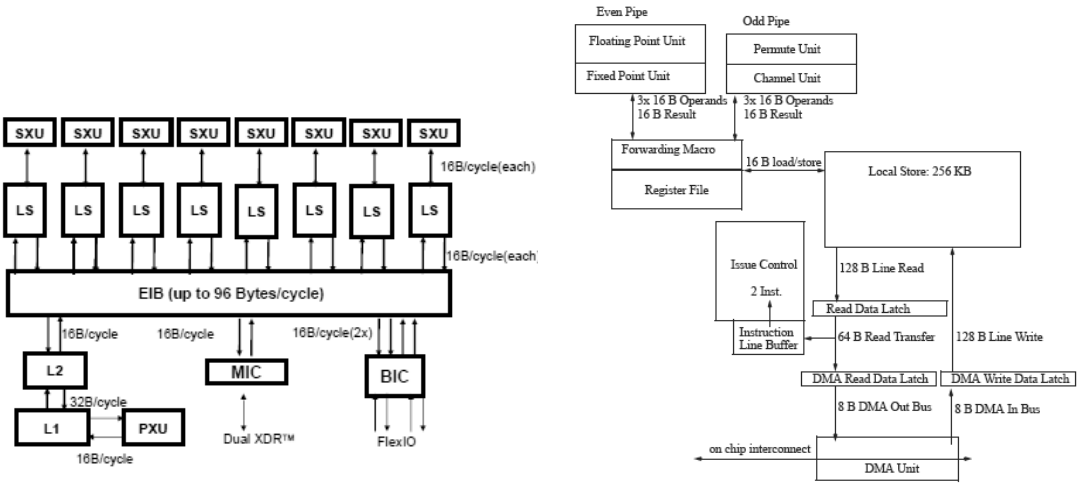


Figura 2.19: (a) arquitetura geral de um Multiprocessador DSP; (b) diagrama da arquitetura do MVP TMS320C80 (Kim, 1996); (c) Multiprocessor DSP da Cradle tecnologies.

2.5.2 Processador Cell e o Elemento Processador Sinergético (SPE)

O processador CELL é um conjunto com um processador PowerPC™ e 8 processadores vetoriais, formando um sistema multiprocessado em um único chip, e um desempenho estimado de 256 Gflops em uma freqüência de 4GHz. Na Figura 2.20(a) pode-se observar um diagrama do processador CELL.

O Elemento Processador Sinergético (*Synergistic Processor Element*) é uma arquitetura voltada para aplicações de *streaming* e multimídia (Asano et al., 2005). Nesta figura, cada SPE é composto por uma memória local LS (*Local Store*), e uma unidade de processamento SXU (*Synergistic eXecution Unit*), as quais são interconectadas através do EIB (*Element Interconnect Bus*). Esta estrutura não é na verdade um barramento, mas quatro anéis unidirecionais (com dois conjuntos de dois anéis, cada conjunto realizando transferência em uma direção), cada um com 16 bytes de largura. Nesta arquitetura a memória não é um cache, mas uma memória local. Na Figura 2.20(b) pode-se observar a organização do SPE com mais detalhes.



(a) (b)

Figura 2.20: (a) Diagrama do processador CELL (D. Pham, 2005); (b) Organização do SPE.

2.6 Componentes utilizados nas arquiteturas

Alguns elementos são comuns na composição de muitas das arquiteturas apresentadas. Eles serão apresentados em mais detalhes a seguir.

2.6.1 Elementos de Processamento

As arquiteturas que se baseiam em matrizes de elementos de processamento têm em geral P.E.s que compartilham as seguintes características em comum:

- Banco de registradores
- ULA
- Multiplicador
- MAC
- Multiplexadores (seleção de entrada)

Exemplos de elementos de processamento presentes em algumas arquiteturas podem ser vistos na Figura 2.21. Na Figura 2.8(b) pode ser visto um elemento de processamento de um chip de visão e na Figura 2.11(b) um elemento de processamento do sensor de visão inteligente. Nestes casos, para acomodar um grande número de elementos de processamento, os processadores foram desenvolvidos de forma bit-serial.

Como pode-se observar, em todos os casos mostrados o elemento de processsamento é composto por uma ULA e bancos de registradores. Em alguns casos, a ULA é acompanhada de um multiplicador. O uso da técnica de desenvolvimento bit-serial é comum para a redução da área, em troca de um tempo de execução maior. O uso destes elementos, contudo, pode ser interessante para a realização de operações sobre imagens binárias.

Um segundo ponto a ser observado é o controle do elemento de processamento. A grande maioria das arquiteturas opera de modo SIMD, com um único controle para todos os elementos de processamento, com algumas poucas exceções, tais como por exemplo o Systolic Ring e as arquiteturas de grão mais grosso. A presença de um controle no P.E. Tem um custo maior em área, mas adiciona mais flexibilidade à arquitetura, permitindo diferentes formas de implementação de um algoritmo.

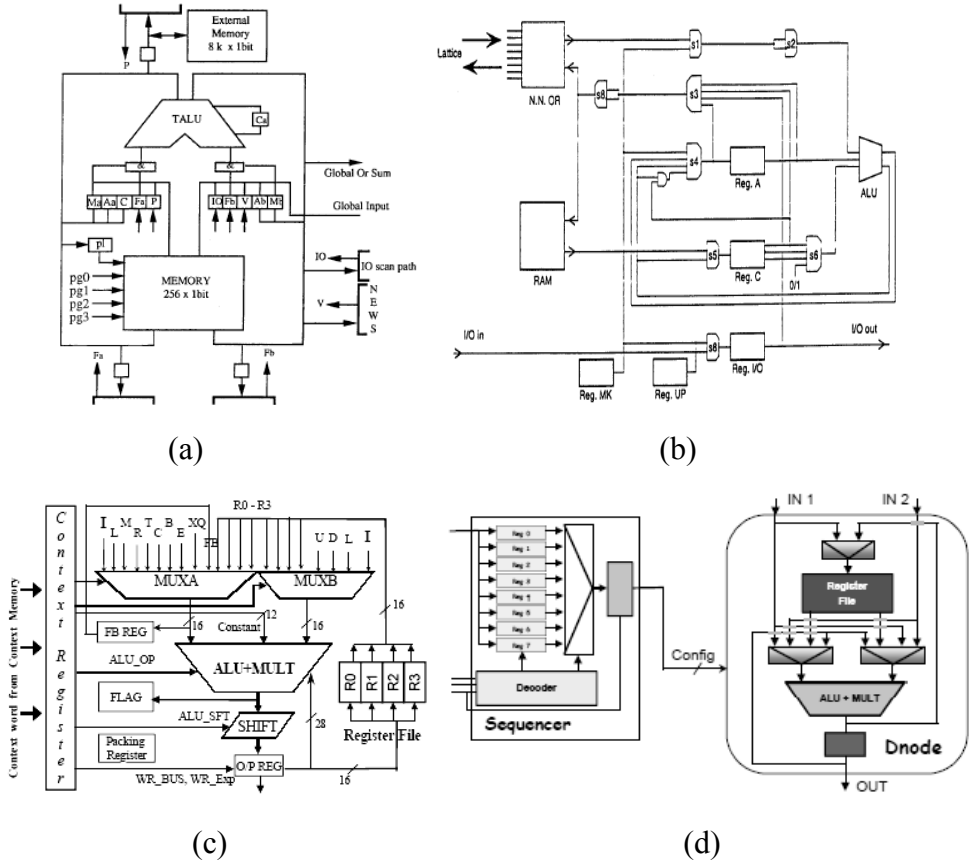


Figura 2.21: Elementos de Processamento de algumas arquiteturas. (a) Sphinx (Bouaziz et al., 1991); (b) Papia II (Albanesi et al., 1994); (c) Morphosys (Singh et al., 1998); (d) Systolic Ring (Sassatelli et al., 2002);

Um terceiro ponto é a presença ou não de uma memória no P.E., que está presente em algumas arquiteturas, tais como a arquitetura SIMPil. A presença desta memória permite que o acesso aos dados seja realizado de uma maneira mais rápida, comparando com uma memória global (onde vários P.E.s concorreriam para o seu acesso) acessada diretamente ou através da passagem de mensagens.

Na Tabela 2.1 é apresentada uma comparação entre as principais arquiteturas multiprocessadas atuais, destacando as idéias centrais envolvidas em cada uma.

Tabela 2.1: Comparação entre as arquiteturas multiprocessadas

Arquitetura	Pontos fortes	Pontos fracos
• Stream Processor	• hierarquia de memória para aumento da largura de banda	• troca de "kernel" tem custo elevado para streams curtos
• Chip de Visão / Processador do Plano Focal / Sensor de Visão Inteligente	• processamento + sensor	• resolução, e exploração do paralelismo em níveis mais altos
• Computational RAM / Arquiteturas com memória CAM	• processamento + memória	• programabilidade (ops. baseadas em busca)
• Multiprocessor SoCs	• integração de múltiplas unidades em um SoC	• ponto a melhorar: estrutura de comunicação / unid. de ponto flutuante

2.6.2 Infra-Estrutura de Comunicação

2.6.2.1 Tipos

Uma revisão bastante completa das redes de interconexão pode ser encontrada em (Zeferino, 2003). Dentre as diversas estruturas utilizadas para comunicação entre os elementos de processamento, as mais comuns são:

1 *Conexões Ponto a Ponto*

As conexões ponto a ponto são comunicações dedicadas, que interligam pontos específicos. É comum ter um alcance relativamente pequeno dentro do chip. Contudo, devem ser reprojetadas para cada arquitetura. Além disto, o custo torna-se elevando quando considera-se um conjunto de N elementos que devem ser totalmente interconectados uns com os outros.

2 Barramento

Um barramento é um canal de comunicação bidirecional compartilhado por diversos elementos em um sistema. Existem já alguns padrões bem conhecidos, como por exemplo OCP, VCI, Core Connect, PCI, AGP, o que permite o seu reuso em vários projetos e facilita a padronização no desenvolvimento da interface de comunicação em IPs. A principal restrição é que os elementos disputam o acesso ao barramento, produzindo um compartilhamento da largura de banda, o que limita a capacidade de comunicação e obriga a trabalhar em freqüências mais elevadas. Uma infra-estrutura de comunicação baseada em barramentos pode ter seu desempenho melhorado com a utilização de uma hierarquia, na qual diversos barramentos são interconectados por chaves.

3 *Rede Intra-chip (NoC)*

NoC é um tipo de infra-estrutura de comunicação desenvolvida para interconectar uma grande quantidade de núcleos, de forma distribuída e que possui escalabilidade. É

composta por um conjunto de roteadores, conectados quase sempre a um núcleo cada um, dispostos em uma dada topologia (como por exemplo malha ou toro) e interconectados com os roteadores vizinhos na rede. Este arranjo permite a transferência de dados entre diferentes núcleos, que podem estar próximos ou distantes, sem interferir nos núcleos não envolvidos na comunicação e que encontram-se no caminho. Já existem diversos tipos de NoCs, e uma coletânea com as principais arquiteturas e descrições, bem como um estudo e desenvolvimento de uma NoC que é utilizada neste trabalho (SoCIN) pode ser encontrada em (Zeferino, 2003). Um dos pontos críticos da NoC é o custo dos roteadores utilizados quando estes possuem uma área significativa frente a dos elementos de processamento. Outro ponto que merece atenção é o tráfego dos dados na rede, que pode provocar redução no desempenho devido aos congestionamentos. A influência deste fator pode ser minimizada através do reposicionamento dos núcleos buscando reduzir a distância percorrida pelos dados (Kreutz, 2005).

Na Tabela 2.2 pode-se observar uma comparação entre os principais tipos de infra-estrutura de comunicação utilizadas no projeto de sistemas digitais.

Tabela 2.2: Comparação entre os principais tipos de infra-estrutura de comunicação

Estrutura	Pontos fortes	Pontos fracos
• Barramento	• Vários elementos são facilmente inter-conectados pelo sistema	• necessita de alta velocidade devido ao compartilhamento do meio de comunicação • maior consumo
• Ligações Ponto a ponto	• mais usadas em arquiteturas dedicadas • ideal para um padrão de comunicação determinado	• é necessário reprojetar a cada nova arquitetura • o custo pode ser elevado para muitos elementos
• NoC	• possibilita múltiplas transferências simultâneas • o tamanho da rede tem pouca influência sobre a freqüência de operação	• custo maior em área • insere uma latência nas comunicações

2.6.2.2 Topologias

De acordo com a disposição espacial dos elementos de processamento pode-se agrupar as infra-estruturas de comunicação nas seguintes classes:

- Linha/coluna

Este tipo de arquitetura é geralmente utilizado para macropipelines ou para realizar processamento da imagem em linhas ou em colunas, com os elementos de processamento operando em paralelo sobre a linha. Em (Lindgren et al., 2005) um sensor de visão inteligente é apresentado, onde diversos elementos de processamento dispostos em linha são responsáveis pelo processamento dos dados das colunas do sensor.

- Anel

Aqui os dados seguem um caminho fechado, retornando ao ponto inicial diretamente ou através de um segundo caminho de realimentação. Um exemplo está na Figura 2.18 (Sassatelli et al., 2002).

- Matriz

Aqui os elementos de processamento são dispostos de forma regular na forma de uma matriz, onde cada elemento de processamento pode comunicar-se com os quatro elementos vizinhos (acima, abaixo, esquerda e direita), exceto nas bordas da matriz. Em alguns casos também é realizada a comunicação com os vizinhos das diagonais. Um exemplo de disposição dos elementos de processamento na forma de matriz pode ser observada na arquitetura da Figura 2.8(a).

- Toro

Esta topologia possui os elementos de processamento dispostos da mesma forma que os elementos da matriz, mas com os elementos da borda superior conectados aos elementos da borda inferior e os da borda esquerda conectados aos da borda direita. Este tipo de topologia praticamente não é encontrado em arquiteturas de processamento de imagens, pois não é comum encontrar operações envolvendo simultâneamente os extremos de uma imagem, e as comunicações limitam-se a uma vizinhança pequena em relação ao tamanho da imagem.

- Toro dobrado

Esta topologia é uma variação do toro 2D, na qual os elementos da rede tem o comprimento de suas interconexões reduzidas através de um rearranjo das mesmas, análogo ao dobramento de um toro, nas duas dimensões, no qual os elementos da extremidade são colocados de forma entrelaçada no interior da rede. Esta topologia pode ser observada na Figura 2.22. Contudo, da mesma forma que o toro, não é encontrada em arquiteturas para processamento de imagens pela mesma razão já citada para o toro.

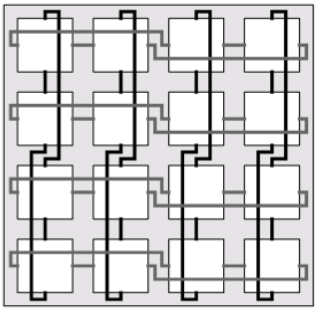
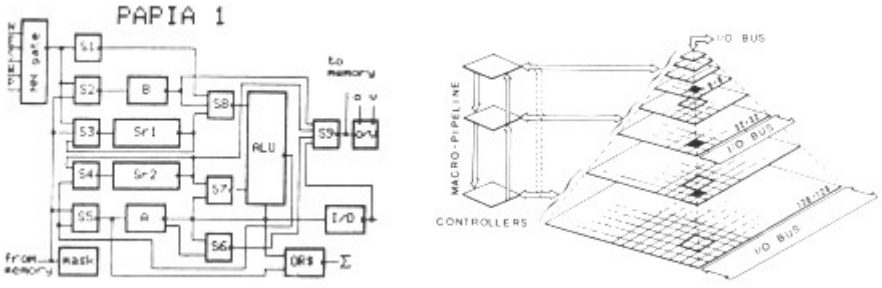


Figura 2.22: Toro Dobrado

- Pirâmide

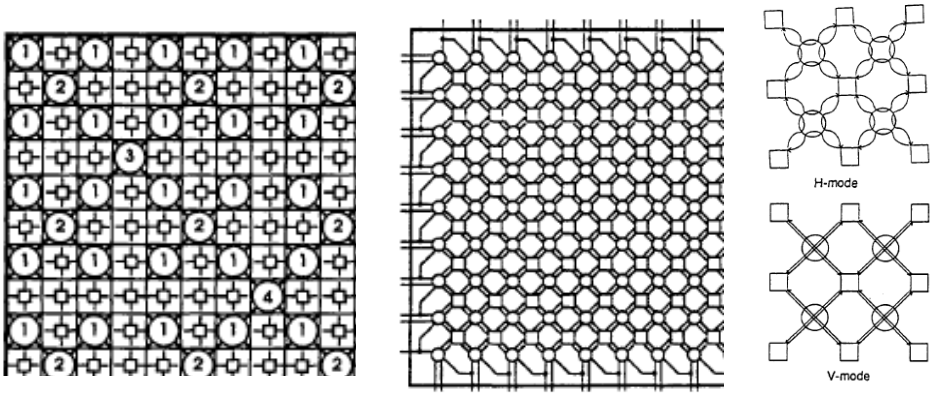
Este último tipo já foi alvo de muita pesquisa, principalmente quando os elementos de processamento eram constituídos por um chip. Exemplos de arquiteturas deste tipo são a arquitetura Papia (*Pyramid Architecture for Image Analysis*), que é mostrada na Figura 2.23, e a GAM (Uhr, 1987). A arquitetura Papia 1 foi alvo de estudos sobre a construção da pirâmide utilizando um leiaute bidimensional, que originou a arquitetura Papia 2 e

que é apresentado na Figura 2.24. Nesta segunda versão, os elementos de processamento do nível mais baixo (base da pirâmide) ocupam toda a matriz. Os elementos do segundo nível ocupam as posições indicadas na figura pelo número 1, e assim sucessivamente. A rede de interconexão escolhida é composta por chaves, que permitem conectar cada elemento com seus vizinhos na horizontal, vertical e diagonal.



(a) (b)

Figura 2.23: Arquitetura piramidal Papia. (a) Elemento de processamento; (b) interconexão entre os elementos de processamento (Uhr, 1987).



(a) (b) (c)

Figura 2.24: Planarização de arquitetura piramidal (Papia 2). (a) posição dos elementos, onde os elementos da base (nível 0) ocupam toda a rede; (b) Infra-estrutura de comunicação utilizada; (c) modos de conexão (Albanesi et al., 1994).

2.7 Conclusões

As arquiteturas que realizam processamento serial simples ou com a exploração do paralelismo através de pipeline necessitam de freqüências de operação extremamente altas para aumentar seu desempenho. Processadores de propósito geral atuais operam na faixa de 4GHz, e são capazes de realizar apenas operações simples sobre imagens em uma taxa próxima à do sinal de vídeo (30 fps) em uma resolução padrão (720x480). Alguns pequenos ganhos podem ser obtidos através da especialização do processador, como nos ASIPs ou nos DSPs. Já arquiteturas que utilizam paralelismo maciço, como no caso dos *vision chips*, existe um grande ganho de desempenho, mas as operações

realizadas são extremamente simples, e a resolução das imagens é muito baixa, devido à limitações tecnológicas, o que reduz a sua aplicação. Entretanto, as arquiteturas que encontram-se entre os extremos, tais como os *multiprocessor* SoCs mostram-se as mais promissoras, pois através da utilização de elementos de processamento mais complexos pode-se, da mesma forma, realizar operações mais complexas, e também com um grau de paralelismo (entre os P.E.s) que permite obter um aumento significativo no desempenho. A exploração do paralelismo dentro do elemento de processamento é um segundo recurso que merece ser estudado um pouco mais a fundo, pois permite aumentar o ganho já obtido as custas do paralelismo entre P.E.s, mas sem restringir o grau de complexidade das operações que podem ser realizadas pela arquitetura, principalmente se for explorado para aplicações específicas. A questão da comunicação entre os elementos de processamento também merece ser explorada, uma vez que a maioria das arquiteturas com um maior grau de paralelismo assume que todas as comunicações se darão apenas com elementos vizinhos, desprezando outras formas de exploração do paralelismo e nas quais pode existir a necessidade de comunicação com elementos mais distantes, como um processador central ou um elemento de I/O. Além disto, o uso de um barramento como elemento de interconexão para os elementos de processamento, como pode ser visto na arquitetura do *multiprocessor* DSP (Figura 2.19) pode-se mostrar um gargalo para determinados algoritmos, e é um ponto que será explorado neste trabalho.

3 ALGORITMOS E PARTICIONAMENTO PARA EXPLORAÇÃO DO PARALELISMO

É apresentada a seguir uma descrição dos algoritmos utilizados para estudo de caso nos experimentos realizados, os quais buscam explorar as diferentes formas de utilização do paralelismo no desenvolvimento do trabalho.

3.1 Algoritmos Com Grande Demanda Computacional

Tendo em vista que são considerados neste trabalho alguns algoritmos que têm grande demanda computacional (para os padrões atuais) e que só podem ser resolvidos em tempo real através da exploração do paralelismo, segue a seguir a descrição básica destes algoritmos:

3.1.1 Algoritmos Convencionais

3.1.1.1 Detecção de padrões

As imagens consideradas neste trabalho são estruturas de dados bidimensionais. Estes dados provém, em geral, de captura de da luz proveniente de uma cena real tridimensional projetada sobre a superfície de um sensor bidimensional através de um sistema óptico. O sensor é composto por duas partes principais, um dispositivo sensível a uma banda do espectro eletromagnético e que produz uma saída proporcional à energia que provocou a excitação. O segundo é o digitalizador, que converte o sinal elétrico de saída do sensor para uma forma digital (Gonzales, 1993). Existem também outras formas de produção de imagens, como: reflexão de ultrassom, absorção de raios-x, tomografia, etc.

Considera-se também que a imagem é uma função bidimensional f(x,y) das coordenadas discretas em um plano XY. O valor de f em um ponto do plano é denominado intensidade luminosa I, ou nível de cinza naquele ponto. Esta função pode ser definida conforme a Equação 3.1.

Equação 3.1: $I(x,y) = f(x,y)$

A detecção de padrões (*Template Matching*) consiste em encontrar o casamento de uma sub-imagem w(x,y) de tamanho J x K em uma imagem f(x,y) de tamanho M x N, onde se assume que J ≤ M e K ≤ N (Gonzales, 1993). Uma maneira de determinar o casamento entre a imagem e a sub-imagem é empregar a correlação entre as duas matrizes, dada por

Equação 3.2: $c(s,t) = \sum_x \sum_y f(x,y)w(x-s,y-t)$

onde s=0,1,2,...,M-1 e t=0,1,2,...,N-1 e o somatório é calculado sobre a região onde w e f se sobrepõe.

Outra possível medida para a correlação é a soma das diferenças absolutas (SAD- *Sum of Absolute Differences*), dada por:

Equação 3.3: $SAD = \sum_x \sum_y |f(x,y) - w(x,y)|$

O algoritmo de detecção de padrões é utilizado freqüentemente quando se conhece os objetos que se procura na imagem e estes apresentam pouca variação.

3.1.1.2 Estimação de Movimento

O problema da detecção e compensação de movimento já é bastante estudado devido ao seu uso nos padrões de compressão de vídeo dominantes, que são MPEG2, MPEG4 e h.264. Ele consiste em prever o conteúdo de um quadro com base na movimentação de elementos de um quadro de referência. A transmissão destes movimentos e da diferença entre o real e o previsto elimina grande parte da redundância temporal do vídeo, reduzindo a largura de banda necessária para a sua transmissão. A determinação do movimento dos blocos consiste basicamente em comparar cada bloco da imagem a ser predito em uma certa área no quadro de referência e tomar como resultado da busca a posição onde houver o melhor casamento do bloco, através de um critério de avaliação, como por exemplo a soma das diferenças absolutas. Esta operação é denominada de casamento de blocos (*block matching*). O problema de detecção de padrões pode, desta forma, compartilhar estruturas para aceleração do cálculo da SAD com o algoritmo de estimação de movimento.

Este casamento de blocos pode ser acelerado através do uso do paralelismo, onde mais de um bloco são verificados na imagem de referência. Para isto, normalmente a imagem é dividida em diferentes regiões e diferentes elementos de processamento realizam essa busca em paralelo. Cabe ressaltar que o bloco a ser processado pode ter diferentes tamanhos, conforme o padrão de compactação onde será utilizado. Outro ponto é que nos padrões mais recentes de compressão de vídeo o casamento de blocos é realizado com precisão sub-pixel. Inicialmente é realizada uma busca com a resolução de um pixel. Em seguida é feita uma interpolação sub-pixel, gerando uma imagem de dimensões maiores. Por último, é feita a etapa final da busca nesta imagem aumentada. Várias soluções tanto no campo dos algoritmos como também no campo das arquiteturas, na maioria delas dedicadas, já foram apresentadas na literatura. Algumas poucas apresentam algum grau de programação, como por exemplo (Lin et al., 1995).

A busca completa é um algoritmo que exaustivamente determina a melhor solução. Existem alternativas que consideram aproximações destes algoritmos, como a busca guiada a partir do armazenamento de vetores de movimento anteriores. Outras abordagens consideram o uso de técnicas que empregam multi-resolução para encontrar de forma mais rápida uma solução próxima da ótima (Byung et al., 2005), mas com cobertura da mesma área de busca e refinamentos sucessivos dos vetores de busca. Neste trabalho serão consideradas duas soluções: a busca completa (FSBMA - *Full*

Search Block Matching Algorithm), com maior demanda computacional, com o processamento mais regular e resultado ótimo e a busca multi-resolução (MRBMA - *Multi-Resolution Block Matching Algorithm*) (Byung, 1999), com menor demanda computacional e com uma maior irregularidade no processamento.

3.1.1.3 Reconstrução Estéreo

Sistemas de visão estéreo determinam a profundidade através de duas ou mais imagens capturadas ao mesmo tempo a partir de pontos de vista ligeiramente diferentes. A tarefa mais importante e que consome mais tempo é o registro das imagens, isto é, a identificação dos pixels correspondentes (Hirschmüller, 2002). Nesta definição, o termo profundidade refere-se à distância de um ponto no espaço, e que aparece na imagem, até a câmera. Em geral, os sistemas desenvolvidos para esta tarefa empregam correlação e janelas de tamanho fixo. Como medida de correspondência entre os pixels pode ser utilizada a SAD, da mesma maneira como no problema de estimação de movimento. Um problema típico desta metodologia ocorre quando há descontinuidades, como por exemplo nas bordas dos objetos. Neste caso, o método das janelas tende a apresentar resultados com borramento nas bordas ou perda de detalhes em objetos, pois assume que todos os pixels na janela possuem o mesmo valor de profundidade. Em (Hirschmüller, 2002), um método para reduzir este problema é apresentado, produzindo imagens de melhor qualidade, mas às custas de uma demanda computacional maior. Este método emprega múltiplas janelas de correlação, tipicamente 9 ou 25, e com base na análise do resultado da SAD deste conjunto de janelas é possível decidir qual o melhor valor de profundidade para o ponto. Este problema mostra-se também um interessante estudo de caso, uma vez que os operadores que necessitam de maior processamento são muito parecidos com o do problema de estimação de movimento, ou seja o cálculo de diferenças absolutas.

3.1.1.4 Segmentação de Imagens

A segmentação da imagem tem por objetivo dividi-la em suas partes constituintes ou objetos (Gonzales, 1993). O nível no qual esta subdivisão é realizada depende do problema que está sendo resolvido, ou seja, a segmentação deve parar quando os objetos de interesse em uma aplicação foram isolados. Neste trabalho serão considerados dois tipos de segmentação: através da descontinuidade da intensidade dos pixels na imagem e segmentação através do movimento. No primeiro caso, as bordas mais significativas da imagem são localizadas e as regiões entre estas bordas são delimitadas. Este caso pode ser observado na Figura 3.1. Já no segundo caso, são verificados na imagem os locais onde ocorreu movimento e estes locais são agrupados em regiões, correspondendo a objetos que se movimentaram ou se deformaram. De acordo com o tamanho da região e a quantidade de movimento, pode-se separar em alguns casos os objetos que se movimentam em um plano de fundo que também se movimenta. Se os vetores de movimento são calculados utilizando blocos de pixels, esta segmentação terá uma resolução menor do que a da imagem. Este tipo de segmentação pode ser visto na Figura 3.2.Outros métodos de segmentação existem, tais como segmentação por textura, mas não serão tratados aqui.

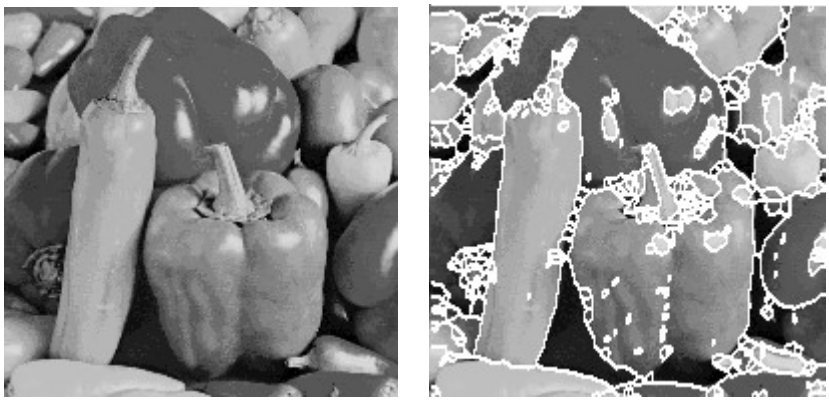


Figura 3.1: Exemplo de operação de segmentação por tons de cinza

(a) (b) (c)

Figura 3.2: Segmentação utilizando o movimento; (a) imagem da seqüência; (b) vetores de movimento; (c) região da imagem da seqüência onde houve movimento.

3.1.2 Algoritmos Multi-Resolução

O uso de algoritmos Multi-Resolução tem como principal vantagem a redução da necessidade de computação de algoritmos muito exigentes, tais como os apresentados. Isto, embora reduza o tempo de processamento, ainda não é o suficiente para que certas aplicações atinjam o desempenho desejado, como por exemplo a estimação de movimento em imagens de resolução muito alta ou com muitos quadros de referência. Além disto, para a utilização de um algoritmo multi-resolução é necessário gerar diferentes planos de imagens com resoluções menores, na forma de uma pirâmide. Um exemplo de algoritmo multi-resolução é o MRBMA, já mencionado, para a estimação de movimento. Outro exemplo é a busca de objetos em imagens. Pode-se observar na Figura 3.3 uma pirâmide de imagens com diferentes resoluções, e a busca de um objeto na figura.

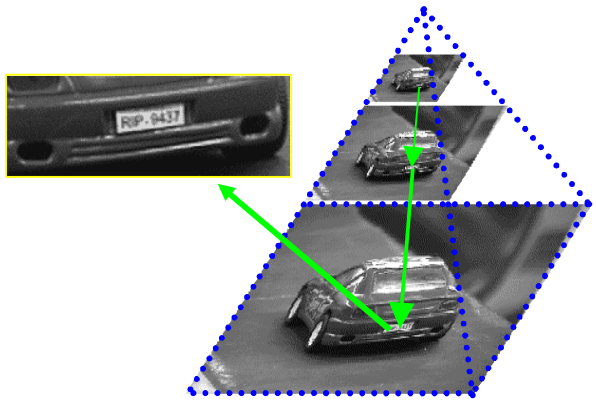


Figura 3.3: Busca de um objeto (placa do carro) em uma pirâmide de imagens.

A geração dos níveis neste trabalho seguirá o mesmo método e numeração apresentado em (Byung et al., 1999), sendo que cada pixel do nível superior (imagem de menor resolução) é obtido através da média dos quatro pixels do nível inferior. Esta estrutura é conhecida como pirâmide de blocos de média (*block mean pyramid*). As dimensões padrão das imagens são potências de dois ou somas de potências de dois, e deve-se escolher um número de níveis tal que não ocorra problemas na redução da resolução. Uma dimensão com um número ímpar de píxels, por exemplo, não poderá ter um nível superior sem que sejam produzidos artefatos em uma das bordas. Deve-se considerar também que um nível superior com resolução muito reduzida poderá não ter mais informação útil (perdida através do cálculo da média) para utilização em conjunto com os níveis inferiores. Este método é bastante adequado, uma vez que a operação de média entre quatro valores tem baixo custo em hardware. O topo da pirâmide terá nível igual a zero.

3.1.2.1 Detecção de Padrões Multi-Resolução

Neste método, ao invés de realizar a varredura do padrão em sua resolução normal sobre toda a imagem, calculando a correlação para todas as posições, realiza-se a mesma operação em resoluções menores e se calcula em uma resolução maior somente quando houver um casamento da imagem com uma correlação acima de um valor pré-determinado. Outra opção é utilizar um padrão em uma resolução menor sobre a imagem, e nas posições de maior correlação verificar com o padrão completo (Nassif, 1997). Este método não é adequado para objetos muito pequenos, que tendem a perder muita informação conforme a imagem sofre redução da resolução.

3.1.2.2 Estimação de Movimento Multi-resolução

A utilização de um casamento de blocos em múltiplas escalas reduzirá o volume de computação em comparação com o da busca completa. Resultados de desempenho em uma arquitetura dedicada podem ser observados em (Byung et al., 1999). Neste método, são produzidas réplicas da imagem original em resoluções menores, seguindo as potências de 2. A partir daí são realizadas sucessivos casamentos de blocos partindo das imagens de menor resolução até se chegar na imagem com a resolução original. Da

mesma forma, os vetores de movimento encontrados em resoluções menores são utilizados nas imagens de resolução maior como ponto de partida, no entorno do qual será realizada a busca e assim sucessivamente. O resultado final será um vetor de movimento relativo a imagem com a resolução original e dentro da distância d, mas obtido através de sucessivas buscas dentro de uma distância menor dm. Assim, o tempo de computação será proporcional a dm^2 e não a d^2, onde dm é menor do que d. Na Figura 3.4 pode-se observar os vetores de movimento e seu refinamento através de imagens com resolução crescente. No nível de resolução mais baixa mostrado no exemplo, não foi detectado movimento. Mais detalhes podem ser encontrados em (Soares, 2006).

Figura 3.4: Exemplo de detecção de movimento multi-resolução. Aqui a pirâmide é composta por quatro níveis.

3.2 Particionamento dos algoritmos para exploração do paralelismo

Os algoritmos podem ser particionados de diferentes formas para a sua execução em paralelo. Duas formas de particionamento serão tratadas aqui, o particionamento do algoritmo em tarefas iguais e o particionamento do algoritmo em uma seqüência de tarefas.

3.2.1 Particionamento do algoritmo em partes regulares ou decomposição de domínio

Grande parte dos algoritmos de processamento de imagens aponta para uma divisão da imagem em partes regulares, ou grupos de pixels, sobre as quais serão executadas as mesmas operações. A imagem pode ser processada de diferentes maneiras de acordo com a forma de associação dos elementos de processamento aos seus pixels, ou seja, conforme o tamanho das regiões, como descrito a seguir:

a)Processamento Pixel a Pixel

O processamento da imagem é realizado através de uma varredura da imagem, em geral da esquerda para a direita e de cima para baixo, seqüencialmente, considerando ou não uma vizinhança no entorno de cada pixel. Um exemplo de arquitetura que emprega este tipo de varredura pode ser encontrada em (Velten, 2004), onde uma arquitetura para

operações morfológicas é apresentada. Nesta arquitetura, uma vizinhança dos pixels que inclui as linhas anteriores é armazenada e utilizada para o processamento de cada pixel.

b)Processamento por Grupos de Pixels

A imagem é dividida em regiões, e cada região será processada de forma independente e paralela. Exemplos de arquiteturas que utilizam este tipo de processamento são encontradas em (Wills et al., 1996b; Morris et al., 1999).

c)Processamento por Pixel

Este é um caso limite da forma b), onde o tamanho da região é igual a um pixel. Exemplos de arquiteturas que utilizam este tipo de processamento podem ser encontradas em (Kagami et al., 2002; Kagawa, 2004).

Os efeitos de borda que podem surgir ao dividir a imagem devem ser tratados (Pitas, 1993). Neste sentido, foram realizados experimentos com segmentação de imagens binárias e com estimação de movimento.

3.2.1.1 Estimação de movimento

O algoritmo para estimação de movimento já descrito pode ser particionado e executado em paralelo através da divisão em regiões tanto da imagem atual quanto da imagem de referência, dentro das quais é realizada uma busca dos blocos da imagem atual na imagem de referência.

Cada imagem terá dimensões T_{ix} por T_{iy}, e poderá ser dividida em regiões de tamanho T_{rx} por T_{ry} pixels no particionamento de algoritmos para a execução por múltiplos processadores. O número de regiões por imagem será igual ao número de processadores por imagem: P_x por P_y, onde

Equação 3.4: $$P_x = \frac{T_{ix}}{T_{rx}}$$

Equação 3.5: $$P_y = \frac{T_{iy}}{T_{ry}}$$

O bloco terá dimensões de 4 por 4 pixels, que é o mínimo tamanho de bloco no padrão h.264, um dos mais atuais e eficientes padrões de compressão atuais (Richardson, 2003). O número de blocos em uma região será igual a m por n, onde

Equação 3.6: $$m = \frac{T_{rx}}{4}$$

Equação 3.7: $$n = \frac{T_{ry}}{4}$$

Estas dimensões podem ser observadas na Figura 3.5.Assume-se que quando o processamento em um bloco da imagem necessitar de dados de elementos vizinhos, serão realizadas comunicações com estes vizinhos. No caso de operações que ocorram sobre as fronteiras, a região adjacente à fronteira será enviada para um o elementos de processamento à esquerda, acima e na diagonal superior esquerda (se existirem), os

quais realizarão as operações necessárias e devolverão os resultados para o elemento vizinho, de acordo com o algoritmo.

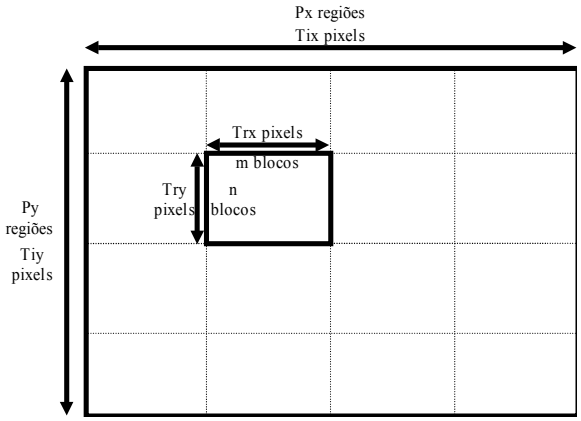


Figura 3.5: Definições utilizadas no modelo do problema

Será adotada a passagem dos dados do elemento da direita da fronteira para o da esquerda, para que este realize as operações de fronteira. Também será adotada a passagem dos dados do elemento de baixo da fronteira para o elemento de cima da fronteira. As comunicações de um elemento de processamento podem ser vistas na Figura 3.6. As direções, que se referem a qual região vizinha os dados serão enviados, estão indicadas pelas letras de A a P. A região cinza corresponde a coluna, linha e bloco em diagonal da imagem de referência das regiões vizinhas e que serão recebidas pela região mostrada. Neste caso específico da imagem de referência, outra possibilidade é a carga da imagem no elemento de processamento já considerando esta sobreposição, a qual exige um percentual adicional de memória relativamente pequeno.

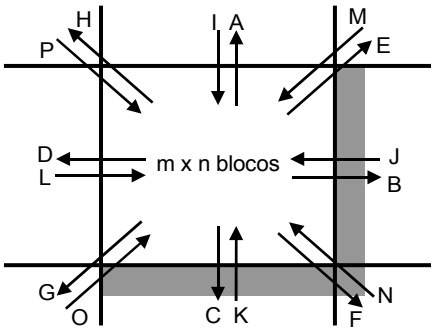


Figura 3.6: Região da imagem e comunicações de um elemento de processamento

Considera-se que cada região será atribuída a um elemento de processamento (P.E.) e uma mesma região tanto da imagem atual quanto da imagem de referência serão atribuídas ao mesmo P.E. No caso dos blocos situados próximos à fronteira, quando a busca envolver uma região vizinha da imagem de referência, envia-se o bloco da imagem atual para o elemento vizinho realizar a sua busca.

Aqui, a computação será relativamente constante em cada P.E., mas a comunicação será fortemente dependente do número de regiões. Um maior número de regiões acarretará

em uma maior quantidade de blocos próximos a uma fronteira, e que deverão ser enviados a um elemento vizinho, produzindo um aumento do tempo de comunicação.

Para a realização da busca de um bloco 4x4 em uma região a uma distância d do centro, cujo número de possíveis posições é igual a

Equação 3.8: $Np=(2d+1)^2$

e assumindo-se que:

- se possui uma parte operativa que possibilite o cálculo de uma diferença absoluta em NC_{SAD} ciclos entre 2 blocos de 4x4 pixels (entre imagem de referência e imagem atual) em cada elemento de processamento
- a carga de um bloco de 4x4 pixels necessita de NC_{CB} ciclos
- a mudança de posição deste bloco sobre a imagem de referência necessita de NC_{MB} ciclos

o número de ciclos necessários para esta busca é igual a

Equação 3.9: $Ncb= NC_{CB}+NC_{MB}\,Np = NC_{CB}+NC_{MB}(2d+1)^2 =$

$$4\,NC_{MB}\,d^2+4\,NC_{MB}\,d+ NC_{MB}+NC_{CB}\ \text{ciclos}$$

Uma vez que um bloco da imagem atual se encontre em uma região adjacente da imagem de referência (ou seja a distância até a fronteira da região é menor do que a distância de busca), este bloco deve ser enviado para a região vizinha para que o processador desta continue com a busca.

Desta forma, o processamento será dividido em duas partes:

a)busca dos blocos da região da imagem atual armazenada no próprio processador;

b)busca dos blocos recebidos dos processadores vizinhos.

De acordo com a posição sobre a imagem, existirão P.E.s com diferentes vizinhanças (9 tipos de vizinhança no total). Assim, algumas das comunicações enumeradas não existirão. Da Tabela 3.1 até a Tabela 3.9 pode-se observar as comunicações de acordo com a posição de cada processador sobre a imagem. Aqui considera-se

Equação 3.10: $$D = \left\lfloor \frac{d}{4} \right\rfloor + 1,$$

que é a distância d, em pixels, convertida em D, que é múltiplo de 4 pixels (largura ou altura dos blocos de 4x4 pixels) e os símbolos $\lfloor\ \rfloor$ indicam o uso da função “maior inteiro”. Os blocos que encontram-se a uma distância menor de D blocos da fronteira deverão ser enviados para o elemento de processamento vizinho para continuar a realizar a busca nesta região, dentro da distância d da posição original do bloco. As comunicações são as indicadas na Tabela Tabela 3.1. Na Tabela 3.10 estão contabilizadas as comunicações para um nível, isto é, um conjunto de processadores atuando sobre a imagem em uma determinada resolução. N é o nível de resolução, utilizado na representação em pirâmide, já apresentada.

Tabela 3.1: Comunicações para um processador na região central da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
B	(D-1)•n		
C	(D-1)•m		
D	D•n	n	
E	D•(D-1)		
F	(D-1)•(D-1)		
G	D•(D-1)		
H	D•D	1	
I			D•m
J			(D-1)•n
K			(D-1)•m
L			D•n
M			D•(D-1)
N			(D-1)•(D-1)
O			D•(D-1)
P			D•D
Total	$4•D^2+(2•m+2•n-4)•D+(1-m-n)$	m+n+1	$4•D^2+(2•m+2•n-4)•D+(1-m-n)$

Tabela 3.2: Comunicações para um processador na esquerda da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
B	(D-1)•n		
C	(D-1)•m		
E	D•(D-1)		
F	(D-1)•(D-1)		
I			D•m
J			(D-1)•n
K			(D-1)•m
M			D•(D-1)
N			(D-1)•(D-1)
Total	$2•D^2+(2•m+n-3)•D+(1-m-n)$	m	$2•D^2+(2•m+n-3)•D+(1-m-n)$

Tabela 3.3: Comunicações para um processador na direita da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
C	(D-1)•m		
D	D•n	n	
G	D•(D-1)		
H	D•D	1	
I			D•m
K			(D-1)•m
L			D•n
O			D•(D-1)
P			D•D
Total	$2•D^2+(2•m+n-1)$ •D-m	m+n+1	$2•D^2+(2•m+n-1)$ •D-m

Tabela 3.4: Comunicações para um processador na borda superior da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
B	(D-1)•n		
C	(D-1)•m		
D	D•n	n	
F	(D-1)•(D-1)		
G	D•(D-1)		
J			(D-1)•n
K			(D-1)•m
L			D•n
N			(D-1)•(D-1)
O			D•(D-1)
Total	$2•D^2+(m+2•n-3)$ •D+(1-m-n)	n	$2•D^2+(m+2•n-3)$ •D+(1-m-n)

Tabela 3.5: Comunicações para um processador na borda inferior da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
B	(D-1)•n		
D	D•n	n	
E	D•(D-1)		
H	D•D	1	
I			D•m
J			(D-1)•n
L			D•n
M			D•(D-1)
P			D•D
Total	$2•D^2+(m+2•n-1)•D-n$	m+n+1	$2•D^2+(m+2•n-1)•D-n$

Tabela 3.6: Comunicações para um processador na borda superior à esquerda da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
B	(D-1)•n		
C	(D-1)•m		
F	(D-1)•(D-1)		
J			(D-1)•n
K			(D-1)•m
N			(D-1)•(D-1)
Total	$D^2+(m+n-2)•D+(1-m-n)$		$D^2+(m+n-2)•D+(1-m-n)$

Tabela 3.7: Comunicações para um processador na borda superior à direita da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
C	(D-1)•m		
D	D•n	n	
G	D•(D-1)		
K			(D-1)•m
L			D•n
O			D•(D-1)
Total	$D^2+(m+n-1)•D-m$	n	$D^2+(m+n-1)•D-m$

Tabela 3.8: Comunicações para um processador na borda inferior à esquerda da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
B	(D-1)•n		
E	D•(D-1)		
I			D•m
J			(D-1)•n
M			D•(D-1)
Total	D^2+(m+n-1) •D-n	m	D^2+(m+n-1) •D-n

Tabela 3.9: Comunicações para um processador na borda inferior à direita da imagem, durante o processamento.

Comunicação	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
A	D•m	m	
D	D•n	n	
H	D•D	1	
I			D•m
L			D•n
P			D•D
Total	D^2+(m+n) •D	m+n+1	D^2+(m+n) •D

Tabela 3.10: Comunicações para uma rede de Px•Py processadores assumindo m•n blocos 4x4 em cada região e busca em uma distância D.

Nível	Num. Proc.	Número de blocos da imagem atual	Número de blocos da imagem de referência	Número de blocos de diferença + vetor de movimento
N	Px•Py	(4•Px•Py-4•Px-4•Py+4)•D^2+ (-2•m•Px-2•n•Py+4•Px+4•Py+ 2•m•Px•Py+2•n•Px•Py-4•Px•Py-4) •D+ (m•Px+n•Py-Px-Py-m•Px•Py-n•Px•Py+Px•Py+1)	(Px•Py-Px)•m+ (Px•Py-Py)•n+ (Px•Py-Px-Py+1)	(4•Px•Py-4•Px-4•Py+4)•D^2+ (-2•m•Px-2•n•Py+4•Px+4•Py+ 2•m•Px•Py+2•n•Px•Py-4•Px•Py-4) •D+ (m•Px+n•Py-Px-Py-m•Px•Py-n•Px•Py+Px•Py+1)

3.2.1.2 Segmentação de imagens binárias

A segmentação de imagens binárias busca dividir basicamente a imagem em fundo e objetos. Os objetos são identificados pelo valor 1 na imagem e o fundo pelo valor 0, formato produzido pela etapa de aquisição da imagem. As diferentes áreas compostas por pixels conectados são identificadas, correspondendo no final do processo a objetos distintos.

A presença de regiões com concavidades impede que um algoritmo de varredura utilizando apenas dados da linha anterior seja utilizado, pois uma mesma região pode ser identificada como duas até que o ponto de união seja atingido. Utilizou-se um algoritmo que preenche todo objeto quando o primeiro pixel deste objeto for encontrado. O algoritmo e seu particionamento estão descritos em detalhes no Apêndice I. O tempo máximo de computação na etapa de varredura é proporcional à área das regiões, pois para cada pixel podem ser verificados os quatro vizinhos (caso o pixel não seja fundo). No caso do algoritmo particionado, o número de regiões também tem influência no tempo de computação, pois uma das etapas do algoritmo elimina a redundância na numeração das regiões (também descrito no Apêndice I).

3.2.2 Particionamento do algoritmo em partes irregulares ou decomposição funcional

Alguns algoritmos, devido ao uso de etapas de processamento complexas e irregulares, ou com menor grau de paralelismo, apontam para um particionamento por função. Neste caso, diferentes elementos de processamento podem realizar diferentes funções em paralelo, sob a forma de um macro-pipeline. Neste caso, os dados são enviados ao primeiro elemento de processamento da cadeia, são processados e o resultado passa para o segundo elemento, o qual realiza uma segunda função e assim por diante, até o final do processamento daquele conjunto de dados. Ao mesmo tempo em que o primeiro conjunto de dados foi enviado para o segundo elemento de processamento, o primeiro já pode receber um segundo conjunto de dados para serem processados, e assim sucessivamente.

Durante o processo, o número de conjunto de dados que podem ser processados em paralelo é igual ao número de elementos de processamento, da mesma forma em que em um processador com pipeline o número de instruções que podem ser processadas em paralelo é igual ao número de unidades funcionais no pipeline.

Um exemplo de algoritmo que pode ser particionado em funções é o algoritmo JPEG, no qual um possível particionamento compreenderia as funções conversão de cores, DCT, Quantização e codificação Hufmann.

3.2.3 Dependência de dados

Os algoritmos de processamento de imagens apresentam dependências de dados que são tratadas basicamente de duas maneiras: através de duplicação de dados ou troca de dados, nos algoritmos particionados por domínio.

Os algoritmos em que os dados ou pixels de saída dependem de uma vizinhança não muito grande no entorno do pixel, na ordem de 5x5 pixels, por exemplo, podem ser

tratadas através da duplicação destas áreas da imagem de entrada, pois representarão um custo adicional relativamente pequeno frente ao tamanho da imagem.

Já nos algoritmos tais como segmentação de regiões, onde o identificador de um pixel pode estar associado a uma região inteira, é mais adequado resolver a dependência através de comunicação entre os elementos de processamento para troca de dados. Esta característica gera um compromisso entre computação (tamanho das regiões na imagem) e comunicação. O destino da comunicação de cada elemento de processamento (de um elemento para o seu vizinho ou para um processador central, por exemplo) determinará o tipo de infra-estrutura de comunicação entre os elementos de processamento.

Algumas funções mais simples, como por exemplo a de filtragem, onde a vizinhança de entrada é muito pequena, são mais adequadas para o processamento de um pixel de cada vez do que para o processamento em paralelo. Estas funções podem ser realizadas durante o processo de transferência da imagem, com o armazenamento da vizinhança realizado através de *buffers* de linha, com um desempenho que atende aos requisitos de uma aplicação real que utilize câmeras comuns, com uma taxa de 30 quadros por segundo, e não justificam uma transferência completa da imagem para uma arquitetura com processamento maciçamente paralelo, pois o tempo de processamento seria muito menor do que o de transferência da imagem.

Em arquiteturas maciçamente paralelas, como em *vision chips*, nas quais todos os pixels são processados em paralelo, as dependências são tratadas através de comunicações com os elementos vizinhos mais próximos. Contudo, ainda existem sérias limitações ligadas à tecnologia que não permitem uma realização em hardware para imagens de alta resolução e que exigem que a função dos elementos seja extremamente simples, embora venham surgindo avanços nesta área (Lindgren et al., 2005). A transferência de dados para a memória onde estes serão processados faz também com que arquiteturas maciçamente paralelas se justifiquem de maneira mais forte quando o processamento está atrelado diretamente ao sensor, onde o gargalo da transferência de dados estará apenas na saída dos dados.

4 DEFINIÇÃO E DESENVOLVIMENTO DE UMA ARQUITETURA PARA PROCESSAMENTO PARALELO DE IMAGENS

Com base no que foi exposto sobre algoritmos e arquiteturas, propõe-se um conjunto de elementos capaz de realizar operações simples, como por exemplo comparação entre blocos, e complexas, como segmentação de imagens ou diferentes etapas de um algoritmo de compressão de imagens, através da exploração do paralelismo de operações em baixo e médio nível. Para a utilização do processamento paralelo, a imagem será particionada em regiões, que serão processadas em elementos de processamento (P.E.s), ou também o algoritmo poderá ser particionado em funções, como mostrado na Figura 4.1.

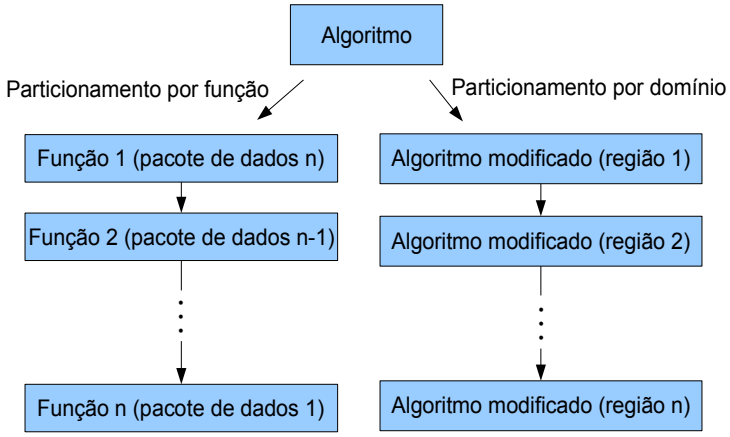


Figura 4.1: Particionamento do algorimo por função e por domínio para processamento paralelo.

Cada elemento de processamento contará com sua própria memória local, onde residirão as regiões da imagem e dados adicionais vindos de elementos vizinhos, bem como os resultados das operações. A atribuição de regiões da imagem abrangendo vários pixels aos P.E.s permitirá a utilização de operadores mais complexos do que seria possível atribuindo um pixel ou mesmo uma coluna da imagem. Isto ocorre porque, considerando a área total da arquitetura, um menor número de P.E.s permite a utilização de uma área maior para o desenvolvimento de cada P.E. Contudo, um menor número de P.E.s irá necessitar de operadores especializados para manter a performance em algumas operações. Isto também deve permitir o processamento de imagens com a resolução maior do que aquelas abrangidas pelos chips de visão, pois neles a área

ocupada é proporcional ao tamanho da imagem, se for considerada a definição de Kagami (seção 2.3.6), na qual é atribuído um elemento de processamento a cada pixel.

4.1 Escolha da infra-estrutura de comunicação

Dentre as possíveis escolhas para a infra-estrutura de comunicação, as duas que apresentam maior largura de banda na interconexão dos elementos de processamento são ligações ponto a ponto e o uso de uma rede intra-chip, por permitirem o paralelismo nas comunicações. Se for considerada somente a etapa de processamento e a intercomunicação com os vizinhos mais próximos, que é típica da maioria dos algoritmos de processamento de imagens, a escolha da primeira é mais adequada, pois não há o custo dos roteadores, os quais não são realmente necessários. Contudo, quando se considera uma arquitetura que pode operar tanto sobre algoritmos particionados por domínio, sobre regiões da imagem, quanto algoritmos particionados por função, realizando diferentes operações sobre uma imagem, na forma de um macropipeline, a NoC é uma estrutura que é mais adequada para permitir diferentes padrões de comunicação, entre diferentes grupos de elementos de processamento (como poderá ser visto no Capítulo 5.1). Neste caso, diferentes aplicações sendo executadas em paralelo, em macropipelines, nas quais cada uma pode ocupar diferentes quantidades de P.E.s podem manter sua comunicação sem, para tanto, interromper outros processadores entre os nodos de origem e o de destino. Além disso, a NoC permite que um sistema mais complexo seja construído, constituído por mais elementos tais como módulos de aquisição de imagens, sensores e outros que também se comuniquem sem interromper os processadores. Esta mesma rede também será responsável pela carga e descarga das imagens.

O sistema será modelado com os elementos de processamento síncronos localmente. Neste caso a opção pela utilização da comunicação baseada em créditos na rede resulta em uma maior performance nas transferências (1 *flit* de dados por ciclo, onde 1 *flit* transporta 1 byte), enquanto que transferências assíncronas tem uma duração 3 vezes maior devido ao *handshake*.

4.2 Arquitetura dos elementos de processamento

Diferentes arquiteturas para o elemento de processamento serão avaliadas, tendo em vista o desempenho da arquitetura. Aqui se busca avaliar diferentes alternativas das já apresentadas (elemento de processamento dedicado, elemento de processamento de propósito geral e elemento de processamento programável com especializações), e observar também o compromisso entre programabilidade e desempenho.

4.2.1 Alternativa 1:ASIC

Aqui buscar-se-á quais elementos podem ser otimizados para obter um incremento no desempenho dos algoritmos. De acordo com a lei de Amdahl, ao acelerar o desempenho de um algoritmo deve-se otimizar primeiro a etapa que consome mais tempo na execução de um algoritmo. A aceleração máxima obtida será limitada pela parte não otimizada.

No processo de busca de bloco, foram identificados os seguintes componentes do algoritmo que podem ser otimizados para o algoritmo de estimação de movimento:

- cálculo das diferenças absolutas
- cálculo dos endereços
- comparação dos resultados com os já armazenados durante o processo de busca

4.2.1.1 Cálculo da soma das diferenças absolutas entre duas regiões

Para a realização do cálculo da SAD, para cada pixel são necessárias as seguintes operações:

- uma subtração entre os valores de intensidade
- calculo do valor absoluto desta diferença
- acumulação ou soma do valor absoluto da diferença

Para a realização do cálculo da diferença absoluta entre dois valores, pode-se utilizar o operador mostrado na Figura 4.2.

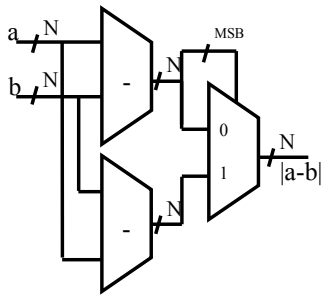


Figura 4.2: Operador para cálculo da diferença absoluta entre dois valores

Diferentes possibilidades para a soma de valores absolutos de toda uma janela podem ser definidas. A mais simples, consiste na serialização desta soma, com a colocação dos dados de cada pixel em seqüência nas entradas do circuito, vindo de uma memória principal e a acumulação destes valores de forma serial em um registrador na saída. Pode-se também dividir a memória em dois bancos de forma a colocar dois valores em paralelo na entrada.

Uma segunda possibilidade é mapear esta iteração temporal numa iteração espacial, através da instanciação de múltiplos operadores. Desta forma, mais de uma diferença absoluta pode ser calculada em paralelo. O resultado destas operações deve ser somado através de uma árvore de somadores, para que não ocorra perda de desempenho. A estrutura resultante pode ser observada na Figura 4.3 (a árvore de somadores está representada simplificadamente). Esta estrutura deve ter uma freqüência de operação menor, pois o caminho de uma entrada até a saída é maior do que no circuito da Figura 4.2, onde não há a árvore de somadores (o que pode ser melhorado através do uso da técnica de pipeline).

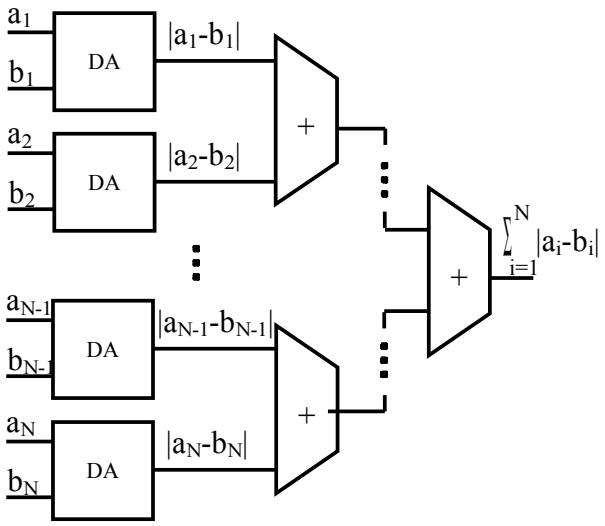


Figura 4.3: Estrutura para o cálculo da diferença absoluta de N elementos

Ainda, neste segundo caso, deve ser colocada uma rede de registradores de forma que os valores dos pixels que são reutilizados no cálculo da SAD de posições consecutivas fiquem armazenados e desta forma o acesso à memória principal será reduzido. Este arranjo pode ser observado na Figura 4.4. Duas matrizes de registradores são necessárias, uma para o bloco da imagem atual e outra para o bloco da imagem de referência. A combinação destes bancos de registradores com o *datapath* da Figura 4.3 permite que o cálculo da SAD seja realizado 8 vezes mais rápido (considerando o númerode ciclos) do que o cálculo que utiliza acesso de 1 em 1 pixel à memória principal. Neste arranjo, os dados são inseridos nos elementos das bordas do conjunto. A interconexão entre os elementos permite que seja realizada uma transferência do valor de cada registrador para seu vizinho de forma paralela.

Um sinal de controle (não representado na figura) comanda a transferência. Uma carga inicial de toda a matriz é necessária no início da operação com a matriz, antes de obter a primeira diferença. Após esta carga inicial, para cada deslocamento de 1 unidade do bloco de comparação, somente os elementos da borda devem ser carregados. Desta forma a comunicação com a memória principal é reduzida por quatro no exemplo.

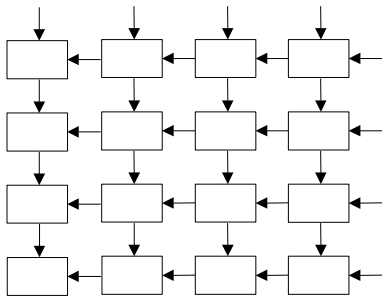


Figura 4.4: Matriz de registradores para armazenamento de uma janela da imagem

O arranjo apresentado na Figura 4.4 permite somente o deslocamento do bloco em duas direções. De forma a permitir o deslocamento em todas as direções pode-se modificar o fluxo dos dados conforme apresentado na Figura 4.5. Esta matriz tem um custo mais alto em hardware devido ao maior número de interconexões, mas reduz os acessos à

memória principal, pois somente uma carga é necessária para realizar toda uma varredura de área.

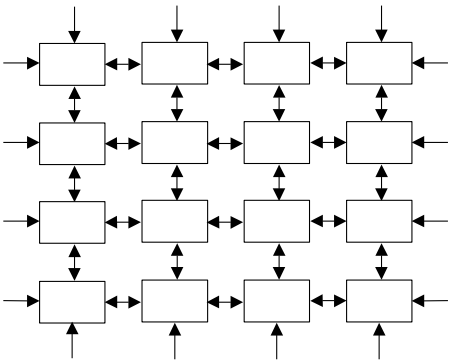
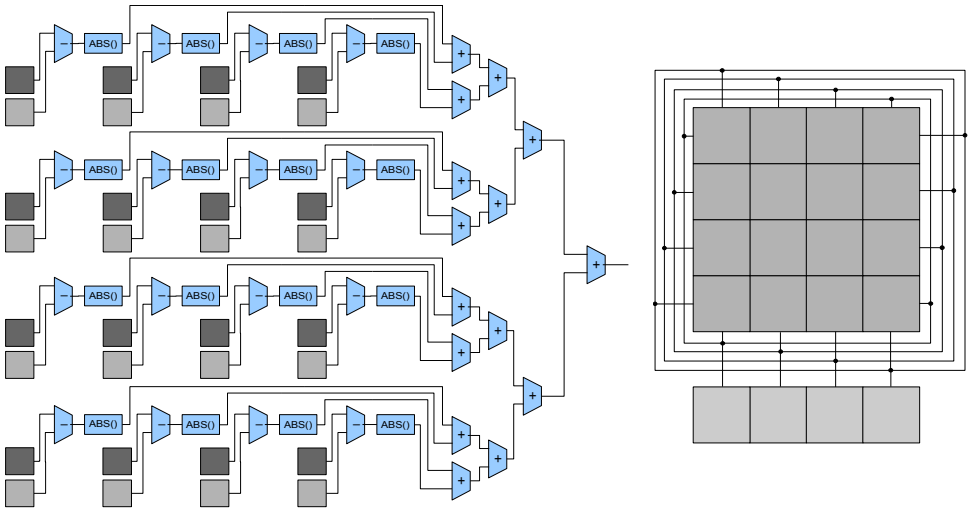


Figura 4.5: Matriz de registradores adaptada para movimento em todas as direções.

A carga do bloco da imagem atual é realizado apenas uma vez para uma busca inteira na imagem de referência. Logo, não é necessário realizar esta carga em paralelo com a carga do bloco da imagem de referência, pois o tempo de busca é muito maior do que o tempo de carga. Na Figura 4.6 está representado o circuito final do cálculo da SAD entre dois blocos, de cada P.E.

Este bloco do hardware será utilizado tanto para as operações de estimação de movimento quanto para a operação de reconstrução estéreo. Em ambos os casos, o valor da SAD é comparado durante a varredura e se for menor do que o valor prévio, o deslocamento relativo é armazenado. Assim se tem no final o deslocamento de menor SAD. Estes valores serão iguais ao vetor de deslocamento no problema de estimação de movimento. Já no caso da reconstrução estéreo, o valor de 9 janelas deve ser calculado e a avaliação do resultado das 9 janelas deve informar a melhor profundidade para o ponto em questão.



(a) (b)

Figura 4.6: (a) Representação final do *datapath* para o cálculo da SAD de um bloco; (b) representação do circuito de carga dos dados na matriz de registradores.

4.2.1.2 Geradores de endereços

Cada elemento de processamento pode contar com um gerador de endereços (DAG - *Data Address Generator*) para realizar o controle da varredura durante as operações de estimação de movimento e reconstrução estéreo. A geração de endereços de uma varredura complexa em um processador através de uma sequência de instruções é um processo que consome uma grande parcela do tempo de execução, o que pode reduzir o ganho obtido através da especialização de operadores. Um gerador de endereços permitirá acelerar o processamento, uma vez que não haverão instruções a serem realizadas para o cálculo destes endereços. Estes geradores também atuarão durante os processos de carga da imagem e cálculo da imagem dos níveis superiores para as imagens multi-resolução.

O bloco completo de cálculo de SAD, composto pela parte operativa, gerador de endereços e comparação dos resultados, foi desenvolvido de forma a receber o endereço do bloco da imagem atual (e que pode ser um endereço da área de blocos recebidos de vizinhos) e as coordenadas do centro da busca. Os limites da busca são verificados e a busca é restringida automaticamente à região do elemento de processamento. O controle dos blocos que devem ser transferidos para regiões vizinhas é realizado em mais alto nível.

O sistema é projetado para operar com memórias de 8 bits de dados e são necessários 4 ciclos para a carga de uma linha ou coluna do banco de registradores do bloco de soma de diferenças absolutas, considerando um bloco de 4x4 pixels. Desta forma, a cada 4 ciclos é realizado o cálculo da SAD de uma posição do bloco da imagem atual sobre a imagem de referência. Quando a varredura está sendo realizada na horizontal são produzidos os endereços à direita ou à esquerda e quando na vertical, acima ou abaixo. A varredura em duas direções aumenta bastante a complexidade do desenvolvimento utilizando memórias de dados de 16 e 32 bits, exigindo o uso de bancos de memória ou memórias de porta dupla. Neste trabalho optou-se por desenvolver um P.E. de complexidade intermediária, isto é, que não são extremamente simples (tais como em chips de visão) para permitir o trabalho com resoluções maiores do que a destes.

O endereçamento a ser utilizado levou em consideração as seguintes restrições:

- deveria permitir o cálculo do endereço a partir das coordenadas do bloco sem utilizar multiplicadores, para não ter impacto significativo na área
- deveria considerar que a imagem deve ser dividida em regiões do mesmo tamanho
- deveria permitir buscas entre fronteiras através do armazenamento de uma linha e coluna dos elementos vizinhos

O calculo de um endereço a partir da posição de um bloco é dado por

Equação 4.1: $$end_{bloco} = end_{base} + l_{região} \cdot pb_y + pb_x$$

onde end_{bloco} é o endereço do bloco, $l_{região}$ é a largura em pixels da região e pb_x e pb_y são os componentes x e y da posição do bloco.

o que requer o uso de um multiplicador. O custo desta operação pode ser reduzido com a utilização de um valor de largura igual a uma potência de dois, de forma que o produto seja calculado com apenas um deslocamento.

Um fator envolvido no aumento da complexidade do cálculo deste endereço é a necessidade de armazenamento de uma coluna e uma linha adicionais. Esta coluna e linha podem ser armazenados das seguintes maneiras:

a)em outra região da memória, o que complica a geração de endereços durante a varredura, devido à alternância de endereços

b)fazendo-se a largura da imagem mais a coluna adicional igual a uma potência de dois, o que facilita o cálculo do endereçamento do bloco durante a varredura, mas impede que a imagem final tenha um tamanho padrão (que normalmente é igual a uma potência de dois ou soma de potências de dois;

c)fazer a largura maior do que uma potência de dois, o que aumenta a complexidade do cálculo do endereço do bloco.

Optou-se pela terceira alternativa, mantendo compatibilidade com os tamanhos padrão de imagens, e mantendo a geração de endereços da varredura. O cálculo fica então igual a

Equação 4.2: $end_{bloco} = end_{base} + (l_{rp21} + l_{rp22}) \cdot pb_y + pb_x$,

onde l_{rp21} e l_{rp22} são componentes da largura, escolhida como uma soma de potências de 2.

Assumindo-se que os componentes da largura da região sejam potências de dois, a operação de multiplicação é substituída por uma operação de soma. A largura de uma coluna é igual a 4 (largura do bloco) e que o segundo termo é igual a $2^{NREGBITS}$, onde NREGBITS é o número da largura da região, tem-se:

Equação 4.3: $end_{bloco} = pb_y * 4 * 2^{NREGBITS} + 4 + pb_x * 4$

As multiplicações por 4 são implementadas através de deslocamentos de 2 bits.

Um módulo de controle fica encarregado de fornecer as coordenadas do centro da busca para o bloco de soma de diferenças absolutas e realizar outras operações como comparação entre diferenças absolutas e armazenamento dos vetores de movimento na memória.

4.2.1.3 Circuito de controle

O circuito de controle do elemento de processamento tem a função de receber a operação a ser realizada, disparar as seqüências de sinais que controlam a carga dos dados durante as varreduras nas operações, e disparar o funcionamento dos geradores de endereços.

Na Figura 4.7 pode-se observar uma representação esquemática da parte de cálculo de SAD elemento de processamento com os blocos agrupados.

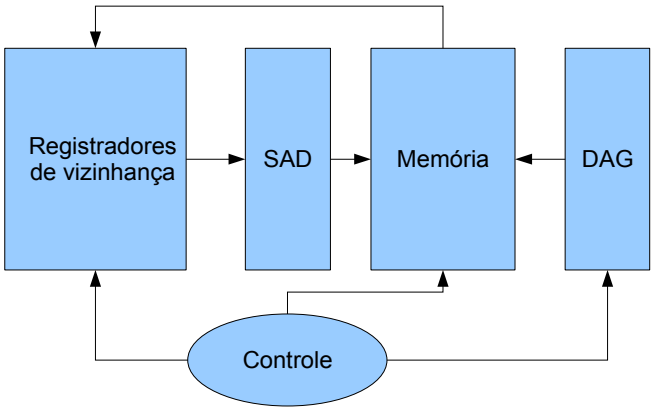


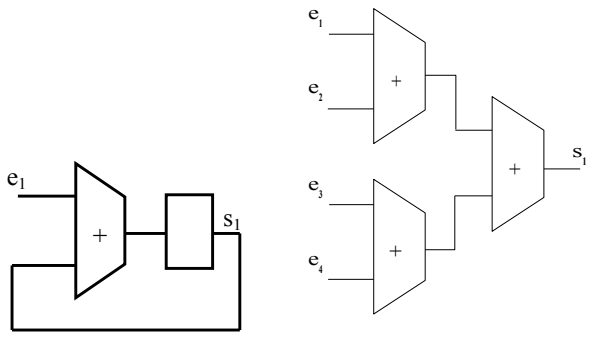
Figura 4.7: Representação esquemática da parte de SAD do elemento de processamento.

4.2.1.4 Geração de Imagens em Múltiplas Resoluções

Se os planos com diferentes resoluções forem processados em regiões diferentes da rede, é necessário gerar estas imagens. Uma possível solução é gerá-las nos elementos de processamento do nível que recebeu a primeira imagem e transmitir a imagem em baixa resolução destes elementos para os elementos do nível seguinte e assim sucessivamente. Isto reduz a necessidade de tráfego da imagem até estes níveis e também a necessidade de armazenar a imagem para após realizar a redução de resolução.

Determinou-se que a imagem em um nível tem a metade da resolução em cada uma das direções, x e y, da imagem do nível imediatamente inferior. Assim, o número de pixels é igual a um quarto do nível imediatamente inferior.

Um circuito simples que pode ser utilizado para o cálculo do valor do plano superior é um acumulador, como mostrado na Figura 4.8(a). Neste circuito, os valores dos pixels são carregados no circuito através da entrada e_1 e a saída é obtida em s_1. Convém ressaltar que a entrada e_1 tem seu número de bits aumentado de 8 para 10, com os dois mais significativos iguais a zero, e a saída s_1 também tem 10 bits, de forma a comportar a soma dos quatro valores. Contudo, somente os 8 mais significativos serão utilizados como saída, que corresponde a dividir por 4 o valor da soma. Se os pixels estivessem disponíveis em grupos de 4 (32 bits) o circuito da Figura 4.8(b) poderia ser utilizado.



(a) (b)

Figura 4.8: Circuitos para cálculo da média de 4 pixels. (a)sequencial; (b)combinacional.

4.2.2 Alternativa 2: Processador de propósito geral

Um processador de propósito geral será utilizado de forma a substituir a arquitetura dedicada e aumentar a flexibilidade para a execução de diferentes algoritmos.

4.2.3 Alternativa 3: Processador de propósito geral+instruções multimídia

Nesta alternativa o mesmo processador de propósito geral será aperfeiçoado para a execução de instruções com uma maior exploração do paralelismo dentro do elemento de processamento, que será utilizado através de instruções especiais (paralelismo explícito).

Optou-se por desenvolver também um elemento de processamento a partir de uma versão modificada de um processador RISC de forma a verificar o quanto se poderia obter de ganho no desempenho em comparação com uma versão dedicada do elemento de processamento, de forma a ter uma maior flexibilidade na utilização da plataforma.

Considerando os itens identificados como maiores consumidores dos recursos computacionais, foram realizadas modificações no *datapath* de um processador compatível com o MIPS, pois este apresentou-se disponível de forma aberta como um *soft core,* já possui um conjunto de ferramentas para desenvolvimento e a sua área é da ordem de grandeza da do elemento de processamento dedicado desenvolvido.

Para acelerar o cálculo da SAD, é necessário o armazenamento de 16 bytes da imagem atual (bloco que será utilizado para a busca) e 16 bytes da imagem de referência (região utilizada para uma comparação) em registradores, para serem utilizados no cálculo sem que seja necessário realizar novamente a carga em cada operação, pois os dados são reutilizados em até 4 posições consecutivas. Uma vez que a imagem pode ser carregada de 32 em 32 bits, é adequado considerar a carga desta imagem em registradores e utilizar instruções de deslocamento para alinhar os dados nos registradores para a posição desejada. Foi feita então uma modificação na unidade de deslocamento (*shifter*) para a realização do alinhamento do bloco para posições não múltiplas de 4.

Então, 4 registradores são carregados com o bloco atual, 4 com o bloco adjacente, (4 ciclos a cada 4 posições na horizontal), e estes são utilizados para uma operação de deslocamento do bloco da imagem de referência, que escolhe 1 byte e substitui na posição à esquerda ou à direita do registrador, conforme mostrado na Figura 4.9. Se o registrador A contiver o valor de uma linha do bloco usado na comparação, por exemplo, diferentes deslocamentos para a esquerda, usando sempre A como entrada e B4, B3, B2 e B1 em cada chamada sucessiva da função de deslocamento equivale a carregar A no endereço base, base+1, base+2 e base+3, permitindo uma emulação da carga desalinhada. Assim, em 5 ciclos, realiza-se o equivalente à carga de um bloco em uma posição não alinhada (distribuindo-se o custo da carga do bloco adjacente em 4 comparações de bloco). O custo médio de uma operação de SAD é então de 9 ciclos.

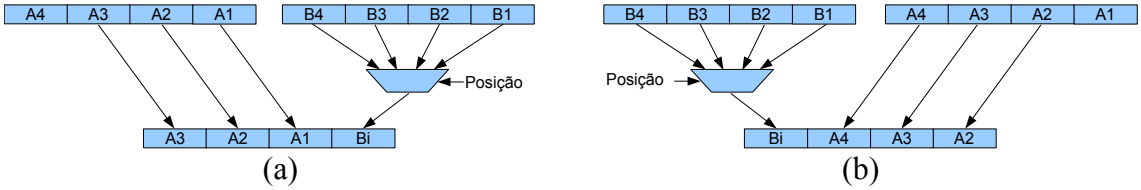


Figura 4.9: Instrução de deslocamento para alinhamento de dados; (a) deslocamento para a esquerda; (b) deslocamento para a direita.

Após armazenados os dados em registradores, uma forma de não alterar significativamente o acesso a estes registradores (pois a adição de conexões adicionais será mais dispendiosa) é realizar o cálculo do SAD de 4 em 4 pixels, os quais são armazenados em um mesmo registrador. Desta forma, em 8 ciclos pode-se ter o cálculo da SAD de uma posição do bloco, com modificações na ULA. Estas modificações incluem o cálculo da diferença entre cada um dos bytes correspondentes individualmente, cálculo do valor da diferença absoluta em paralelo e soma das diferenças absolutas. O circuito para este cálculo pode ser visto na Figura 4.10.

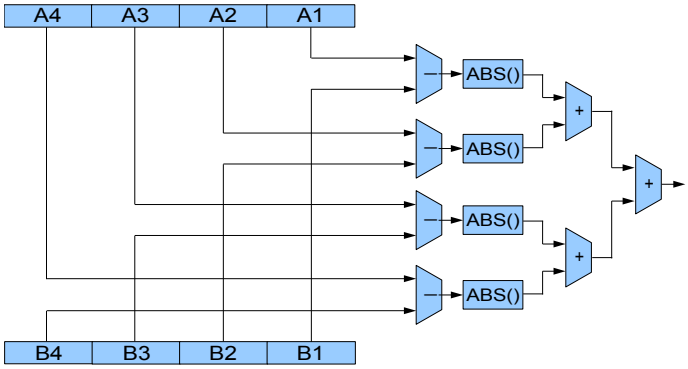


Figura 4.10: Representação esquemática da função adicionada para o cálculo da soma de diferenças absolutas entre 4 pixels. Na figura, podem ser observados os dois valores de 32 bits de entrada e o operador de soma de diferenças absolutas.

Após esta operação, o valor do SAD calculado é comparado com o melhor valor de SAD obtido até o momento, e se for menor, o valor do SAD e a posição atual são armazenados. O custo destas modificações gerou um acréscimo de 2% na área do processador (obtida a partir da comparação da síntese em um FPGA XC2VP2, da Xilinx). O ganho de desempenho obtido com estas modificações, em comparação com a versão inicial da busca foi de 7 vezes, sendo reduzido de 260 para 36 ciclos.

4.2.4 Gerenciador de I/O do Elemento de Processamento

Este circuito tem por função controlar o acesso à memória principal, alternando entre o elemento de processamento, durante as operações, e roteador nas operações de carga e descarga de dados e transferência de dados entre elementos vizinhos. Deve ser provido de funções básicas para interface com a rede, através da implementação do seu protocolo e também de capacidade de transferência de dados para diferentes regiões da memória. Um gerador de endereços neste elemento deve ser capaz de realizar varreduras conforme o tipo de transferência a ser realizada. Cada P.E. possui um gerenciador de IO

especializado para a comunicação de dados associados à imagens, que é capaz de se comunicar com a rede e realizar transferências de ROI (regiões de interesse) de imagens, de blocos de dados e de realizar uma varredura em blocos alternando com dados, através do encapsulamento dos dados da memória em pacotes. Dentre estas, as mais importantes são de dados e de regiões da imagem para a classe de problemas em questão. O desenvolvimento de um circuito dedicado permite que os dados sejam transferidos da memória para a rede e vice-versa sem atrasos no cálculo dos endereços,e portanto em uma velocidade compatível com a da rede (1 transferência por ciclo).

A geração de endereços para a transferência de dados baseia-se na utilização de um ponteiro base mais um deslocamento, onde o deslocamento é armazenado em um contador, para controle do tamanho do bloco a ser transferido e cuja contagem é acrescida a um endereço inicial.

Considera-se aqui que a imagem é armazenada na memória em linhas. Para a transferência de uma ROI (*Region of Interest*), que é uma janela retangular de dados na imagem, foi elaborado um método utilizando dois contadores e dois dados, que são a largura da ROI e a diferença desta largura para a largura da imagem (definido como largura ROIC – *Region of Interest Complement*), conforme pode ser observado na Figura 4.11. O endereço inicial informado para a transferência é o do canto esquerdo superior da ROI, e os dados são transferidos em seqüência até que a contagem do primeiro contador seja igual à largura da ROI, quando então o endereço é incrementado não de 1 mas de um valor igual a ROIC, passando desta forma para a linha seguinte da ROI. A cada incremento no endereço, o segundo contador também é incrementado e seu valor comparado com a área da ROI. Quando ambos forem iguais, a transferência é encerrada. Este mecanismo acabou por substituir o primeiro para a transferência de dados, pois utilizando-se a largura ROIC=0 tem-se uma transferência de dados de uma área contínua na memória.

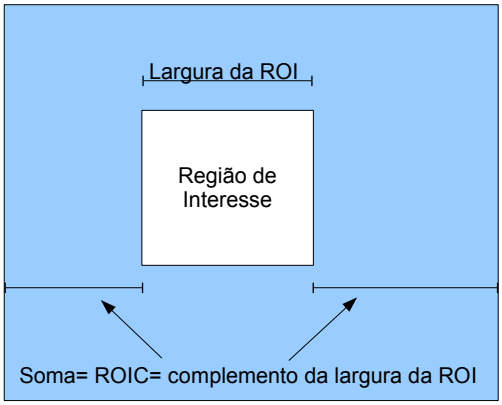


Figura 4.11: Definições utilizadas para transferências pelo gerenciador de I/O na rede.

O tratamento da busca em regiões vizinhas requer a transferência de blocos entre regiões, o que requer uma varredura mais complexa, pois os blocos não seguem um endereçamento contínuo na memória, já que a imagem está armazenada em linhas. A transferência de blocos também deve ocorrer em uma linha ou coluna de blocos da imagem, e fornecer para as regiões vizinhas alguns dados associados aos blocos, como por exemplo, o centro da região de busca.

Para a transferência da imagem em blocos 4x4 e mais 4 bytes de dados por bloco (com as coordenadas do centro da busca), é necessário utilizar um contador para os dados do bloco e um para os dados adicionais a serem transferidos. Quando a contagem do primeiro contador chega a 4 (2 bits menos significativos iguais a 0), o endereço é incrementado do tamanho de uma linha menos 4, de forma que a varredura passe para a linha seguinte. Quando o tamanho do bloco é completado (4 bits menos significativos iguais a 0), o endereço é armazenado e alternado para a área de transferência de dados e o contador de dados é acrescido ao endereço de base para que os dados sejam colocados nesta área. Este processo de varredura está representado na Figura 4.12. Quando a contagem de dados chega a 4, o endereço é alternado para a área de blocos, e o processo reinicia. Um código utilizado na área de dados informa quando o pacote de blocos chegou ao fim, e este código é utilizado para permitir que seja processado um número variável de blocos pelos P.E.s vizinho, identificando o seu final. Para otimizar as transferências, os dados são transferidos antes do bloco, de forma que após o último bloco venha o código de encerramento do processamento. Este código, que é armazenado na área de dados do receptor, não tem relação com o código de encerramento do pacote. Com o uso deste código o receptor não necessita contar os blocos recebidos (os quais são colocados na área de dados recebidos), e o posterior processamento destes blocos é interrompido quando este código é encontrado na área de dados.

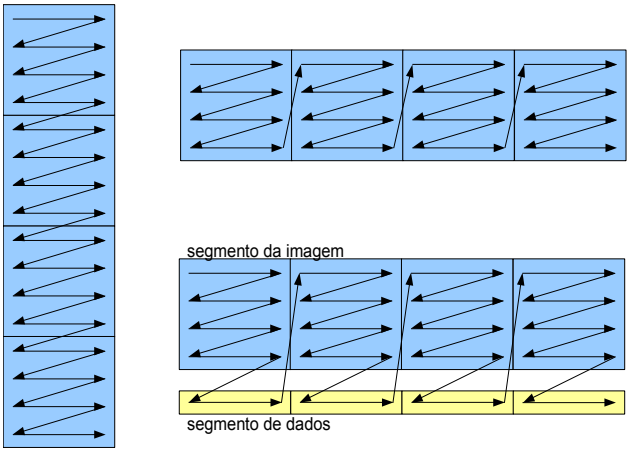


Figura 4.12: Representação da varredura da memória em coluna de blocos, em linha de blocos e alternando linha de blocos com linha de dados.

Optou-se por não receber dados nos P.E.s durante a realização da busca. Um motivo forte para esta decisão é que a maior parte da memória está ocupada com as imagens envolvidas na busca (atual e de referência) e os dados do seu processamento. Assim, não há por que receber outros dados além de regiões vizinhas, que são armazenadas em uma área reservada da memória. Além disto, a recepção de dados (os quais são colocados na memória) diminui o desempenho da operação de busca, na qual a memória é acessada pelo processador em todos os ciclos. Assim, as comunicações entre a rede e o P.E. são interrompidas nesta etapa.

Os blocos recebidos são associados a cada um dos vizinhos (vizinho superior, esquerdo, etc) de acordo com um identificador enviado junto com o pacote. Conforme o P.E. que enviou o pacote, este é encaminhado para uma área específica na memória dos

P.E. que o recebe. Isto é uma forma de permitir a devolução do resultado para o P.E. de origem. Uma representação esquemática do gerenciador de I/O do P.E. pode ser observada na Figura 4.13.

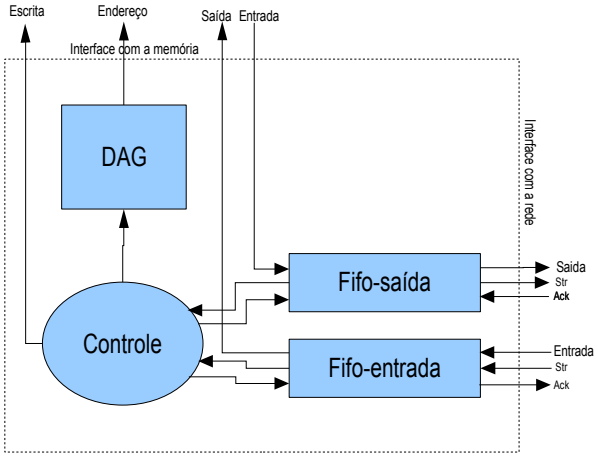


Figura 4.13: Representação esquemática do gerenciador de I/O do elemento de processamento

4.2.4.1 Realização da Carga e Descarga da Imagem

A carga da imagem será realizada através de mensagens do ponto de entrada da rede até os elementos de processamento. Não é adequada a realização de um *broadcast*, uma vez que os elementos de processamento que não necessitassem de uma determinada parte da imagem continuariam em espera e a imagem geraria um tráfego desnecessário pela rede, o que tem como conseqüência um maior consumo de potência.

4.2.5 Diagrama Completo do Elemento de Processamento

Na Figura 4.14 pode-se observar o diagrama final do elemento de processamento com todas as estruturas já mencionadas. Em (a) pode-se observar os módulos de soma de diferenças absolutas, gerador de bordas, o controle e o sub-amostrador podem ser facilmente agrupados em um único módulo. Em (b) o elemento de processamento dotado de um processador de propósito geral, o qual pode ser acrescido ou não de especializações. Conforme os resultados, é possível ainda adicionar ao elemento de processamento em (b) alguns dos módulos do item (a) para aumento do desempenho.

4.3 Organização da Memória

A memória será distribuída em conjunto com os elementos de processamento na arquitetura de forma não compartilhada. Isso evitará a disputa no acesso por diferentes P.E.s. Em cada P.E., a memória é composta por dois bancos, sendo que cada banco é capaz de armazenar uma região da imagem. Esta memória de cada P.E. deve ter capacidade para armazenar no mínimo duas regiões da imagem (atual e de referência), cujo tamanho deve ser parametrizável. Para a execução do algoritmo de estimação de movimento, um dos bancos armazenará uma região do quadro atual e o outro a região correspondente do quadro de referência, por exemplo. Um percentual adicional é

utilizado para armazenar dados enviados por P.E.s vizinhos, como blocos para ser realizada a busca na região vizinha sobre a imagem de referência.

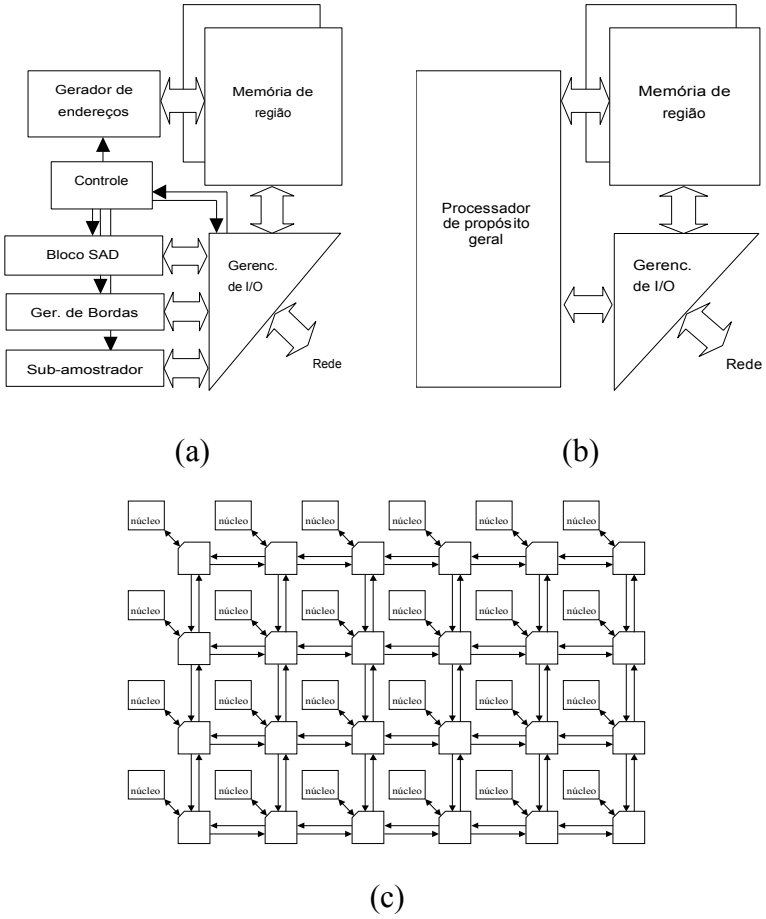


Figura 4.14: Diagrama completo do elemento de processamento;(a) dedicado; (b) processador de propósito geral (com ou sem especializações); (c) representação da arquitetura, onde cada núcleo é um elemento de processamento representado.

As imagens serão tratadas em tons de cinza e serão armazenadas na memória em um formato de 8 bits por pixel, com 1 pixel em cada posição de memória (assumindo uma memória de 8 bits).

4.4 Controlador da Rede (Mestre)

Este elemento é um processador RISC e tem como função gerenciar a entrada e saída de dados da rede, bem como enviar comandos determinando quais as operações devem ser realizadas. Ele é conectado a uma memória externa. Ele deve realizar as operações globais de mais alto nível que forem necessárias.

4.5 Desenvolvimento da arquitetura como um soft-core

O desenvolvimento da plataforma proposta foi feito utilizando-se a linguagem VHDL (*Very High Speed Integrated Circuit Hardware Description Language* (D'AMORE, 2005)). O uso de uma linguagem de descrição de hardware permite

customizar a plataforma conforme a aplicação para diferentes dimensões de imagem e com um número arbitrário de elementos de processamento por imagem. Buscou-se desenvolver componentes parametrizáveis para aumentar sua flexibilidade e, portanto, o seu reuso. Após a parametrização, pode-se utilizar um FPGA para implementar a arquitetura final ou utilizar como entrada em uma ferramenta para geração do leiaute e posterior fabricação.

A rede intra-chip utilizada, SoCIN (*System on Chip Interconection Network*), é um componente parametrizável, permitindo interconectar um número pré-determinado de núcleos, e que devem obedecer a um protocolo de comunicação pré-determinado (Zeferino , 2003).

Em cada roteador da rede foi conectado o conjunto de unidades funcionais, as quais formam em conjunto com o roteador uma unidade denominada de *tile,* que é um elemento de processamento (P.E.) + roteador no conjunto. Os elementos de processamento foram desenvolvidos de forma a considerar a parametrização da rede e manter seu princípio de funcionamento de acordo com os parâmetros fornecidos.

Dentre os parâmetros informados, estão:

- dimensões da imagem, em pixels;
- dimensões da rede, horizontal e vertical, em número de elementos de processamento;
- tamanho das memórias dos elementos de processamento.

A cada P.E. é associado automaticamente um identificador que informa sua a posição sobre a imagem. O P.E. é classificado automaticamente como:

- canto superior esquerdo (CSE);
- canto superior direito (CSD);
- canto inferior esquerdo (CIE);
- canto inferior direito (CID);
- borda superior (BS);
- borda esquerda (BE);
- borda direita (BD);
- borda inferior (BI);
- central (C);
- mestre (M, fora da imagem);
- outro (O, fora da imagem).

Este identificador permite que se controle o tipo de processamento a ser realizado e o padrão de comunicação a ser seguido, no caso de execução do algoritmo particionado por domínio. Na Figura 4.15, pode-se observar uma possível configuração da rede onde 16 elementos de processamento são responsáveis pelo processamento da imagem (escravos) e um dos elementos é o mestre.

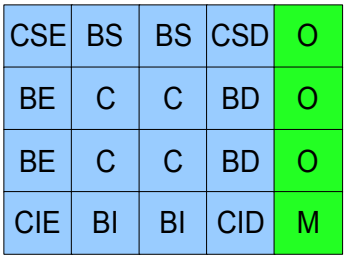


Figura 4.15: Representação da classificação dos elementos de processamento na rede, conforme a posição sobre a imagem.

4.6 Controlador de I/O

São elementos cuja finalidade é receber dados do mundo externo e injetar na rede, durante a execução de uma aplicação.

4.7 Comparação de área entre as diferentes implementações

Na Tabela 5.1 pode-se observar a área ocupada pelas diferentes implementações, para os elementos da rede, obtida a partir da síntese em um FPGA XC2VP2, para uma comparação relativa. Não está sendo considerada aqui a memória das diferentes arquiteturas. A memória de dados terá aproximadamente o mesmo tamanho tanto para o MIPS quanto para o ASIC. Contudo, o MIPS deverá contar com uma memória de programa, a qual não é necessária no ASIC. O tamanho desta memória será proporcional ao tamanho das tarefas que se queira executar na arquitetura. Para o trabalho descrito em (Ngounga 2006), o tamanho utilizado foi 16kb por P.E..

Tabela 4.1: Área ocupada pelos diferentes elementos de processamento, obtida a partir da síntese em um FPGA XC2VP2 da Xilinx.

Elemento de processamento	*LUTs de 4 entradas*	*Slices (LUT+FF+logica de carry)*	*FFs*
Processador MIPS	2253	1240	420
ASIC	2873	1700	673
Processador MIPS + instruções multimídia	2329	1271	421

4.8 Conclusões

Com relação às modificações introduzidas no processador de propósito geral, pode-se concluir que a adição das instruções multimídia representou um pequeno aumento na área do processador, uma vez que a instrução de deslocamento consiste apenas na

introdução de multiplexadores e a instrução de cálculo de diferenças absolutas reutiliza parte do *datapath* já disponível na ULA, e novos elementos de memória não foram introduzidos.

Com relação aos elementos de processamento, a área do ASIC ficou em uma faixa comparável a área do processador de propósito geral, devido à implementação em paralelo dos elementos citados no seu desenvolvimento.

Uma solução intermediária, mas que pode trazer resultados é a combinação das unidades funcionais do ASIC com o processador de propósito geral, de forma a reduzir o tempo de processamento do mesmo.

No próximo capítulo são realizados experimentos com os P.E.s desenvolvidos e seu desempenho é avaliado.

5 EXPERIMENTOS

5.1 Utilização da arquitetura com algoritmos particionados por função

Nestes experimentos, utilizou-se a arquitetura composta por processadores especializados e rede intra-chip. A utilização unicamente do ASIC como elemento de processamento permitiria somente a realização da maioria das etapas de processamento que utilizam a operação de *block matching.*

5.1.1 *Experimentos com Macro-Pipelines em Nocs*

No caso de projeto de um sistema com macropipelines, pode-se utilizar conexões ponto a ponto para ligar os elementos de processamento. Para um sistema onde serão executadas múltiplas aplicações, as quais serão modificadas com o passar do tempo, e cujo comprimento (número de funções) pode ser modificado, deve-se buscar uma alternativa mais flexível.

Foram realizados diferentes experimentos com o mapeamento de aplicações para NoCs com um conjunto de elementos de processamento homogêneo, onde cada elemento de processamento é um processador RISC. Para este experimento, o código de um codificador JPEG foi quebrado em quatro tarefas e adaptado para rodar as diferentes funções em seqüência sobre os blocos. Uma representação deste fluxo pode ser vista na Figura 5.1.

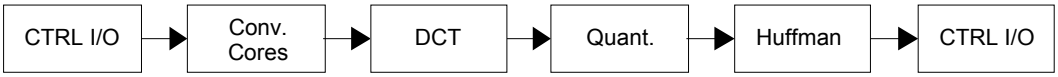


Figura 5.1: Fluxo para processamento de um bloco no algoritmo JPEG. O fluxo inicia e termina em um Controlador de I/O.

Foi desenvolvido um software para possibilitar a entrada de grafos de aplicações (que devem ser particionadas em tarefas) e realizar o mapeamento sobre uma NoC considerando uma função de custo, como o congestionamento na rede ou o comprimento do caminho dos dados sobre a rede (Figura 5.2). A NoC permite que os nodos de início e fim do macropipeline não estejam necessariamente ligados diretamente a um nodo de entrada e saída da rede. Nesta situação os nodos intermediários necessitariam parar o seu processamento para realizar a transmissão dos dados caso fosse utilizada uma plataforma composta por uma matriz de processadores e

comunicações ponto a ponto. Como plataforma foi considerada uma rede com dimensões de 4x4. O processador de propósito geral escolhido foi o Plasma (disponível em www.opencores.org), compatível com o MIPS. Cada nodo da rede possui uma memória de 16 kbytes.

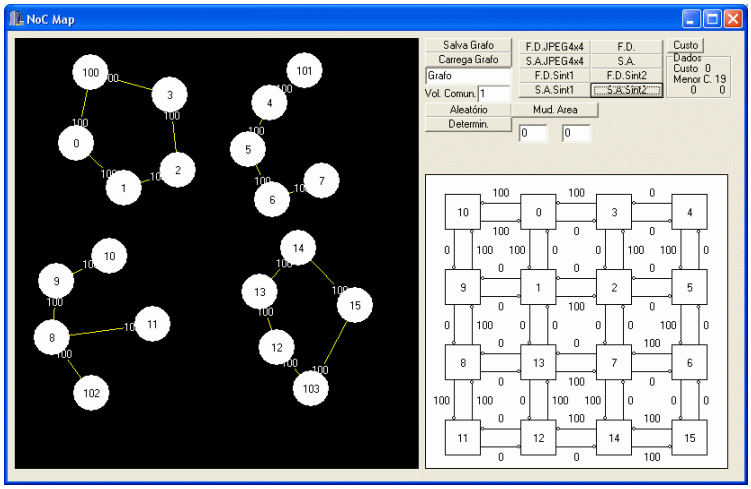


Figura 5.2: Interface do software desenvolvido para testes com posicionamento de tarefas na rede.

Nesta plataforma foram adicionados 4 elementos nos 4 vértices da rede, que são responsáveis pela entrada e saída de dados (I/O). No software onde é realizado o mapeamento, estes elementos estão identificados por nodos com numeração escolhida arbitrariamente como sendo igual a 100 adicionado ao número do nodo de entrada (i.e. 100, 101, 102, 103 para os nodos de I/O 0, 1, 2, 3). A estrutura da rede pode ser observada na Figura 5.3. As tarefas foram modeladas através de grafos, considerando-se o tempo de computação em cada nodo e o de comunicação. O resultado conforme esperado, foi que as tarefas apresentam tendência a serem posicionadas com o início da cadeia perto do elemento de entrada e com o final da cadeia perto do elemento de saída.

Foram testados os algoritmos *simulated annealing* e *force direct* para a realização do posicionamento, e mais detalhes podem ser encontrados em (NGOUNGA 2006). A idéia principal é a adaptação da rede a novas tarefas que sejam colocadas no sistema. Assim, um dos P.E.s executa o algoritmo de posicionamento conforme novas aplicações são introduzidas no sistema. O algoritmo *simulated annealing* teve o número de iterações limitado de forma a obter um posicionamento adequado em um tempo aceitável. O algoritmo é projetado para ser executado antes do posicionamento das tarefas da aplicação, pois supõe-se que o MPSoC possa executar diferentes tarefas e a configuração das mesmas pode ser diferente no momento da adição de uma nova tarefa. Também se supõe que as tarefas serão executadas durante um tempo bem maior do que o tempo necessário para o cálculo do posicionamento, de forma que o ganho obtido por este posicionamento seja justificado.

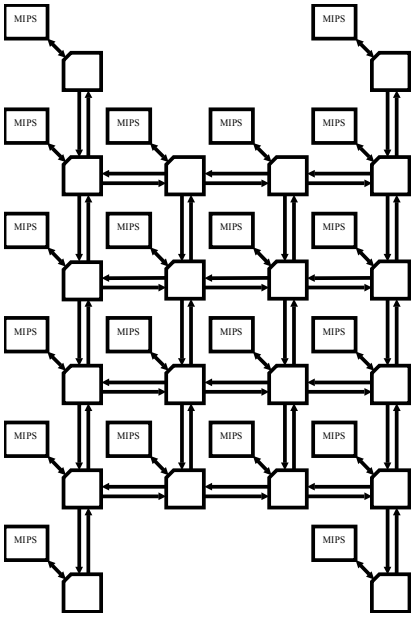


Figura 5.3: Diagrama de blocos da NoC utilizada para os testes

O tempo de execução destes algoritmos pode ser observado na Tabela 5.2. O comprimento do caminho das mensagens na NoC após o posicionamento automático pode ser visto na Tabela 5.3, e o ganho em tempo de processamento na Tabela 5.4. Observa-se que mesmo quando as aplicações têm um grafo de comunicação simples, como um macro-pipeline, o posicionamento das tarefas na rede ainda deve ser realizado de maneira inteligente, pois um ganho, em termos de desempenho, pode ser obtido. Outro detalhe a ser destacado é o comportamento do posicionamento com relação aos elementos responsáveis pelo I/O. Na Tabela 5.4 foram considerados dois casos para o tratamento da entrada e saída dos dados. No primeiro um mesmo gerenciador de I/O é responsável pela entrega e coleta dos dados para cada canal de decodificação, e está indicado na tabela por 1NC.

Tabela 5.1: Tempos de execução de cada tarefa do algoritmo JPEG para um bloco e comparação entre diferentes implementações considerando *pipeline*

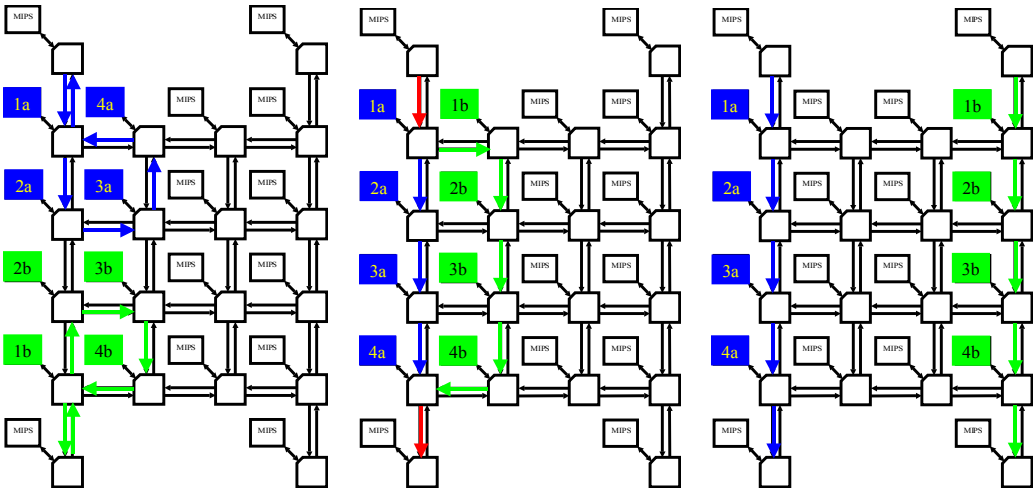
	1 CPU	*2 CPUs*		*4 CPUs*			
		cpu1	cpu2	cpu1	cpu2	cpu3	cpu4
RGB2YUV							
DCT							
QUANT							
VLC							
Computação (total in ciclos)	33687	17406	16281	5500	5397	6509	16281

No segundo caso, um gerenciador entrega os dados e o segundo coleta os dados. Isto é mostrado na Figura 5.4 (a) e (b). No primeiro caso, se houver um ou quatro canais de

codificação de dados na rede, o desempenho será o mesmo, pois cada canal está ligado ao seu nodo de entrada e saída. Já no segundo caso, se houverem até dois canais na rede, o desempenho não será afetado, mas se forem colocados mais dois canais, haverá um atraso devido ao tráfego dos dados de diferentes macropipelines pelos mesmos nodos de entrada e saída. Já na Figura 5.4 (c) pode-se observar o caso em que há um elemento para a entrada e outro para a saída. Das tabelas 5.3 e 5.5 a conclusão é que somente é vantajoso a execução do posicionamento das tarefas quando o tempo de execução do posicionamento apresentado na Tabela 5.3 é menor do que a redução no tempo de execução da tarefa, que é apresentada na Tabela 5.5. Caso contrário, o processo incrementará o tempo total da tarefa, e é mais vantajoso realizar um posicionamento randômico.

Tabela 5.2: Tempo de execução para os algoritmos de posicionamento (ciclos)

Algoritmo, tamanho do grafo e tamanho da NoC	Simulated Annealing	Force Directed
JPEG (8 nodos) - 4x4	3771030	270741
Sintética 1 (12 nodos) - 4x4	6840670	470341
Sintética 2 (16 nodos) - 6x6	5716176	499721



(a) (b) (c)

Figura 5.4: Exemplo de 3 possíveis posicionamentos do canal de codificação JPEG sobre a rede, considerando a entrada e saída de dados. (a) sem compartilhamento do nodo de I/O, com um único elemento para entrada e saída; (b) com compartilhamento de I/O, com um elemento para a entrada e outro para saída; (c) sem compartilhamento de I/O, com um elemento para a entrada e outro para saída.

Tabela 5.3: Comprimento do caminho das mensagens na NoC, do nodo de entrada até o nodo de saída, após o posicionamento automático

Algoritmo, tamanho do grafo e tamanho da NoC	Simulated Annealing	Force Directed
JPEG (8 nodos) - 4x4	6	6
Sintética 1 (12 nodos) - 4x4	21	23
Sintética 2 (16 nodos) - 6x6	19	18

Tabela 5.4: Comparação entre o tempo de execução da aplicação considerando um posicionamento aleatório das tarefas na rede e um posicionamento otimizado.

Algoritmo, tamanho do grafo e tamanho da NoC	Tempo de execução (aleatório)	Tempo de execução (otimizado)	speedup
2 x JPEG (8 nodos) – 4x4	40172	36512	9.1%
4 x JPEG (16 nodos+1NC) - 4x4	43633	36536	16.2%
4 x JPEG (16 nodos+2NC) - 4x4	42785	35851	16.2%
Sintética 1 (12 nodos) - 4x4	13186	7745	41%
Sintética 2 (16 nodos) - 6x6	17775	12491	29.7%

5.2 Utilização da plataforma com algoritmos particionados por domínio

Nestes experimentos, utilizou-se a arquitetura composta por processadores especializados e rede intra-chip. A título de comparação, utilizou-se também o ASIC como elemento de processamento para a realização da operação de busca de blocos. O controle do ASIC foi customizado de forma a permitir a realização de toda a operação, desde a distribuição da imagem até a composição da matriz final dos vetores de movimento e controle da comunicação.

5.2.1 Experimentos com o algoritmo de Estimação de Movimento

O particionamento do algoritmo de estimação de movimento utilizando a busca completa já foi apresentado no capitulo 3.

Na etapa seguinte, buscou-se estimativas para o desempenho do sistema com os diferentes elementos de processamento propostos. Também foi realizada uma comparação com processadores comerciais. Como plataforma comercial, foi utilizado para comparação um DSP TMS320C64XX, que é utilizado para decodificação de vídeo. Considerou-se o desempenho de cada uma das arquiteturas para a execução de uma operação de comparação de bloco, pois é a que apresenta maior demanda computacional no problema. Também foram incluídas na comparação a arquitetura ASIC desenvolvida e uma versão modificada do processador MIPS para a execução de instruções multimídia que têm relevância para o problema em questão.

A versão ASIC também foi utilizada como elemento de processamento na plataforma composta pela matriz de elementos de processamento interligados pela rede

intra-chip. A partir da simulação desta plataforma, pode-se também obter uma estimativa do desempenho para a mesma rede, com diferentes arquiteturas para o elemento de processamento, mantendo o mesmo padrão de comunicação.

Devido à dificuldade de acesso aos processadores e ambiente de simulação, não foi possível realizar uma comparação com os processadores CELL e com o DSP da Cradle. Entretanto, considerando o número de instruções por segundo é possível realizar uma comparação indireta.

A comparação indireta com chips de visão não foi realizada pois os mesmos tem resoluções reduzidas e em geral, realizam buscas em uma distância pequena.

O algoritmo foi escrito em linguagem C e compilado utilizando os seguintes ambientes:

- Code Composer Studio no DSP TMS320C64XX
- Compilador GCC para o processador MIPS (PLASMA)
- Compilador GCC e Assembler AS (para as instruções adicionais) na plataforma MIPS MMX.

Na Tabela 5.6 pode-se observar a performance de cada um dos elementos testados para a realização da operação de *block matching* em uma posição de um bloco de 4x4 pixels. Uma comparação justa dos resultados é baseada no tempo de execução dos algoritmos em cada uma das plataformas. Contudo, uma vez que não forma todas desenvolvidas empregando a mesma tecnologia, o tempo pode não ser a grandeza mais adequada. Da mesma forma, mesmo utilizando uma mesma tecnologia, diferentes arquiteturas podem não ter a mesma frequência de operação, uma vez que esta freqüência está associada ao maior atraso entre dois pontos em um sistema síncrono, embora se possa contornar esta condição através de técnicas apropriadas, como por exemplo o uso de pipeline, o reposicionamento de blocos ou inserção de *buffers*. Por estes motivos, os resultados neste trabalho serão apresentados considerando que o MIPS ou o ASIC são projetados na mesma tecnologia que o DSP, e considerando uma mesma freqüência de operação de 1GHz. Esta é a freqüência de operação do DSP escolhido, e que é comum ou mesmo superada pelos processadores atuais do estado da arte, os quais estão, hoje, em um limite próximo a 4GHz.

Tabela 5.5: Performance de cada elemento para a execução da operação de *block matching* (bloco de dimensões 4x4) em uma posição

Processador	***Ciclos***	***Block Matching por segundo (1GHz)***
TMS320C6400	335	2,9M
MIPS	260	3,8M
MIPS MMX	36	27M
ASIC	4	250M

Observa-se que a grande diferença no desempenho do ASIC só é possível devido à utilização do processamento paralelo dentro do elemento de processamento. Enquanto que nas três primeiras arquiteturas o processamento ainda ocorre de maneira serial (a operação de SAD não é realizada em paralelo), na quarta arquitetura (MIPS MMX) esta operação foi implementada no hardware do processador. Ainda nesta quarta arquitetura, o acesso à memória foi acelerado devido à operação de deslocamento adicionada, a qual permite carregar os 4 bytes em paralelo de acordo com o endereço fornecido (carga de palavra desalinhada), o que lhe conferiu um ganho de desempenho total de cerca de sete vezes o desempenho original. Já na arquitetura ASIC, esta carga ocorre em 4 ciclos, na direção do movimento, sem a recarga da matriz.

Um segundo fator que é responsável pelo decréscimo no tempo de execução é a realização do cálculo do endereço de carga dos dados em conjunto com a comparação do valor atual da SAD com o valor da melhor SAD e subseqüente armazenamento. A combinação deste fator em conjunto com o *datapath* de cálculo de SAD conferiu ao ASIC um desempenho 8 vezes maior do que o desempenho do processador MIPS com instruções especializadas.

Após determinar qual o desempenho de cada P.E. para a operação de busca de bloco em uma posição, o próximo passo foi calcular a demanda computacional do algoritmo e o tempo de carga da imagem, para obter uma estimativa da freqüência de operação do sistema.

O volume de dados,para uma imagem de 640x480 é de:

resolução 640x480: 307200 bytes.

O tempo de carga da imagem considerando-se uma resolução de 640x480, para uma transferência pela rede com o protocolo baseado em créditos, onde um byte é transferido por ciclo, é de 307200 ciclos.

Uma imagem de 640x480 pixels possui 640x480/16=19200 blocos de tamanho 4x4. Se considerarmos uma distância d=64, utilizando a Equação 3.9 chegamos aos resultados mostrados na Tabela 5.7 para a estimativa de processamento, utilizando o algoritmo de busca completa. Como se pode observar, não é possível atingir tais freqüências de operação nem com a tecnologia atual e nem em um futuro próximo, mesmo considerando a arquitetura desenvolvida com ASIC. Na Tabela 5.8 estão estimativas para a freqüência de operação na plataforma desenvolvida, com um diferente número de elementos de processamento. Nesta tabela não está sendo considerado o tempo de transferência de blocos entre os elementos de processamento. Uma vez que este tempo é menor do que o tempo de transferência de uma imagem, a sua introdução será irrelevante para o resultado, dado o tempo necessário para a busca.

Da observação da tabela pode-se concluir que somente com a utilização do paralelismo em diferentes níveis é possível atingir um desempenho que permita construir um sistema que opere dentro de freqüências factíveis. Para este algoritmo, se é desejado manter a flexibilidade de reuso da plataforma, deve-se adicionar ao processador de propósito geral a unidade de geração de endereços e o módulo de comparação de resultados, para otimizar o restante do algoritmo.

Tabela 5.6: Estimativa de frequência de operação considerando cada elemento de processamento

Elemento de Processamento	Número de ciclos/bloco	Número de ciclos/imagem	Imagens por segundo (1GHz)	Freq. de operação para 30 quadros/s
DSP	5574735	107035M	0,009	3211GHz
MIPS	4326660	83072M	0,012	2492GHz
MIPS com instruções multimídia	599076	11502M	0,086	345GHz
ASIC	66564	1278M	0,782	38GHz

Tabela 5.7: Estimativa de desempenho considerando um sistema composto por uma matriz de 64 elementos de processamento

Matriz de 64 Elementos de Processamento	Número de ciclos/bloco	Número de ciclos/imagem	Imagens por segundo (1GHz)	Freq. de operação para 30 quadros/s
DSP	5574735	1672M	0,59	50GHz
MIPS	4326660	1298M	0,77	38GHz
MIPS com instruções multimídia	599076	180M	5,55	5.4GHz
ASICs	66564	20M	50,00	608MHz

Pode-se dividir o tempo total da operação de busca de cada P.E. nas seguintes etapas:

1. tempo de recepção da região da imagem atual;
2. tempo de recepção da região da imagem de referência (somente na primeira imagem, nas seguintes a imagem atual passa a ser a de referência);
3. tempo de processamento dos blocos locais;
4. tempo de transferência de blocos para P.E.s vizinhos;
5. tempo de processamento dos blocos vizinhos;
6. tempo de recepção de vetores de movimento dos vizinhos;
7. tempo de seleção dos vetores de movimento, dos recebidos ou dos vetores calculados localmente,com base no valor de SAD e módulo do vetor;
8. tempo de envio dos vetores de movimento para o mestre.

A comunicação entre os P.E.s também aumenta conforme aumenta o número de regiões. Idealmente, as regiões devem ter dimensões iguais ou maiores do que a distância de busca, para que os pacotes não avancem muito na rede durante as etapas de comunicação entre elementos vizinhos.

5.2.1.1 Comunicação para o tratamento de dependências de dados

Uma vez que neste modelo cada elemento de processamento armazena uma região da imagem atual e uma região correspondente da imagem de referência, é necessário realizar o tratamento das dependências de dados de regiões vizinhas. Através do envio e recebimento de pacotes de dados é que é executada a busca de um bloco em uma parte da área de busca que situa-se em uma região vizinha (ou seja, em um P.E. vizinho).

No desenvolvimento das etapas que envolvem comunicação mostrou-se necessário organizar as transferências, pois o gerenciador de IO realiza apenas o envio ou a recepção de dados, e não os dois simultaneamente, devido ao conflito no acesso à memória. Assim, se todos os P.E.s iniciarem uma transmissão simultaneamente, nenhuma será completada, pois nenhum estará no estado de recepção. A ordem das transferências foi organizada conforme o esquema mostrado na Figura 5.5. Nesta figura é utilizada como exemplo uma matriz de 4x4 P.E.s, os quais estão associados a uma parte correspondente na imagem. Outros elementos do sistema, como por exemplo o mestre, não estão representados aqui.

Na Fig. 5.5(a) está a primeira etapa do envio, onde somente os vizinhos na horizontal e na vertical de um determinado P.E. recebem dados. Na segunda etapa do envio (Fig. 5.5(b)), as funções de envio e recepção são invertidas, paraque os P.E.s que enviaram blocos para serem processados nas regiões vizinhas realizem a busca dos blocos de regiões vizinhas na sua região. Após a segunda etapa, todos os P.E.s realizam a busca dos blocos recebidos. Nesta busca, são armazenados o vetor de movimento da posição de menor diferença absoluta e o valor desta diferença. A seguir, os dados calculados para cada bloco são devolvidos, utilizando o mesmo ordenamento do envio e da recepção. Os vetores recebidos são comparados com os calculados através da realização da busca na região da imagem de referência armazenada no P.E. O vetor com menor valor de SAD é armazenado ou com menor módulo, no caso de valores de SAD iguais.

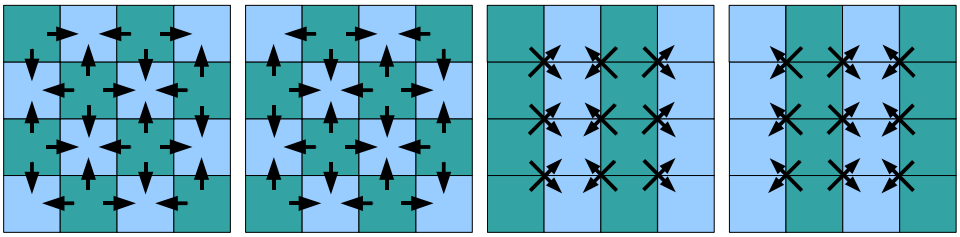


Figura 5.5: Etapas de envio de mensagens por elementos da rede, para uma rede com 16 elementos de processamento.

A seguir, é realizado o envio dos blocos para os elementos de processamento nas diagonais. Aqui a rede é dividida conforme o esquema mostrado na Fig. 5.5(c). Novamente a transferência dos blocos ocorre em duas etapas e após o processamento, a devolução dos dados também ocorre em duas etapas, completando-se na Fig. 5.5(d).Esta ordem foi estabelecida após a observação das comunicações necessárias para cada P.E.

(apresentada no capítulo 3) e verificação dos possíveis conflitos entre P.E.s que devem trocar mensagens, quando ambos tentam enviar uma mensagem simultaneamente ou receber.

Para realização do controle deste processo, cada elemento de processamento recebeu um identificador de 1 a 4, de acordo com a sua posição na rede, sobre a imagem, conforme mostrado na Fig. 5.6. Com este identificador, é possível determinar em cada etapa a função a ser executada pelo P.E. (envio ou recepção). Na Fig. 5.5, os elementos com identificador 1 enviam nas etapas (a) e (c), os elementos com identificador 2 enviam nas etapas (b) e (d), os elementos com identificador 3 nas etapas (b) e (c) e os elementos com identificador 4 nas etapas (a) e (d).

1	2	1	2
3	4	3	4
1	2	1	2
3	4	3	4

Figura 5.6: Divisão da rede para envio de dados

A ordem de envio dos pacotes por um determinado P.E. se dá arbitrariamente no sentido horário, conforme mostrado na Fig.5.7 e na Fig.5.8. Nas figuras, foi utilizada a mesma matriz de 4x4 P.E.s, a qual apresenta as 9 possíveis posições de um elemento de processamento dentro da imagem (i.e. central, borda direita, etc.). Na figura, cada etapa mostra o estado de um elemento de processamento durante o envio (início, envio para o elemento superior, etc.). De acordo com esta posição, o elemento enviará pacotes para os elementos vizinhos na mesma ordem que está indicado nas figuras. Junto com o cabeçalho do pacote, é enviado um código que indica a direção de transmissão. Este código é armazenado no receptor para a devolução do resultado do processamento. O receptor só é informado do número de mensagens a serem recebidas.

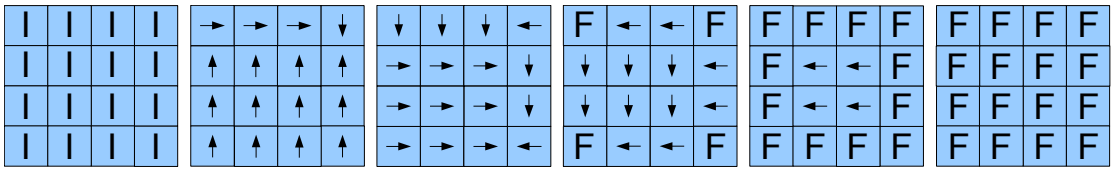


Figura 5.7: Ordem de envio dos pacotes, na horizontal e vertical, por cada um dos elementos de processamento, conforme sua posição sobre a imagem. I e F indicam o início e o final do envio, respectivamente.

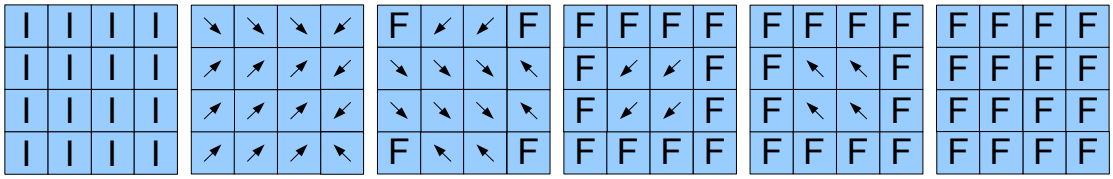


Figura 5.8: Ordem de envio dos pacotes, nas diagonais, por cada um dos elementos de processamento, conforme sua posição sobre a imagem. I e F indicam o início e o final do envio, respectivamente.

Devido à capacidade de armazenamento limitada dos P.E.s este esquema é realizado sucessivas vezes para a busca completa conforme aumenta a distância dos blocos até a fronteira, dentro de uma região. O processo para quando os blocos que situam-se em uma distância da fronteira igual à distância de busca forem atingidos. Esta operação foi realizada dividindo-se o conjunto de blocos em camadas, conforme a Fig 5.9. Cada camada consiste em duas linhas (acima e abaixo na região) e duas colunas (à esquerda e à direita na região, que serão enviadas aos elementos na horizontal e na vertical e em 4 linhas menores, que serão enviados aos elementos na diagonal. Optou-se pelo envio em linhas para os elementos de processamento nas diagonais devido à facilidade de geração do endereço das linhas. Na Figura 5.9 (a) é mostrada a distância (em blocos) de um bloco até a fronteira, considerando as linhas, e em (b) considerando as colunas. Já na Figura 5.9 (c) são mostrados os blocos enviados para os elementos nas diagonais. Embora todas as 3 camadas estejam representadas na mesma figura, elas são enviadas em etapas consecutivas, uma após a recepção dos resultados do processamento da outra. Em (d),(e),(f) são mostrados os elementos enviados nas 3 primeiras camadas, para os elementos na horizontal e na vertical.

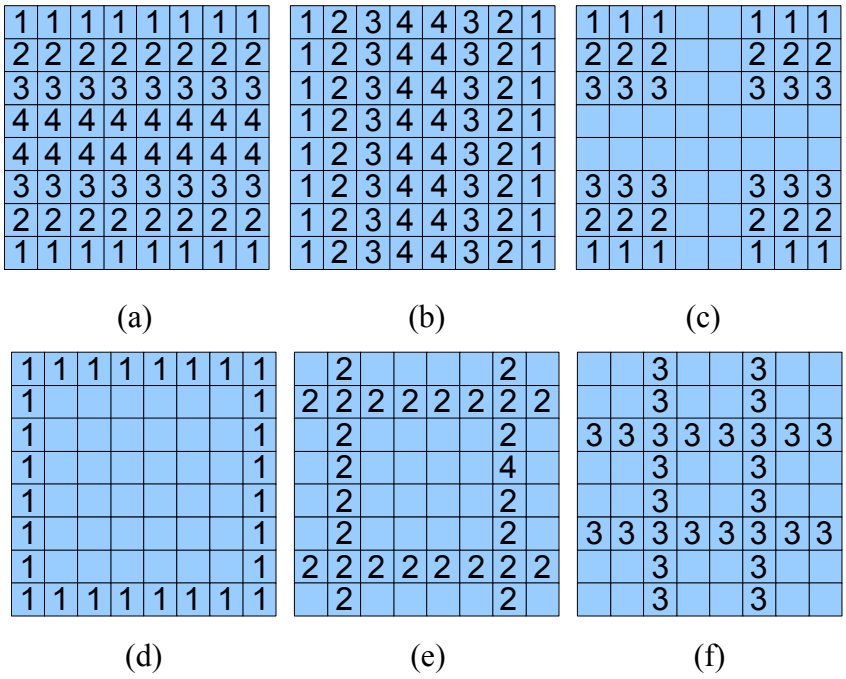


Figura 5.9: Ordem de envio dos blocos, em camadas, para os vizinhos, para uma região de 8x8 blocos. Esta ordem está associada à distância de um bloco até a borda, dentro de uma região da imagem. Para regiões com um número diferente de blocos, a numeração será diferente, mantendo o mesmo critério.

Os vetores de movimento calculados são enviados após a busca para o P.E. de origem. Neste P.E. o vetor é comparado com o já calculado para o mesmo bloco considerando a área de busca e o de menor módulo é armazenado.

A última etapa do processo é a devolução dos vetores de movimento para o mestre, onde é realizada a montagem dos dados em uma matriz única correspondente a toda a imagem.

O mestre da rede é informado na sua criação de quantos P.E.s existem na rede, para que possa realizar a divisão da imagem em regiões, enviá-las a cada um dos P.E.s e compor os vetores de movimento recebidos em uma única matriz correspondente à imagem. O mestre utiliza-se do mesmo gerenciador de IO disponível nos P.E.s.

De forma a facilitar a manutenção do código, os P.E.s e o mestre compartilham da mesma descrição do controle, que realiza diferentes funções conforme a identificação recebida.

5.2.2 Otimização do processamento através da mudança de algoritmo

Foram realizados experimentos com um algoritmo multi-resolução de forma a otimizar a execução do algoritmo de estimação de movimento na plataforma através do uso de uma boa aproximação para a solução, pois o algoritmo de busca completa apresentou uma demanda computacional que só permitiu atingir o desempenho desejado utilizando o ASIC como elemento de processamento. Este algoritmo passou por duas etapas durante os experimentos: particionamento e posicionamento das partes sobre a arquitetura.

5.2.2.1 Particionamento do algoritmo MRBMA

O algoritmo MRBMA (*Multi-Resolution Block Matching Algorithm*) pode ter seu desempenho incrementado através do particionamento para execução em diferentes elementos de processamento. Isto pode ser feito através da divisão da imagem de cada nível em regiões, e estas regiões serão processadas por P.E.s em paralelo, da mesma forma que o algoritmo FSBMA em cada nível. A diferença mais significativa estará na distância de busca, que no algoritmo MRBMA será bem menor. Observando a Tabela 3.10 pode-se constatar que o número de mensagens cresce com o quadrado da distância para uma determinada resolução, e assim o algoritmo MRBMA tem como vantagem, além da redução do tempo de computação, uma redução no número de mensagens que trafegam pela rede. O resultado disto será um menor consumo de potência por parte da rede. Outro ponto a ser considerado é se a divisão da imagem nos diferentes níveis será uniforme ou não. Optou-se por dividir a imagem em regiões do mesmo tamanho (número de pixels) em todos os níveis, sendo que no nível de menor resolução apenas um processador opera sobre toda a imagem. Na Tabela 5.8 pode-se observar o número de mensagens transmitidas em um nível como função do número de regiões, das dimensões das regiões e da distância de busca.

Tabela 5.8: Total de blocos transferidos em um mesmo nível

Nível	No de Proc	Blocos transferidos por nível (imagem atual)	Blocos transferidos por nível (img. referência)	Vetores de movimento
0	1	0	0	0
1	4	$4•D^2+(4•m+4•n-4)•D+$ $(-2•m-2•n+1)$	$2•m+2•n+1$	$4•D^2+(4•m+4•n-4)•D+$ $(-2•m-2•n+1)$
2	16	$36•D^2+$ $(24•m+24•n-36)•D+$ $(-12•m-12•n+9)$	$12•m+12•n+9$	$36•D^2+$ $(24•m+24•n-36)•D+$ $(-12•m-12•n+9)$
N	Px•Py	$(4•Px•Py-4•Px-4•Py+4)•D^2+$ $(-2•m•Px-2•n•Py+4•Px+4•Py+$ $2•m•Px•Py+2•n•Px•Py-4•Px•Py-4)•D+$ $(m•Px+n•Py-Px-Py-m•Px•Py-$ $n•Px•Py+Px•Py+1)$	$(Px•Py-Px)•m+$ $(Px•Py-Py)•n+$ $(Px•Py-Px-Py+1)$	$(4•Px•Py-4•Px-4•Py+4)•D^2+$ $(-2•m•Px-2•n•Py+4•Px+4•Py+$ $2•m•Px•Py+2•n•Px•Py-4•Px•Py-4)•D+$ $(m•Px+n•Py-Px-Py-m•Px•Py-$ $n•Px•Py+Px•Py+1)$

5.2.2.2 A comunicação entre níveis

Além da comunicação intra-nível, entre o processamento de dois níveis consecutivos deve haver uma transferência dos vetores de movimento encontrados em um nível para o nível seguinte, onde serão reutilizados. Contudo, estes vetores representam um volume de dados bem menor em comparação com o dos blocos. Um vetor de movimento vindo de um nível superior é ampliado e replicado nos 4 blocos correspondentes do nível inferior. Na Tabela 5.9 encontra-se um cálculo do número de vetores de movimento que serão transmitidos de um nível para outro.

Tabela 5.9: Número de vetores de movimento a serem transmitidos entre os níveis

Niveis	No de vetores de movimento
0,1	$Px•Py•m•n$
1,2	$Px•Py•4•m•n$
2,3	$Px•Py•16•m•n$
(N-1),N	$Px•Py•4^{(N-1)}•m•n$

5.2.2.3 A computação e a comunicação intra-nível

Pode-se modelar o problema de tal forma que o tempo de computação na busca multi-resolução será aproximadamente igual para cada um dos planos. Neste caso, cada plano disporá de elementos de processamento iguais e que atuarão sobre regiões do mesmo tamanho. Logo, se a resolução dobra nas duas direções x e y sobre a imagem, de um plano para outro, o número de processadores quadruplicará de um plano para outro. Partindo de uma imagem de baixa resolução sobre a qual atuará um processador, sobre a imagem do nível seguinte, com o dobro de resolução atuarão 4 processadores e assim

por diante. Logo, durante o processamento do nível 0, o tempo de computação será igual ao expresso pela Equação 3.9 e não haverão comunicações durante a busca.

Já no processamento do nível 1, o tempo total de computação, realizado por 4 processadores, será o mesmo, mas será adicionado um tempo necessário à troca de mensagens entre os processadores. Estes quatro processadores terão mensagens do tipo das que se encontram da Tabela 6 a Tabela 9, pois estarão nos cantos da imagem.

Já no nível 2, haverão 16 processadores e mensagens de todos os tipos, e assim sucessivamente. Um fator de grande importância na redução do tempo de busca é que, como a distância dm será bem menor do que d, o número de blocos a ser transferido entre os processadores nos diversos níveis será menor com a utilização da busca multi-resolução.

Aqui convém observar que embora parte da busca de alguns blocos seja realizada no elemento de processamento onde se encontra o bloco e parte no elemento de processamento vizinho, o tempo de computação total será praticamente o mesmo, pois este elemento de processamento também receberá blocos do vizinho para completar a busca. Neste caso deve-se acrescentar os ciclos necessários para a carga destes blocos, representado pelo fator 16 na equação (2), e que deve ser multiplicado pelo número de blocos recebidos de elementos vizinhos. Contudo, a influência destes ciclos de carga será menor.

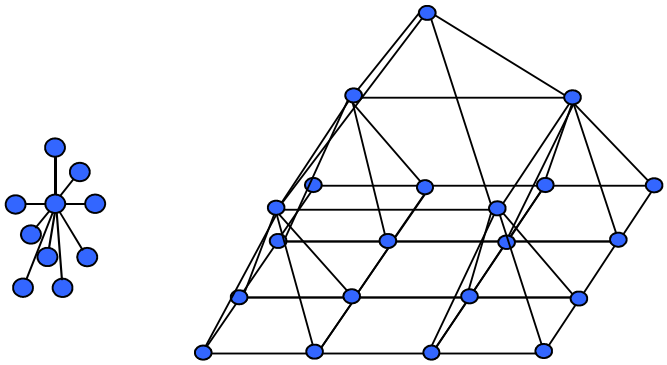
No caso da busca multinível, como a distância de busca será bem menor do que a distância utilizada na busca completa, o tempo de computação será reduzido em relação à busca completa. Com relação à comunicação, espera-se que nos locais onde há pouco ou não há movimento a comunicação seja bastante reduzida em relação à busca completa, uma vez que o vetor de movimento dos níveis superiores já informa que o centro da busca do nível inferior estará próximo da posição original e dm é pequeno. Já no caso da busca completa, é necessário o envio de blocos distantes da borda da região, uma vez que d é grande.

Assim, o uso da busca multi-resolução é mais adequado também do ponto de vista de comunicação quando o problema é particionado.

5.2.2.4 Mapeamento da aplicação para a arquitetura

Sabe-se que a arquitetura ideal pode ser uma pirâmide, com conexões ponto a ponto nos diversos níveis e entre os níveis. Neste caso, cada processador fica interconectado com quatro processadores em um nível inferior, aos quatro vizinhos e a um processador conectado no nível superior. Tal estrutura pode ser observada na Figura 5.10.

Contudo, o leiaute deste tipo de arquitetura não fica muito otimizado quando tal arquitetura é passada para duas dimensões, dentro de um SoC. Uma solução é mapear esta arquitetura para uma rede de interconexão com um leiaute mais regular e avaliar a penalidade pelo uso desta arquitetura não tão bem adaptada ao problema no nível arquitetural, mas que seja melhor adaptada ao posicionamento diretamente no silício. A simples interconexão ponto a ponto dos processadores entre si ainda não é a melhor solução, uma vez que existe a comunicação entre os diversos níveis.



(a) (b)

Figura 5.10: Arquitetura piramidal ; (a) 9 conexões de um elemento de processamento; (b) pirâmide com 3 níveis

Foram realizados alguns experimentos mapeando os elementos de processamento desta arquitetura para uma rede intra-chip bidimensional. Esta rede permite, graças a seus roteadores, a comunicação tanto entre núcleos vizinhos como entre núcleos distantes sem a intervenção dos núcleos não envolvidos na comunicação. Isto adiciona flexibilidade à arquitetura, uma vez que é possível posicionar em diferentes partes da rede os conjuntos de processadores que atuam em diferentes níveis, onde localmente se comunicarão entre si, e ao mesmo tempo a rede permitirá a comunicação entre os diferentes planos.

Uma vez que a comunicação em cada plano ocorre entre os elementos vizinhos e entre planos ocorre entre um elemento de um plano e os quatro do nível inferior, optou-se por mapear os diferentes planos em uma mesma região, de forma irregular, e os diferentes planos em áreas próximas entre si. Para algoritmos como MRBMA, esta alternativa mostra-se uma boa solução, pois a comunicação entre os planos consiste apenas em vetores de movimento, em uma resolução no mínimo quatro vezes menor (em cada eixo) do que a da imagem (quando utiliza-se blocos 4x4).

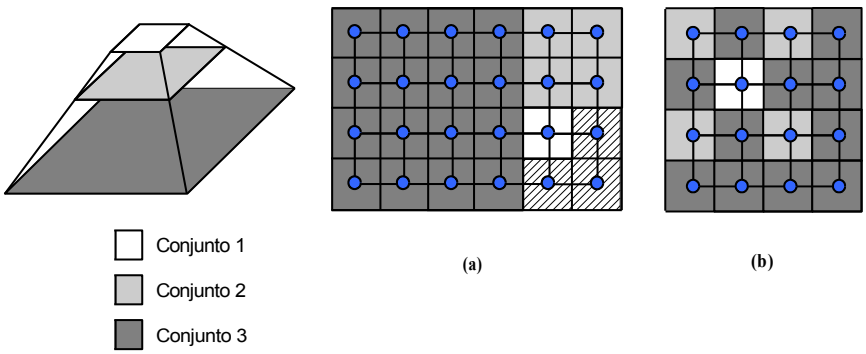


Figura 5.11: Exemplo de 3 conjuntos de processadores que atuam em diferentes planos e dois possíveis posicionamentos destes grupos em uma rede intra-chip do tipo grelha

É possível a utilização de algoritmos de mapeamento automático com base no grafo de volume de comunicações entre os elementos de processamento. Neste caso, é possível obter uma redução no consumo de energia do sistema se este consumo for levado em conta no mapeamento. Contudo, devido ao fato do volume de comunicação entre os elementos vizinhos ser maior do que entre os diferentes níveis, em geral as soluções tendem a ser como a solução apresentada na Figura 5.11(a), com diferenças na posição do conjuntos 2, que pode situar-se mais abaixo e do conjunto 3, que ocupará uma posição no espaço restante, próximo ao conjunto 2.

Nota-se aqui, que se fosse utilizado um arranjo tipo o apresentado na Figura 5.11(b), se teria uma menor área ocupada e um tempo menor de comunicação entre os diversos planos. Neste caso não é possível realizar o processamento nos níveis em paralelo, uma vez que os processadores são compartilhados entre os níveis.

Comparação entre os algoritmos FSBMA e MRBMA quanto ao número de mensagens total

Pode-se comparar os algoritmos multi-resolução com os algoritmos tradicionais não só quanto ao aumento de desempenho com a redução no volume de computação, mas também quanto ao tráfego na rede relacionado com certas aplicações. Conforme foi visto para o caso da estimação de movimento, é obtida uma estimativa da solução final nos níveis de menor resolução, a qual é refinada com a troca de nível e aumento da resolução. Contudo, em cada nível a quantidade de computação e de troca de dados entre P.E.s é reduzida, uma vez que baseia-se na estimativa obtida no nível anterior. Logo, estes algoritmos apresentam a possibilidade de redução de tráfego na rede.

Assumindo que tanto a arquitetura que realizará a busca completa como a que realizará a busca multi-resolução utilizarão o mesmo tipo de elemento de processamento, resta fazer um balanço do número de mensagens. Sabe-se número e tamanho das mensagens está relacionado com o número de chaveamentos nos elementos da rede, os quais são diretamente relacionados com a potência dinâmica consumida. Já o tempo de execução, que pode ser reduzido (considerando a parte de comunicação) através da diminuição do número de mensagens, destá relacionado diretamente com o consumo de potência estática. Contudo, a potência total só pode ser obtida através de uma simulação, dada a complexidade do sistema.

Na Tabela 5.10 pode-se observar o desempenho para cada um dos métodos de busca, quando mapeados em uma rede no chip e para diferentes números de processadores. Ao aplicar-se o modelo para imagens de resoluções típicas utilizadas nos padrões de vídeo, pode-se observar que o método multi-resolução não tem apenas vantagens sob o ponto de vista da computação mas também sob o ponto de vista de troca de mensagens.

Os resultados mostraram que com o uso de um método multi-resolução para a estimação de movimento, não somente a demanda computacional foi reduzida, como já era esperado, mas também o volume de comunicação entre os diferentes elementos de processamento. Isto implica, além da redução do tempo de computação, em uma redução do tempo de comunicação. Uma vez que a rede e os elementos de

processamento são os mesmos para ambos os métodos, o consumo de energia será também reduzido tanto por parte do elemento de processamento quanto por parte da rede intra-chip.

Tabela 5.10: Comparação entre os dois métodos de busca para diferentes resoluções de imagem e número de processadores

	FS				MRS					
	Número de mensagens	Computação (ciclos)	E.P.s	d	Número de mensagens	Computação (ciclos)	Níveis	Dnivel	deq	E.P.s
640x480	100,0k	17,5M	64	60	5,8k	408,0k	4	4	60	85
720x480	103,4k	19,3M	64	60	6,2k	448.8k	4	4	60	85
1600x1152	185,7k	105,4M	64	60	16,8k	2448k	4	4	60	85
1920x1080 (1920x1088)	199,1k	119,5M	64	60	18,6k	2774,4k	4	4	60	85

Uma vantagem do método escolhido é a possibilidade de reutilização da mesma arquitetura para os elementos de processamento, com a adição apenas do cálculo da pirâmide de imagens. É necessário também a mudança no padrão de comunicação da rede para a troca de vetores de movimento, que consiste basicamente em modificações na unidade de controle.

Na Tabela 5.11 pode-se observar uma estimativa para a freqüência de operação para os elementos de processamento desenvolvidos, para imagens de resolução 640x480 pixels e na Tabela 5.12 para uma matriz de 16 elementos de processamento, considerando 4 níveis de processamento e uma distância equivalente igual a 60.

Tabela 5.11: Estimativa de frequência de operação considerando cada elemento de processamento, para imagens com uma resolução de 640x480 pixels.

Elemento de Processamento	Número de ciclos/bloco	Número de ciclos/imagem	Imagens por segundo (1GHz)	Freq. de operação para 30 quadros/s
DSP	5574,7K	34,5M	29,0	1,034GHz
MIPS	4326,6K	26,8M	37,3	804,8MHz
MIPS com instruções multimídia	599,1K	3,9M	256,4	119,3MHz
ASIC	66,6K	0,7M	1428,6	21,4MHz

Tabela 5.12: Estimativa de desempenho considerando um sistema composto por uma matriz de 16 elementos de processamento, para imagens com uma resolução de 640x480 pixels.

Matriz de 16 Elementos de Processamento	Número de ciclos/bloco	Número de ciclos/imagem	Imagens por segundo (1GHz)	Freq. de operação para 30 quadros/s
DSP	5574,7K	2442,8K	409	73,3MHz
MIPS	4326,6K	1964,7K	509	58,9MHz
MIPS com instruções multimídia	599,1K	536,7K	1863	16,1MHz
ASICs	66,6K	332,7K	3005	9,9MHz

5.2.3 Conclusões

A arquitetura desenvolvida mostrou-se adequada para a execução de diferentes algoritmos de processamento de imagens, os quais foram particionados de maneira diferente, conforme o tipo de problema, e mostrou-se capaz de aproveitar o paralelismo que foi exposto.

O uso de um processador com otimizações permitiu a arquitetura acelerar a execução dos algoritmos estudados, mantendo ainda a flexibilidade para a execução de outros algoritmos desconhecidos. Para os casos onde a especialização não mostra-se útil, o paralelismo entre tarefas ainda pode ser utilizado para obter uma melhoria no desempenho.

A comparação realizada entre diferentes processadores mostrou que não há diferenças substanciais quando considerado o número de ciclos para a execução de uma operação de busca, para tarefas simples (monotarefa).

O ganho de desempenho obtido para o sistema final pode ser então dividido em

1. ganho obtido através da exploração do paralelismo entre processadores;
2. ganho obtido através da exploração do paralelismo dentro dos operadores, em cada processador;
3. ganho obtido através da exploração do paralelismo entre operadores (no caso do ASIC);
4. ganho obtido através da exploração do paralelismo na comunicação entre os componentes do sistema;
5. ganho obtido através da mudança de algoritmo.

O primeiro ganho vem com o custo do particionamento do problema, tratamento das dependências de dados e uso de um sistema com uma área proporcionalmente maior, mas mantendo a capacidade de processamento para os mais diversos problemas (propósito geral).

O segundo e o terceiro ganho significam limitar a aceleração obtida a um grupo de operações presentes no algoritmo, o que limita um pouco a sua aplicação, mas ainda mantém a programabilidade e reuso para outras aplicações.

O quarto ganho, embora pouco expressivo para o exemplo de estimação de movimento devido ao pequeno tempo de comunicação entre os elementos durante o processamento, pode vir a tornar-se bastante significativo no momento em que as comunicações têm duração da mesma ordem de grandeza que o processamento.

Já o quinto ganho implica em modificar o próprio problema, o que em certos casos, como o apresentado anteriormente, significa ter de aceitar uma aproximação para o a solução deste problema. Em outros casos, pode-se simplesmente trocar um algoritmo por outro mais eficiente, mas que produza a mesma solução.

Assim, o ganho de desempenho total é igual ao produto dos ganhos obtidos pela exploração do paralelismo nos N eixos do espaço de projeto, mantendo o algoritmo original. A mudança de algoritmo, quando possível, também trará um ganho associado.

Dos resultados obtidos também é possível observar que, sem a realização de mudanças substanciais no algoritmo, os maiores ganhos de desempenho serão obtidos na exploração do paralelismo entre elementos de processamento e através de modificações nos operadores, estando aqui respeitados os limites da parte paralelizável do problema.

6 COMENTÁRIOS FINAIS

Neste trabalho foram realizados experimentos com algoritmos de processamento de imagens buscando explorar diferentes aspectos da estratégia de modelamento e da exploração do paralelismo resultante. A primeira estratégia utilizada como estudo de caso foi o particionamento do algoritmo, com a execução das mesmas operações em todos os elementos de processamento. A segunda estratégia foi a distribuição das funções do algoritmo em elementos de processamento conectados formando um macro pipeline. A terceira estratégia foi o particionamento ds dados em conjuntos de mesmo tamanho, com a execução de diferentes operações em cada conjunto.

Foi proposta uma arquitetura que permite a execução dos algoritmos considerando o paralelismo de acordo com as estratégias adotadas no modelamento. Esta arquitetura é composta por um conjunto de elementos de processamento, interconectados por uma rede intrachip. A escolha da infraestrutura de comunicação foi baseada na busca por um paralelismo nas comunicações entre os elementos de processamento. Também o fator flexibilidade na realização de comunicações, isto é, a capacidade da troca de dados entre quaisquer elementos da rede e sem a interrupção dos elementos de processamento envolvidos no caminho dos dados, foi considerado. O uso de comunicações do tipo ponto a ponto é comum em comunicações entre elementos adjacentes, mas requer a interrupção de todos os elementos encontrados no caminho percorrido pelos dados.

Considerando a parte da arquitetura relacionada ao processamento, os experimentos realizados apontam na direção da execução de algoritmos de processamento de imagens de maior demanda computacional com uso do paralelismo em granularidade mais grossa, com diversos elementos de processamento executando operações em paralelo em conjunto com paralelismo dentro dos elementos de processamento, com operadores especializados. A utilização de um conjunto maciço de elementos de processamento muito simples, similar aos presentes nos chips de visão, mostra-se adequado apenas para a execução de funções extremamente simples, tais como estimação de movimento para pequenas distâncias ou cálculo de centróides, conforme apresentado na literatura, e assim como impõe o desenvolvimento do algoritmo de uma forma extremamente regular. Já a utilização de elementos de processamento mais complexos implica em uma utilização de um número menor de elementos, devido à área ocupada, bem como reduz o desempenho caso seja mantida a execução serial de instruções, o que leva ao uso de freqüências mais altas. O uso de elementos de processamento dedicados tende a resultar no melhor casamento entre freqüência de operação e número de elementos de processamento, contudo limita o espectro de aplicação da arquitetura. Uma solução intermediária, que apresenta flexibilidade e desempenho tenderá a ser composta por

elementos de processamento programáveis, com alguns operadores dedicados, tendo em vista as aplicações de maior demanda computacional que poderão ser executadas na arquitetura e ainda permitirá o reuso. Um exemplo são os elementos de processamento apresentados, constituídos por processadores simples, e com a unidade lógico-aritmética modificada para a realização eficiente da operação de SAD.

Considerando o conjunto computação e comunicação, os experimentos mostraram que os maiores ganhos foram obtidos na exploração do paralelismo. Isto ficou claro na seção onde o mapeamento de tarefas foi explorado. Para um problema relativamente complexo, como a execução de compactação JPEG, o ganho em desempenho foi de cerca de 100% para a paralelização das funções em apenas dois processadores. Ao mesmo tempo, o ganho obtido com o posicionamento das tarefas foi de cerca de 40%. Da mesma forma, o ganho obtido com a especialização do processamento também foi maior do que o ganho obtido com o mapeamento de tarefas. Isto leva a crer que a infraestrutura de comunicação escolhida já permitiu a troca de mensagens de forma relativamente eficiente mesmo sem uma otimização das comunicações. Assim, conforme as conclusões do capítulo 5, uma possível seqüência de passos para o projeto de uma arquitetura de processamento de imagens, visando aumentar o desempenho da arquitetura com um esforço próximo do mínimo, deve seguir os seguintes passos (em ordem decrescente de possíveis ganhos após uma otimização):

- escolha do algoritmo que apresente o melhor desempenho, com um resultado aceitável (mesmo sendo uma aproximação) e seja paralelizável;
- particionamento do algoritmo e atribuição a diferentes elementos de processamento;
- especialização dos elementos de processamento para exploração do paralelismo dentro dos operadores (como por exemplo as diferentes formas de implementação de SAD apresentadas);
- otimização do uso da infra-estrutura de comunicação, através de posicionamento das tarefas ou de núcleos especializados.

A descrição da arquitetura sob a forma de um *soft core* também possibilita a parametrização e consequente otimização da quantidade de recursos utilizados para determinadas aplicações. Isto confere um maior grau de flexibilidade à arquitetura, pois permite o seu reuso combinado com um ajuste do volume de processamento para a aplicação, no momento do projeto.

Os experimentos também mostraram que o processamento de vídeo em tempo real, já em resolução padrão VGA, não é possível sem a utilização do processamento paralelo, apesar dos inúmeros avanços nas freqüências de operação dos processadores nos últimos anos. Esta condição deve se manter, uma vez que a resolução dos sensores vem aumentando com o passar dos anos e novos formatos de vídeo (HDTV – *High Definition Television*, por exemplo) tem surgido.

Como trabalhos futuros serão implementados outros algoritmos na arquitetura de forma a explorar suas características e capacidade de expansão.

REFERÊNCIAS

AKITA, J. et al. An image sensor with fast objects' position extraction function. **IEEE Transactions on Electron Devices,** New York, v.50, n.1, p.184 – 190, Jan. 2003.

ALBANESI, M. G. et al. A Vlsi 128-processor chip for multiresolution image processing. In: INTERNATIONAL CONFERENCE ON MASSIVELY PARALLEL COMPUTING SYSTEMS, 1., 1994. **Proceedings...** [S.l.:s.n.], 1994. p.296 – 307.

ASANO, T. et al. Low-Power Design Approach of 11FO4 256-Kbyte Embedded SRAM for the Synergistic Processor Element of a Cell Processor. **IEEE Micro**, New York, v.25, n.5, p.30 – 38, Sept.-Oct. 2005.

BECK FILHO, A. C.; WAGNER, F. R.; CARRO, L. CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 16., 2003. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p.349-354.

BOSCHETTI, M. R.; SILVA, I. S.; BAMPI, S. A run-time reconfigurable datapath architecture for image processing applications**.** In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2004. **Proceedings...** Los Alamitos: IEEE Computer Society, 2004. p.242-247.

BOUAZIZ, S. et al. Some hardware and software considerations for the Multi-SIMD control strategy of massively parallel machines. In: ANNUAL EUROPEAN COMPUTER CONFERENCE, COMPEURO, 5., 1991, Bologna. **Proceedings...** Los Alamitos:IEEE Computer Society Press, 1991. p.180 – 183.

CANTONI, V.; LEVIALDI, S. Multiprocessor computing for images**. Proceedings of the IEEE,** New York, v.76, n.8, p.959 – 969, 1988.

CHAI, S. M. et al. Hyper-spectral image processing applications on the SIMD Pixel Processor for the digital battlefield. In: IEEE WORKSHOP ON COMPUTER VISION BEYOND THE VISIBLE SPECTRUM: METHODS AND APPLICATIONS, CVBVS, 1999. **Proceedings...** [S.l.:s.n.], 1999. p.130 – 138.

CHALLAPALI, K. et al. Real-time object segmentation and coding for selective-quality video communications. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.14, n.6, p.813 – 824, 2004.

CRADLE TECHNOLOGIES. **CT3600 Family of Multiprocessor DSPs – Product Brief**. [S.l.], 2005. Disponível em: <http://www.cradle.com/downloads/3056_CT3600_v3.pdf>. Acesso em: jan. 2006.

D'AMORE, R. **VHDL – Descrição e Síntese de Circuitos Digitais.** Rio de Janeiro : LTC, 2005.

DASU, A.; PANCHANATHAN, S. Reconfigurable media processing. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: CODING AND COMPUTING, 2001. **Proceedings...** [S.l.:s.n.], 2001. p.300 – 304.

DASU, A.; PANCHANATHAN, S. A survey of media processing approaches. **IEEE Transactions on Circuits and Systems for Video Technology,** New York, v.12, n.8, p.633 – 645, 2002.

DEHON, A.; WAWRZYNEK, J. Reconfigurable computing: what, why, and implications for design automation. In: DESIGN AUTOMATION CONFERENCE, DAC, 36., 1999, New Orleans. **Proceedings ...** New York:ACM, 1999. p.610 – 615.

DUDEK, P.; HICKS, P. J. A general-purpose processor-per-pixel analog SIMD vision chip. **IEEE Transactions on Circuits and Systems I**, New York, v.52, n.1, p. 13 – 20, 2005.

EHANDARKAR, S. M.; ARABNIA, H. R. Parallel computer vision on a reconfigurable multiprocessor network. **IEEE Transactions on Parallel and Distributed Systems**, New York, v.8, n.3, p.292 – 309, 1997.

ELLIOT, D.; SNELGROVE, W.; STUMM, M. Computational RAM: A Memory-SIMD Hybrid and its Application to DSP. In : IEEE CUSTOM INTEGRATED CIRCUITS CONFERENCE , CICC, 1992. **Proceedings ...** [S.l.:s.n.], 1992.

ENOMOTO, T.; EI, T. Low-power CMOS circuit techniques for motion estimators. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2003. **Proceedings ...** [S.l.:s.n.], 2003. v.5, p.V-409 - V-412.

ETIENNE-CUMMINGS, R.; KALAYJIAN, Z. K.; CAI, D. A programmable focal-plane MIMD image processor chip. **IEEE Journal of Solid-State Circuits**, New York, v.36, n.1, p.64 – 73, 2001.

EZER, G. Xtensa with user defined DSP coprocessor microarchitectures. In: INTERNATIONAL CONFERENCE ON COMPUTER DESIGN, 2000. **Proceedings...** [S.l.:s.n.], 2000. p.335 – 342.

FIGUEIRÓ,T.; SOARES,A.; GUIMARÃES,L.; SUSIN,A. Gradient Pile Up for Edge Detection on a DSP. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 2005, Florianópolis-Brasil. **Student Fórum...** [S.l.:s.n.], 2005.

FLYNN, M.J. Very high-speed computing systems. **Proceedings of the IEEE**, New York, v.54, n.12, p.1901 – 1909, 1966.

FUJIYOSHI, T. et al. An H.264/MPEG-4 audio/visual CODEC LSI with module-wise dynamic voltage/frequency scaling. In: INTERNATIONAL SOLID-STATE CIRCUITS

CONFERENCE, ISSCC, 2005. **Digest of Technical Papers...** [S.l.:s.n.], 2005. v.1. p.132 – 589.

GEALOW, J. C. et al. System design for pixel-parallel image processing. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, New York, v.4, n.1, p. 32 – 41, 1996.

GONZALES, R.; WOODS, R. E. **Digital Image Processing**. New York: Addison Wesley, 1993. 716 p.

GONZALEZ, R. E. Xtensa: a configurable and extensible processor. **IEEE Micro**, New York, v.20, n.2, p. 60 – 70, 2000.

GUIMARÃES, L. V.; SOARES, A. B.; SUSIN, A. A.; SILVA, V.C. Gradient Pile up Algorithm for Edge Enhancement and Detection. In: INTERNATIONAL CONFERENCE ON IMAGE ANALYSIS AND RECOGNITION, ICIAR, 2004, Porto. **Proceedings...** Berlin:Springer, 2004. p.187-194. (Lecture Notes in Computer Science, v.3211).

HARTENSTEIN, R. Coarse Grain Reconfigurable Architectures. In: ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASPDAC, 2001. **Proceedings...** [S.l.:s.n.], 2001. p.564-569.

HARTENSTEIN, R. A Decade of Reconfigurable Computing: a Visionary Perspective. In: DESIGN, AUTOMATION AND TEST IN EUROPE, DATE, 2001, Munich. **Proceedings...**, [S.l.]:IEEE, 2001.p.642-649.

HENNESSY, J. L. **Computer organization and design :** the hardware/software interface. San Francisco: Morgan Kaufmann, 1998. 759 p.

HIRSCHMÜLLER, H.; INNOCENT, P.; GARIBALDI, J. Real-Time Correlation-Based Stereo Vision with Reduced Border Errors**. International Journal of Computer Vision,** [S.l.] v.47,n.1/2/3, p.229–246, 2002.

HWANG, K.; BRIGGS, F. **Computer Architecture and Parallel Processing.** New York: McGraw Hill Book Company, 1984. 846 p.

HWANG, K.; DEGROOT, D. **Parallel Processing for Supercomputers & Artificial Intelligence**. New York: McGraw Hill Book Company, 1989. 673 p.

HWANG, K. **Advanced Computer Architecture:** Parallelism, Scalability, Programmability. New York: McGraw Hill Book Company, 1993. 770 p.

IKENAGA, T.; OGURA, T. A DTCNN universal machine based on highly parallel 2-D cellular automata CAM2. **IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications**, New York, v.45, n. 5, p.538 – 546, 1998.

IKENAGA, T.; OGURA, T. A fully-parallel 1 Mb CAM LSI for real-time pixel-parallel image processing. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE, ISSCC, 1999. **Digest of Technical Papers...** [S.l]:IEEE, 1999. p.264 – 265.

KAGAMI, S.; et al. A real-time visual processing system using a general-purpose vision chip. In: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, ICRA, 2002. **Proceedings...** [S.l.]:IEEE, 2002. v.2. p.1229 – 1234.

KAGAWA, K. et al. Pulse-domain digital image processing for vision chips employing low-voltage operation in deep-submicrometer technologies. **IEEE Journal of Selected Topics in Quantum Electronics**, New York, v.10, n.4, p.816 – 828, 2004.

KAPASI, U. J. et al. Programmable stream processors. **Computer**, New York, v.36, n.8, p.54 – 62, 2003.

KIM, J.; KIM,Y. Performance monitoring and tuning for a single-chip multiprocessor digital signal processor. In: IEEE INTERNATIONAL CONFERENCE ON ALGORITHMS AND ARCHITECTURES FOR PARALLEL PROCESSING, ICAPP, 2., 1996. **Proceedings...** [S.l.]:IEEE, 1996. p.76 – 83.

KOMURO, T.; KAGAMI, S.; ISHIKAWA, M. A dynamically reconfigurable SIMD processor for a vision chip. **IEEE Journal of Solid-State Circuits**, New York, v.39, n.1, p.265 – 268, 2004.

KREUTZ, M. **Método para a otimização de plataformas arquiteturais para sistemas multiprocessados heterogêneos**. 2005. 171f. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

KUROKAWA, M. et al. 5.4 GOPS linear array architecture DSP for video-format conversion**.** IN: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE , ISSCC, 43., 1996. **Digest of Technical Papers...** [S.l.]:IEEE,1996. p.254 - 255.

LIN, H. D. et al. A programmable motion estimator for a class of hierarchical algorithms. In: WORKSHOP ON VLSI SIGNAL PROCESSING, VIII, 1995, Sakai. **Proceedings...**, [S.l.]:IEEE , p.411 – 420, 1995.

LINDGREN, L. et al. A multi-resolution 100-GOPS 4-Gpixels/s programmable smart vision sensor for multisense imaging. **IEEE Journal of Solid-State Circuits**, New York, v.40, n.6, p.1350 – 1359, 2005.

MARCON, C.A.M.; BORIN, A.; SUSIN, A.; CARRO, L.; WAGNER,F. Time and Energy Efficient Mapping of Embedded Applications onto NoCs. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 2005, Shangai, China. **Proceedings...** Piscataway, N.J.:IEEE, 2005. v.1, p.33-38.

MORRIS, T. et al. A column-based processing array for high-speed digital image processing. In: CONFERENCE ON ADVANCED RESEARCH IN VLSI, 1999. **Proceedings...** [S.l.:s.n.], 1999. p.42 – 56.

MURAMATSU, S. et al. Image processing LSI “ISP-IV” based on local parallel architecture and its applications**.** In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, ICIP, 1998. **Proceedings...** [S.l.:s.n.], 1998. v.3, p.1000 – 1004.

NAKAMURA, K.; YOSHITOME, T.; YASHIMA, Y. Super high resolution video codec system with multiple MPEG-s HDTV codec LSI's. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2004. **Proceedings...** [S.l.:s.n.], 2004. v.3, p.793-6.

NASSIF,S.; CAPSON,D. Real-Time Template Matching Using Cooperative Windows. In: IEEE - CANADIAN CONFERENCE ON ELECTRICAL AND COMPUTER ENGINEERING, CCECE, 1997. **Proceedings...** [S.l.]:IEEE, 1997. v.2, p.391-394.

NAVAUX, P.O.A.; DE ROSE, C. **Arquiteturas Paralelas.** Porto Alegre: Sagra-Luzzato, 2003.

NGOUNGA, A.; SASSATELLI,G.; TORRES1,L.; GIL,T.; SOARES,A.B.; SUSIN, A.A. A contextual resources use: a proof of concept through the APACHES' platform In: IEEE WORKSHOP ON DESIGN AND DIAGNOSTICS OF ELECTRONIC CIRCUITS AND SYSTEMS, DDECS, Prague, 2006. **Proceedings...** [S.l.:s.n.], 2006.

NGOUNGA, A.; SASSATELLI, G.; TORRES, L.; GIL, T.; SOARES, A. B.; SUSIN, A.A. Run-time resources management on coarse grained, packet-switching reconfigurable architecture: a case study through the APACHES' platform. In: INTERNATIONAL WORKSHOP ON APPLIED RECONFIGURABLE COMPUTING, ARC, Delft,2006. **Proceedings...** [S.l.:s.n.], 2006.

PHAM, D. et al. The Design and Implementation of a First-Generation CELL Processor. In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE, ISSCC, 2005. **Proceedings...** [S.l.]:IEEE, 2005. p. 184-185.

PITAS, I. **Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks.** Chichester: John Wiley & Sons, 1993.

RAJAGOPAL, S.; CAVALLARO, J.R.; RIXNER, S. Design space exploration for real-time embedded stream processors. **IEEE Micro**, [S.l.], v.24, n.4, p.54 – 66, 2004.

RICHARDSON, I. E. G. **H.264 and MPEG-4 Video Compression – Video Coding for the Next-generation Multimedia.** Chichester: John Wiley & Sons, 2003.

RIXNER, S. et al. A bandwidth-efficient architecture for media processing. In: ANNUAL ACM / IEEE INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, MICRO, 31., 1998. **Proceedings...** New York: ACM, 1998. p.3-13.

SASSATELLI, G. et al. The Systolic Ring: A Scalable Dynamically Reconfigurable Core for Embedded Systems. In: SOPHIA ANTIPOLIS FORUM OF MICROELECTRONICS, SAME, 2002. **Proceedings...** [S.l.:s.n.], 2002.

SINGH, H. et al. MorphoSys: a reconfigurable architecture for multimedia applications**.** In: BRAZILIAN SYMPOSIUM ON INTEGRATED CIRCUIT DESIGN, SBCCI, 11., 1998. **Proceedings...** Los Alamitos: IEEE Computer Society, 1998. p.134 – 139.

SOARES, A. B.; GUIMARÃES, L. V.; CORDEIRO, V.; SUSIN, A. A. Gradient Pile up for Edge Detection on Hardware. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005 Kobe, Japan. **Proceedings...** Los Alamitos: IEEE, 2005. 1 CD-ROM.

SOARES, A. B.; BOSCHETTI, M.; CARRO, L. et al. Power Consumption Analysis in Architectures for the Wavelet Transform Processing. In: WORKSHOP IBERCHIP, 10., Cartagena de Indias, 2004. **Resumos...** Cartagena de Indias: Universidad de Los Andes, 2004. p.78-79.

SOARES, A. B.; CARRO, L.; SUSIN, A. A. Reconfigurable Communications for Image Processing Applications. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2006, Greece. **Proceedings...** [S.l.]:IEEE, 2006.

SONG, B. C.; RA, J. B. A fast motion estimation algorithm based on multi-resolution frame structure. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 1999. **Proceedings...** [S.l.:s.n.], 1999. v.6, p.3361 – 3364.

SONG, B. C. et al. Fast multi-resolution motion estimation algorithm and its VLSI architecture. In INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS, 2005. **Proceedings...** [S.l.:s.n.], 2005. p.71 – 72.

TREMEAC, Y. G.; INGGS, M. R. An example of rapid prototyping on the TMS320C80 Multimedia Video Processor (MVP). In: SOUTH AFRICAN SYMPOSIUM ON COMMUNICATIONS AND SIGNAL PROCESSING, COMSIG, 1998. **Proceedings...** [S.l.:s.n.], 1998. p.233 – 236.

TUNG, Y. et al. DSP-based multi-format video decoding engine for media adapter applications. In: INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS, ICCE, 2005. **Digest of Technical Papers...** [S.l.:s.n.], 2005. p.139 – 140.

UHR, L. **Parallel Computer Vision**. Boston: Academic Press, 1987.

UJVAL, J. et al. Programmable Stream Processors. **IEEE Computer**, Los Alamitos, p.54-62, 2003.

VAN DER WAL, G.; HANSEN, M.; PIACENTINO, M. The Acadia vision processor. In: IEEE INTERNATIONAL WORKSHOP ON COMPUTER ARCHITECTURES FOR MACHINE PERCEPTION, CAMP, 5., 2000. **Proceedings...** [S.l.]: IEEE, 2000. p.31 – 40.

VELTEN, J.; KUMMERT, A. Implementation of a high-performance hardware architecture for binary morphological image processing operations. In: MIDWEST SYMPOSIUM ON CIRCUITS AND SYSTEMS, MWSCAS, 47., 2004. **Proceedings...** [S.l.:s.n.], 2004. v.2, p. II-241 - II-244.

WILLS, D. S. Smart pixel architectures for image processing. In : IEEE/LEOS 1996 SUMMER TOPICAL MEETINGS: ADVANCED APPLICATIONS OF LASERS IN MATERIALS PROCESSING, BROADBAND OPTICAL NETWORKS/SMART PIXELS/OPTICAL MEMS AND THEIR APPLICATIONS, 1996. **Proceedings...** [S.l.]: IEEE,1996. p.93 – 94.

WILLS, D. S. et al. Processing architectures for smart pixel systems. **IEEE Journal of Selected Topics in Quantum Electronics**, [S.l.], v.2, n.1, p.24 – 34, 1996.

WU, Q.; HE, X.; HINTZ, T. Image Segmentation on Spiral Architecture. In: PAN-SYDNEY AREA WORKSHOP ON VISUAL INFORMATION PROCESSING **,** VIP, 2001, Sydney – Australia. **Proceedings...** New York: ACM, 2001.

ZATARI, A.; DODDS, G. Practical stereo vision and multi-laser scanning in object face detection and orientation determination. In: IEEE / RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, IROS, 1997. **Proceedings...** New York: IEEE, 1997. v.2, p. 746 – 751.

ZEFERINO, C. A. **Redes-em-Chip :** arquiteturas e modelos para avaliação de área e desempenho. 2003. 242f. Tese de Doutorado (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

APÊNDICE APLICAÇÃO DE SEGMENTAÇÃO DE IMAGENS

O uso de métodos de inspeção visual e processamento de imagens em aplicações diversas tem crescido bastante nos últimos anos em decorrência do significativo aumento da capacidade computacional aliada a popularização de câmeras digitais. Contudo, para muitas destas aplicações o desempenho de um único processador de prateleira ainda não é o suficiente. Em outros casos, o desempenho é aceitável, mas por apresentar um consumo de potência elevado, não torna-se prático quando este sistema utiliza baterias (sistema embarcado). Nestes casos, existe a necessidade de realizar uma exploração do espaço de projeto para buscar a alternativa que satisfaz os requisitos de desempenho e potência.

Este trabalho descreve uma aplicação que será utilizada como estudo de caso, e um algoritmo parametrizável criado para ser utilizado em uma exploração de espaço de projeto, permitindo a determinação do número de elementos de processamento.

Problema

Dada uma imagem binária, com pixels que representam o fundo com valor 0 e pixels que representam os objetos com valor 1, gerar uma imagem com os pixels de cada objeto identificados com um valor. Este valor será um número associado ao objeto.

Deseja-se que a solução proposta permita resolver o problema de forma paralela e com a exploração do paralelismo parametrizável.

Soluções Propostas

Versão Serial

Propõe-se inicialmente realizar uma varredura na imagem, marcando em cada pixel se este já foi visitado.

Quando se tratar de um pixel pertencente ao fundo, nenhuma informação é adicionada ao pixel. No caso de se encontrar um pixel de um objeto, este é marcado com o número de identificação do objeto, que é iniciado em 1 e incrementado a cada objeto.

Um objeto é completamente marcado quando o primeiro pixel é encontrado. Durante a marcação, após um pixel do objeto ser verificado, seus 4 vizinhos são também verificados, e assim sucessivamente, de forma recursiva. Se o pixel sendo verificado

pertencer ao objeto, mas já tiver sido visitado, ou se for um pixel representando o fundo, nenhuma informação é adicionada e seus vizinhos não são visitados. Se o pixel não tiver sido visitado, é adicionada a informação do número do objeto e os vizinhos são verificados. Este processo ocorre de forma recursiva, até que todos os pixels do objeto tenham sido verificados. Neste ponto, a varredura prossegue até o fim da imagem, verificando se há outros objetos.

Versão Paralela

Com a finalidade de verificar se é possível explorar de forma eficiente o paralelismo presente na solução proposta foi desenvolvida uma segunda solução paralela. Nesta versão, a imagem é dividida em M x N sub-imagens. Em cada uma das sub-imagens é executado o algoritmo da versão serial. Contudo, após esta etapa, cada sub-imagem terá uma numeração própria, pois supõe-se que não há como saber a priori quantos objetos existirão em cada sub-imagem. Poderão existir também objetos nas fronteiras entre duas ou mais sub-imagens, os quais devem ser identificados corretamente como apenas um objeto.

A determinação das intersecções é realizada após o processamento de cada sub-imagem utilizando a versão serial. A coluna da direita de uma sub-imagem é comparada com a coluna da esquerda da sub-imagem da sua direita(quando existir). A linha inferior de uma sub-imagem também é comparada com a linha superior da sub-imagem de baixo (quando existir). Se houverem dois pixels na mesma coluna diferentes do fundo (quando estiver verificando as linhas) ou dois pixels na mesma linha (quando estiver verificando as colunas), se considera que uma intersecção foi encontrada.

Optou-se por gerar um algoritmo que, dado uma lista de pares indicando as correspondências de dois objetos em duas sub-imagens vizinhas com um único objeto e o número de regiões de cada sub-imagem, gera uma numeração única para todas as sub-imagens. Esta numeração única é então substituída em cada região para produzir uma imagem final com numeração única dos objetos.

Este algoritmo funciona da seguinte maneira:

1) É criada uma lista (lista A) de todos os objetos de todas as sub-imagens. Esta lista retém a informação de qual sub-imagem é a detentora do objeto.

2) É criada uma lista (lista B) de todas as intersecções de objetos entre as sub-imagens vizinhas. Esta lista retém a informação de quais são as duas sub-imagens as quais se refere cada intersecção

3) Substitui-se na lista A o identificador de cada objeto por um identificador global único.

4) A cada substituição na lista A, verifica-se na lista B quais os elementos da lista que se referem a mesma sub-imagem e possuem a mesma numeração e substitui-se estes elementos pela numeração global.

5) De posse das duas listas já com uma numeração global, inicia-se o processo de eliminação de intersecções. Para cada elemento da lista B escolhe-se arbitrariamente um dos elementos do par e substitui-se nas duas listas este elemento pelo outro elemento do par. Este processo equivale a unir os objetos partidos entre duas ou mais sub-imagens. A

substituição na lista B garante que não haverão substituições incorretas (ou seja, elimina-se uma referência adicional a um mesmo objeto e impede-se que esta referência seja utilizada em futuras substituições).

6) Ao final do processo, tem-se na lista A uma lista com todos os objetos de todas as imagens com a numeração global. Se diferentes objetos dentro da região tiverem um mesmo número global significa que são partes do mesmo objeto, unidas em diferentes sub-imagens.

7) O último passo consiste em percorrer a lista A substituindo a numeração resultante, que pode não ser uniforme por uma numeração uniforme. Isto é feito montando-se uma tabela de substituição, adicionando-se um novo elemento a tabela cada vez que um objeto diferente for encontrado.

8) A numeração final é substituída em cada sub-imagem.

Implementação Final

A versão final da aplicação SegImage foi descrita em C++. Cada tipo de componente do sistema, é representado por uma classe. A comunicação entre tarefas é realizada através de funções especializadas, que representarão trocas de mensagens. Utilizou-se funções de recebimento de dados, que são chamadas pelo componente que deseja enviar a mensagem. Assim, quando a aplicação inicia a sua execução, é necessário passar um ponteiro para referenciar o objeto de destino, de forma que o componente que deseja enviar a mensagem possa chamar a função no objeto de destino. Desta forma o envio de uma mensagem fica explícito e pode-se simplesmente substituir esta chamada de função no objeto de destino por uma função do tipo *envia*. Isto será útil se a arquitetura for executada em um hardware com suporte a função send, e não mais uma simulação.

Foi criada uma classe para o elemento de processamento, responsável pelo processamento de uma sub-imagem. Os dados mais importantes a serem armazenados nesta classe são a matriz que armazena a imagem de entrada, a que armazena a imagem de saída e a pilha que armazena as coordenadas durante o preenchimento de um objeto. As principais funções são visitar um pixel, que trata da marcação com um número de objeto e a visita a pixels vizinhos, a função de processamento de fronteiras e as funções de recebimento de mensagens.

Foi criada também uma classe para o processador central, responsável pela geração da numeração global única. As principais funções são as responsáveis pela montagem da lista de intersecções e função de montagem da tabela de saída.

A divisão da imagem em sub-imagens é feita através da divisão da imagem na direção horizontal (M colunas) e na direção vertical (N linhas). Valores potências de 2 produzem um funcionamento correto. Pode-se utilizar no caso extremo apenas um elemento de processamento, e neste caso não é necessário o uso do processador central, uma vez que não existem intersecções de objetos, pois não existem sub-imagens.

Vantagens para a implementação em hardware

Da forma como foi proposta a solução do problema, foram atingidos os objetivos de exploração do paralelismo através da possibilidade de execução da etapa de rotulação em paralelo nas M x N sub-imagens.

Pode-se supor que a imagem inicial encontra-se armazenada em uma memória principal e que cada sub-imagem será processada em um núcleo com uma memória local. Assim, tem-se para uma imagem dividida em M x N sub-imagens, M x N núcleos ou elementos de processamento. Uma boa forma de interligar estes núcleos é através da utilização de uma Rede Intra-chip ou NoC (Network on Chip). A utilização de tal rede permite também a interligação da matriz de M x N elementos de processamento ou P.E.s (*processing elements*) com elementos adicionais no sistema, como a memória principal e um processador central onde será determinada a numeração única. Esta mesma rede permite que os núcleos troquem informações sobre linhas e colunas para determinação das intersecções.

Taxas de comunicação entre os núcleos de processamento

Com a finalidade de modelar a aplicação para realizar um posicionamento automático em uma NoC, foi calculada a taxa de comunicação entre os diferentes elementos do sistema. Na Tabela 1 encontra-se um cálculo do volume de dados, em bytes, transmitido em cada quadro de uma imagem de dimensões 640x 480 pixels.

Tabela 1: Volume de dados a ser transferido pela aplicação de segmentação de imagens, considerando uma imagem de 640x480 bytes e apenas um quadro.

PA ↔ ME		Número de bytes total da imagem (n_{BI}) = 640x480 = 307200 bytes
		Número de bytes de cada segmento (n_{BS}) = n_{BI} / n_{PE} = 307200 / 4 = 76800 bytes (considerar 2x – ida e volta)
PA ↔ PA	Em X	Número de bytes da imagem em Y (n_{BY}) = 480 bytes
		Número PAs em Y (n_{PAY}) = 2
		Número de bytes na fronteira de cada PA para Y (n_{BPY}) = n_{BY} / n_{PAY} = 480 / 2 = 240 bytes
	Em Y	Número de bytes da imagem em X (n_{BX}) = 640 bytes
		Número PAs em X (n_{PAX}) = 2
		Número de bytes na fronteira de cada PA para X (n_{BPX}) = n_{BX} / n_{PAX} = 640 / 2 = 320 bytes
PA ↔ PC		Número de bytes de controle de vizinhança (n_{CY}) = 128 bytes (estimado com base no tamanho dos objetos em imagens típicas, equivale a 64 pares)

Este valor deve ser dividido pelo produto do número de quadros por segundo pela fração do tempo que ocupa cada transferência. A soma do percentual de todas as transferências que ocorrem em série para cada quadro não pode ultrapassar (100-P)%, onde P é o percentual de tempo necessário para a realização da computação.

Grafo de comunicação da aplicação

Pode-se extrair um grafo da aplicação descrita considerando-se cada elemento do sistema como sendo um nodo e usando uma aresta para ligar elementos que sem comunicam. Adicionalmente, pode-se adicionar um valor à cada aresta do grafo, sendo que este valor corresponde ao volume de dados transferido entre os dois nodos.

Tal grafo, considerando-se um número arbitrário de processadores, pode ser observado na Figura 1. Não foi incluído nenhum valor de volume de dados transferido porque estes valores são particulares para uma dada configuração.

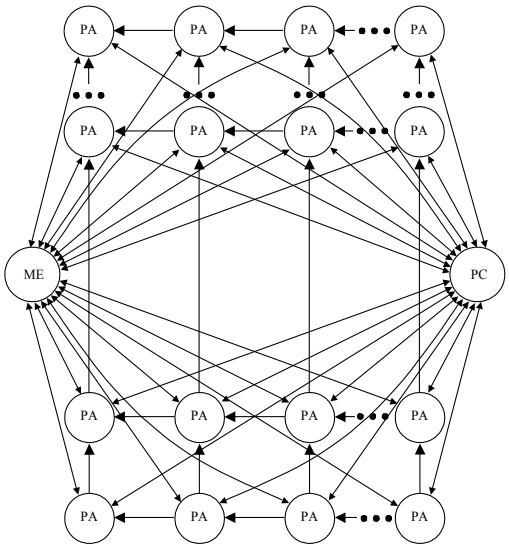


Figura 1: Grafo de comunicação genérico para o problema de segmentação de imagens, representando um número arbitrário de processadores.

Tarefas que são executadas em cada um dos elementos de processamento

Na Tabela 2 estão listados os passos que são executadas nos elementos do sistema.

Tabela 2: Passos executados nos elementos do sistema

Processamento da sub-imagem	Eliminação de redundâncias
P_1: Recebe imagem da ME P_2: Processa imagem gerando numeração local P_3: Se (tem P.E. vizinho esquerdo) Envia fronteira p/ P.E. esquerdo P_4: Se (tem P.E. vizinho acima) Envia fronteira para PE acima P_5:Se tem vizinho direito, recebe coluna P_6:Se tem vizinho abaixo, recebe linha P_7:processa fronteiras P_8: Envia cv para PC P_9: Recebe cv do PC P_{10}:Processa imagem P_{11}:Envia imagem para ME	P_1: Enquanto (não terminar todos Pas) Recebe cv de cada P.E. P_2: Elimina redundância dos objetos e mapeia numeração local para numeração global P_3:Envia nova numeração para os objetos Legenda: cv – controle de vizinhança

Exemplo

A aplicação SegImage está exemplificada aqui com uma imagem de 640x480, uma taxa de processamento de 15 quadros/seg, e 4 P.E.s. Assim, o número de pixels em X é 640 e o número de pixels em Y é 480. Como existem 2 P.E.s para cada dimensão, o número de bytes que cada P.E. deve processar na fronteira em X é 320 e o número de bytes que cada PA deve processar na fronteira em Y é 240 (sub-imagem de 320 x 240). As características da imagem e cada segmento são apresentadas na Figura2.

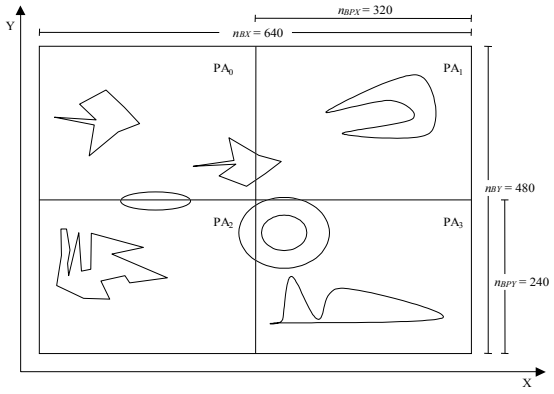


Figura 2: Segmentação da imagem avaliada com os correspondentes PAs.

Na Figura 3 estão apresentadas duas imagens e os objetos com a respectiva numeração. Pode-se observar na Figura 3(a) que em cada sub-imagem é produzida uma numeração independente, que é incrementada segundo a ordem dos topos dos objetos e da borda esquerda, como conseqüência do processo de varredura. A eliminação da redundância na numeração dos objetos é eliminada com auxílio do algoritmo descrito, resultando na numeração presente na Figura 3(b).

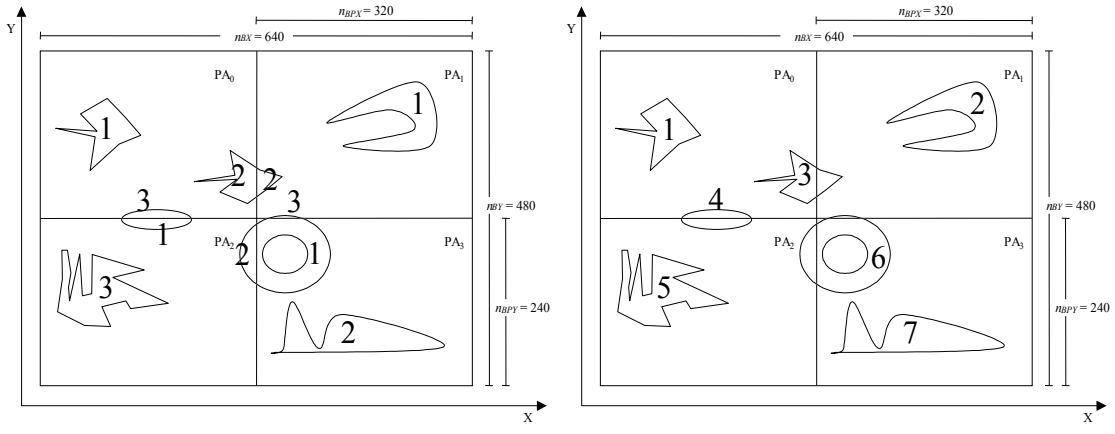


Figura 3: (a) Numeração inicial com redundância entre sub-imagens; (b) Numeração final após eliminação de redundâncias

Pode-se observar na Tabela 3 os diversos passos executados no algoritmo de eliminação de redundâncias.

Tabela 3: Passos para a eliminação de redundância dos objetos do exemplo da Figura 3. Na lista B, P é o objeto principal e S é o objeto secundário (mesmo objeto considerado como outro objeto). A Tabela 3(a) refere-se ao estado inicial das 2 listas; (b)-(f) são os passos intermediários e (g) é o estado final das duas listas.

Início			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	3	2h
2	0	3	1v
3	0		
1	1	2	1v
2	1		
1	2	1	1h
2	2		
1	3		
2	3		
3	3		

(a)

Passo 1			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	3	5
2	0	3	6
3	0		
4	1	5	8
5	1		
6	2	6	8
7	2		
8	3		
9	3		
10	3		

(b)

Passo 2			
Lista A		Lista B	
Obj.	P.E	P	S
1	0	-	-
2	0	3	6
3	0		
4	1	3	8
3	1		
6	2	6	8
7	2		
8	3		
9	3		

(c)

Passo 3			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	3	8
3	1		
3	2	3	8
7	2		
8	3		
9	3		
10	3		

(d)

Passo 4			
Lista A		Lista B	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-
3	1		
3	2	3	3
7	2		
3	3		
9	3		
10	3		

(e)

Passo 5			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-
3	1		
3	2	-	-
7	2		
3	3		
9	3		
10	3		

(f)

Fim			
Lista A		ListaB	
Obj.	P.E.	P	S
1	0	-	-
2	0	-	-
3	0		
4	1	-	-
3	1		
3	2	-	-
5	2		
3	3		
6	3		
7	3		

(g)

Na Tabela 3(a) encontra-se a lista A e a lista B, montadas de acordo com o método já descrito anteriormente. A lista A contém os objetos de cada sub-imagem, ordenados por objeto e por sub-imagem. Na Tabela 3-lista B, o símbolo P é utilizado para indicar o objeto principal e S é utilizado para indicar o objeto secundário, ou seja, o objeto principal que foi detectado como outro objeto na sub-imagem vizinha. Cada número em

S está acompanhado do símbolo h para indicar que o vizinho considerado é o da direção horizontal e v para indicar que é o vizinho da direção vertical. No passo 1 (Tabela 3(b)), a numeração local é substituída por uma numeração global, mantendo-se a redundância dos objetos mas corrigindo-se a coluna S da lista B. Nos passos 2-5, é realizada a substituição da numeração dos objetos, seguindo a ordem das linhas da lista B e substituindo-se em toda a lista A e no restante das linhas da lista B. Na Tabela 3(g) encontra-se o resultado final, obtido a partir da substituição da numeração da lista A da Tabela 3(f) por uma numeração uniforme.

Na Figura 4, pode-se observar o grafo de comunicação para o exemplo apresentado, para uma implementação em hardware do sistema. Neste caso, pode-se observar que a comunicação é unidirecional entre os P.E.s e bidirecional entre cada P.E e os outros elementos do sistema.

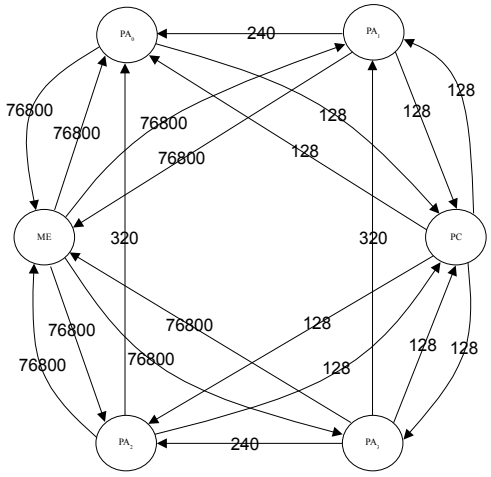


Figura 4: Grafo de comunicação do exemplo de segmentação de imagens.

Código Fonte dos Componentes

Elemento de Processamento

```
//Modificações no número de entradas e saídas afetam estas definições, os pinos
// de entrada, pinos de saída e a rotina de escrita nas saídas.
#define NUMMAXREGIOES 256 //numero maximo de regioes a serem computadas as areas

//consideracoes:
//0=vazio
//255=cheio
//outro valor=numero da regiao a qual pertence
enum {LESTE, NORTE, OESTE, SUL, FOI};
enum {RECMEMORIA, RECCOLUNA, RECLINHA, RECREGIOES, RECNUMERACAO, ENVMEMORIA};

class ProcCentral;
```

```
class ProcElem{
  public:
  unsigned int linha_memoria;
  unsigned int pos_x;
  unsigned int pos_y;
  unsigned int valor_anterior;
  unsigned int area;
  unsigned int areas_locais[NUMMAXREGIOES];//deve poder conter TAMX*TAMY
  unsigned char numero_do_processador;
  unsigned char idt_regiao;
  unsigned int numero_de_regioes;//tipo depende do numero maximo de regioes
  unsigned int numero_real_de_regioes;//tipo depende do numero maximo de regioes
  unsigned int * numeracao_de_saida;

  //--------------------
  unsigned char conteudo[TAMY][TAMX];//deve conter a imagem de entrada
  unsigned char conteudoM[TAMY][TAMX];//deve conter a imagem rotulada como saida
  unsigned char linha_superior_vizinho[TAMX];
  unsigned char linha_superior_vizinhoM[TAMX];
       unsigned   char   linha_superior[2*TAMX];//Armazena   intensidades(0~TAMX-1)   e
regioes(TAMX~2TAMX-1)
  unsigned char coluna_esquerda_vizinho[TAMY];
  unsigned char coluna_esquerda_vizinhoM[TAMY];
       unsigned   char   coluna_esquerda[2*TAMY];//Armazena   intensidades(0~TAMY-1)   e
regioes(TAMY~2TAMY-1)
  unsigned int linha_memoria_saida;
  bool linha_recebida;
  bool coluna_recebida;

  bool conteudo_visitado[TAMY][TAMX];
  bool direcao[TAMY][TAMX];

  //--------------------
  class pilha{
    public:
    unsigned int pilhaCX[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int pilhaCY[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int pilhaVA[TAMPILHA];//pode ser tipo char se a dimensao < 256
    unsigned int num_elem;
    pilha(){
      num_elem=0;
      }
    void EmpilhaCoordenadas(unsigned int x,unsigned int y,unsigned int va){
      pilhaCX[num_elem]=x;
      pilhaCY[num_elem]=y;
      pilhaVA[num_elem]=va;
```

```
    num_elem++;
    }
  void DesempilhaCoordenadas(unsigned int * x,unsigned int * y,unsigned int * va){
    num_elem--;
    (*x)=pilhaCX[num_elem];
    (*y)=pilhaCY[num_elem];
    (*va)=pilhaVA[num_elem];
    }
  }Pilha;
//--------------------
class pilhamod:pilha{
  public:
  void Adiciona(unsigned int a,unsigned int b){
    bool encontrou=false;
    for(unsigned int i=0;i<num_elem;i++)
      if((pilhaCX[i]==a)&&(pilhaCY[i]==b))encontrou=true;
    if(encontrou==false)EmpilhaCoordenadas(a,b,0);
    }
  unsigned int NumeroDeElementos(){
    return num_elem;
    }
  void CopiaRP(unsigned int * rp){//regiao primaria(passa a numeracao)
    for(unsigned int i=0;i<num_elem;i++)
      rp[i]=pilhaCX[i];
    return;
    }
  void CopiaRS(unsigned int * rs){//regiao secundaria(recebe a numeracao)
    for(unsigned int i=0;i<num_elem;i++)
      rs[i]=pilhaCY[i];
    return;
    }
  }InterseccoesH,InterseccoesV;
//--------------------
ProcElem()
  {
  Reset();
  }

~ProcElem(){
  if(numeracao_de_saida!=NULL)delete [] numeracao_de_saida;
  }

void Reset();
void Executa();
void VisitaElemento();
void PegaIdentificacao(unsigned char num);
```

```
   void RecebeMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned
char tipo);
   void RecebeMensagemW(unsigned int * conteudo_msg, unsigned int comprimento,unsigned
char tipo);
   void EnviaMensagem(unsigned char * conteudo_msg, unsigned int comprimento,unsigned
char tipo);
  void ProcessaFronteiras();
  void EnviaFronteiras();



  ProcElem * PEEsq;
  ProcElem * PESup;
  ProcCentral * PC;


};


#include "systemc.h"


#include "geral.h"
#include "ProcCentral.h"
#include "ProcElem.h"


//------------------------------------------------------------------------------
void ProcElem::Executa()
{
  numero_de_regioes=0;//tipo depende do numero maximo de regioes
  valor_anterior=1;
  idt_regiao=1;
  for(unsigned int posicao_y=0;posicao_y<TAMY;posicao_y++){
    for(unsigned int posicao_x=0;posicao_x<TAMX;posicao_x++){
      pos_x=posicao_x;
      pos_y=posicao_y;
      area=0;
      valor_anterior=conteudo[pos_y][pos_x];
      VisitaElemento();
      if(area!=0){
        numero_de_regioes++;
        idt_regiao++;
        }
      }
    }
  printf("PE no.%d encontrou %d ",numero_do_processador,numero_de_regioes);
  if(numero_de_regioes!=1)
    printf("regioes.\n");
  else
    printf("regiao.\n");
```

```
  //Comunica com vizinho superior
  bool toca_borda=false;
  for(int x=0;x<TAMX;x++){
    linha_superior[x]=conteudo[0][x];
    linha_superior[TAMX+x]=conteudoM[0][x];
    if(linha_superior[TAMX+x]!=0)toca_borda=true;
    }

  if(toca_borda==true){
    //envia mensagem
    if(PESup!=NULL){
      PESup->RecebeMensagem(linha_superior,2*TAMX,RECLINHA);
      }
    }

  //Comunica com o vizinho da esquerda
  toca_borda=false;
  for(int y=0;y<TAMY;y++){
    coluna_esquerda[y]=conteudo[y][0];
    coluna_esquerda[TAMY+y]=conteudoM[y][0];
    if(coluna_esquerda[TAMY+y]!=0)toca_borda=true;
    }

  if(toca_borda==true){
    //envia mensagem
    if(PEEsq!=NULL){
      PEEsq->RecebeMensagem(coluna_esquerda,2*TAMY,RECCOLUNA);
      }
    }
 }
//---------------------------------------------------------------------------
void ProcElem::VisitaElemento(){
    if((conteudo_visitado[pos_y][pos_x]==false)&&(conteudo[pos_y][pos_x]!=0)){
        conteudo_visitado[pos_y][pos_x]=true;//deve marcar aqui para os subniveis nao
repercorrerem o caminho
      conteudoM[pos_y][pos_x]=idt_regiao;
      if(pos_x==159)
        pos_x=pos_x;
      area++;
      if(pos_x>0){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_x-=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_y>0){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
```

```
        valor_anterior=conteudo[pos_y][pos_x];
        pos_y-=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_x<TAMX-1){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_x+=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      if(pos_y<TAMY-1){
        Pilha.EmpilhaCoordenadas(pos_x,pos_y,valor_anterior);
        valor_anterior=conteudo[pos_y][pos_x];
        pos_y+=1;VisitaElemento();
        Pilha.DesempilhaCoordenadas(&pos_x,&pos_y,&valor_anterior);
        }
      }
    }
//-------------------------------------------------------------------------------
void ProcElem::PegaIdentificacao(unsigned char numero){
  numero_do_processador=numero;
  }
//-------------------------------------------------------------------------------
void    ProcElem::RecebeMensagem(unsigned    char    *    conteudo_msg,    unsigned    int
comprimento,unsigned char tipo){
  if(tipo==RECMEMORIA){
    for(unsigned int x=0;x<comprimento;x++)
      conteudo[linha_memoria][x]=conteudo_msg[x];
    linha_memoria++;
    }
  if(tipo==RECLINHA){
    for(unsigned int x=0;x<comprimento/2;x++){
      linha_superior_vizinho[x]=conteudo_msg[x];
      linha_superior_vizinhoM[x]=conteudo_msg[x+comprimento/2];
      }
    linha_recebida=true;
    printf("Recebida linha no processador %d\n",numero_do_processador);
    }
  if(tipo==RECCOLUNA){
    for(unsigned int y=0;y<comprimento/2;y++){
      coluna_esquerda_vizinho[y]=conteudo_msg[y];
      coluna_esquerda_vizinhoM[y]=conteudo_msg[y+comprimento/2];
      }
    coluna_recebida=true;
    printf("Recebida coluna no processador %d\n",numero_do_processador);
    }
  }
```

```
//-------------------------------------------------------------------------------
void    ProcElem::RecebeMensagemW(unsigned    int    *    conteudo_msg,    unsigned    int
comprimento,unsigned char tipo){
  if(tipo==RECNUMERACAO){
    if(numeracao_de_saida!=NULL)delete [] numeracao_de_saida;
    numeracao_de_saida=new(unsigned int [comprimento+1] );
    numeracao_de_saida[0]=0;
    for(unsigned int i=1;i<=comprimento;i++){
      numeracao_de_saida[i]=conteudo_msg[i];
      }
    }
  printf("Recebida a numeracao de saida do processador %d\n",numero_do_processador);
  numero_real_de_regioes=conteudo_msg[comprimento+1];
  printf("Numero de regioes corrigido para %d\n",conteudo_msg[comprimento+1]);
  extern unsigned int total;
  total+=conteudo_msg[comprimento+1];
  }
//-------------------------------------------------------------------------------
void    ProcElem::EnviaMensagem(unsigned    char    *    conteudo_msg,    unsigned    int
comprimento,unsigned char tipo){
  if(tipo==ENVMEMORIA){
    for(unsigned int x=0;x<comprimento;x++)
                                                              conteudo_msg[x]=(unsigned
char)numeracao_de_saida[conteudoM[linha_memoria_saida][x]];
    linha_memoria_saida++;
//    numero_real_de_regioes=conteudo_msg[comprimento];
    }
  }
//-------------------------------------------------------------------------------
void ProcElem::Reset(){
  pos_x=0;
  pos_y=0;
  area=0;
  linha_memoria=0;
  linha_memoria_saida=0;
  linha_recebida=false;
  coluna_recebida=false;
  idt_regiao=1;
  PEEsq=NULL;
  PESup=NULL;
  numero_de_regioes=0;
  for(int x=0;x<TAMX;x++){
    linha_superior_vizinho[x]=0;
    linha_superior_vizinhoM[x]=0;
    linha_superior[x]=0;
    linha_superior[x+TAMX]=0;
    }
```

```
  for(int y=0;y<TAMY;y++){
    coluna_esquerda_vizinho[y]=0;
    coluna_esquerda_vizinhoM[y]=0;
    coluna_esquerda[y]=0;
    coluna_esquerda[y+TAMY]=0;
    }
  for(int y=0;y<TAMY;y++){
    for(int x=0;x<TAMX;x++){
      conteudo[y][x]=0;
      conteudoM[y][x]=0;
      conteudo_visitado[y][x]=false;//marca como nao visitado
      }
    }
  numeracao_de_saida=NULL;
  numero_real_de_regioes=0;
  valor_anterior=0;
  }
//---------------------------------------------------------------------------
void ProcElem::ProcessaFronteiras(){
  //Compara com base em 4 vizinhos
  printf("Processador %d comparando fronteiras\n");
  //Linha
  for(int x=0;x<TAMX;x++){
    if(abs(conteudo[TAMY-1][x]-linha_superior_vizinho[x])<TOLERANCIA)
      if((conteudoM[TAMY-1][x]==0)||(linha_superior_vizinhoM[x]==0))
        x=x;
      else
        InterseccoesV.Adiciona(conteudoM[TAMY-1][x],linha_superior_vizinhoM[x]);
    }
  //Coluna
  for(int y=0;y<TAMY;y++){
    if(abs(conteudo[y][TAMX-1]-coluna_esquerda_vizinho[y])<TOLERANCIA)
      if((conteudoM[y][TAMX-1]==0)||(coluna_esquerda_vizinhoM[y]==0))
        y=y;
      else
        InterseccoesH.Adiciona(conteudoM[y][TAMX-1],coluna_esquerda_vizinhoM[y]);
    }
                        printf("Processador           %d           encontrou           %d
intersecções.\n",numero_do_processador,InterseccoesH.NumeroDeElementos()+InterseccoesV.N
umeroDeElementos());
  }
//---------------------------------------------------------------------------
void ProcElem::EnviaFronteiras(){
  unsigned int * RPH,* RSH,* RPV,* RSV;
  RPH=new(unsigned int[InterseccoesH.NumeroDeElementos()]);
  RSH=new(unsigned int[InterseccoesH.NumeroDeElementos()]);
  RPV=new(unsigned int[InterseccoesV.NumeroDeElementos()]);
```

```
  RSV=new(unsigned int[InterseccoesV.NumeroDeElementos()]);
  InterseccoesH.CopiaRP(RPH);
  InterseccoesH.CopiaRS(RSH);
  InterseccoesV.CopiaRP(RPV);
  InterseccoesV.CopiaRS(RSV);
                                                                                PC-
>RecebeMensagem(RPH,RSH,InterseccoesH.NumeroDeElementos(),RPV,RSV,InterseccoesV.NumeroDe
Elementos(),
    numero_de_regioes,numero_do_processador);
  delete [] RPH;
  delete [] RSH;
  delete [] RPV;
  delete [] RSV;
  }
//-------------------------------------------------------------------------
```

Processador Central

```
//Modificações no número de entradas e saídas afetam estas definições, os pinos
// de entrada, pinos de saída e a rotina de escrita nas saídas.
#define NUMMAXREGIOES 1024 //numero maximo de regioes a serem computadas as areas

//consideracoes:
//0=vazio
//255=cheio
//outro valor=numero da regiao a qual pertence

class ProcElem;

class ProcCentral{
  class PE{
    public:
    unsigned int * pxH;
    unsigned int * pyH;
    unsigned int * pxV;
    unsigned int * pyV;
    unsigned int neph;
    unsigned int nepv;
    unsigned int netotal;
    unsigned int *regioes_old;
    unsigned int **regioes_new;
    unsigned char n_regioes;
      unsigned int menor;
      unsigned int indice;
      unsigned int dependencias[255];
```

```
PE(){
  regioes_old=NULL;
  regioes_new=NULL;
  pxH=NULL;    pyH=NULL;    pxV=NULL;    pyV=NULL;
  neph=0;      nepv=0;      netotal=0;
  n_regioes=0;
  }
~PE(){
  if(pxH!=NULL)delete[]pxH;
  if(pyH!=NULL)delete[]pyH;
  if(pxV!=NULL)delete[]pxV;
  if(pyV!=NULL)delete[]pyV;
  if(regioes_old!=NULL)delete[]regioes_old;
  for(int i=0;i<netotal;i++)
    if(regioes_new[i]!=NULL)delete[]regioes_new[i];
  if(regioes_new!=NULL)delete[]regioes_new;
  }
//Aloca espaço para a tabela de renomeação das regiões e outras informações
void Aloca(unsigned int h,unsigned int v,unsigned int total){
  pxH=new(unsigned int [h]);
  pyH=new(unsigned int [h]);
  pxV=new(unsigned int [v]);
  pyV=new(unsigned int [v]);
  netotal=total;
  neph=h;
  nepv=v;
  regioes_old=new(unsigned int [netotal]);
  regioes_new=new(unsigned int * [netotal]);
  for(int i=0;i<netotal;i++)
    regioes_new[i]=NULL;
  }
void MontaTabela(unsigned int * contador){
  for(unsigned int posicao=0;posicao<netotal;posicao++){
    regioes_new[posicao]=new(unsigned int [neph+nepv+1]);
    for(unsigned int i=0;i<neph+nepv+1;i++)
      regioes_new[posicao][i]=0;
    regioes_old[posicao]=posicao+1;
    regioes_new[posicao][0]=(* contador);
    printf("Regioes_new[%d]=%d\n",posicao,regioes_new[posicao][0]);
    (* contador)++;
    }
  }
void AdicionaDependenciasH(PE * PEE){
  for(unsigned int dependencia_h=0;dependencia_h<neph;dependencia_h++){
    unsigned int pos=1;
    while(regioes_new[pxH[dependencia_h]-1][pos]!=0)pos++;
```

```
            regioes_new[pxH[dependencia_h]-1][pos]=PEE->regioes_new[pyH[dependencia_h]-
1][0];
                                    printf("Regioes_newH[%d][%d]=%d\n",pxH[dependencia_h]-
1,pos,regioes_new[pxH[dependencia_h]-1][pos]);
        }
      }
    void AdicionaDependenciasV(PE * PEE){
      for(unsigned int dependencia_v=0;dependencia_v<nepv;dependencia_v++){
        unsigned int pos=1;
        while(regioes_new[pxV[dependencia_v]-1][pos]!=0)pos++;
            regioes_new[pxV[dependencia_v]-1][pos]=PEE->regioes_new[pyV[dependencia_v]-
1][0];
                                    printf("Regioes_newV[%d][%d]=%d\n",pxV[dependencia_v]-
1,pos,regioes_new[pxV[dependencia_v]-1][pos]);
        }
      }
     void SubstituiDependencias(unsigned int *lista_de_dependencias,unsigned int n,
unsigned int novo_valor){
      for(unsigned int linha=0;linha<netotal;linha++){
        for(unsigned int pos_linha=0;pos_linha<neph+nepv+1;pos_linha++){
          if(regioes_new[linha][pos_linha]==0)break;
          for(unsigned int pos_lista=0;pos_lista<n;pos_lista++){
            if(regioes_new[linha][pos_linha]==lista_de_dependencias[pos_lista])
              regioes_new[linha][pos_linha]=novo_valor;
            }
          }
        }
      }
    void ProcessaDependencias(unsigned int posicao){
        menor=32767;
        for(indice=0;indice<neph+nepv+1;indice++){
          dependencias[indice]=regioes_new[posicao][indice];
          if(regioes_new[posicao][indice]==0)break;
          if(regioes_new[posicao][indice]<menor)menor=regioes_new[posicao][indice];
          if(indice>0)regioes_new[posicao][indice]=0;
          }
        if((menor!=32767)&&(indice!=1)){
          printf("Menor:%d\n",menor);
          }
      }
     void Copia(unsigned int * pCXH, unsigned int * pCYH,unsigned int * pCXV, unsigned
int * pCYV){
      for(unsigned int i=0;i<neph;i++){
        pxH[i]=pCXH[i];
        pyH[i]=pCYH[i];
        printf("(%d,%d)h\n",pxH[i],pyH[i]);
        }
      for(unsigned int i=0;i<nepv;i++){
        pxV[i]=pCXV[i];
```

```
        pyV[i]=pCYV[i];
        printf("(%d,%d)v\n",pxV[i],pyV[i]);
        }
      }
    void MontaTabelaDeSaida(unsigned int * TabelaDeSaida,unsigned int * tabela, unsigned
int *contador){
      for(unsigned int posicao=0;posicao<netotal;posicao++){
        if(TabelaDeSaida[regioes_new[posicao][0]]==0){
          TabelaDeSaida[regioes_new[posicao][0]]=(* contador);
           printf("TabelaDeSaida[%d]=%d\n",regioes_new[posicao][0],TabelaDeSaida[regioes_
new[posicao][0]]);
          (*contador)++;
          n_regioes++;
          }
        }
      tabela[0]=0;
      for(unsigned int i=0;i<netotal;i++){
        tabela[i+1]=TabelaDeSaida[regioes_new[i][0]];
        printf("Tabela[%d]=%d\n",i+1,tabela[i+1]);
        }
      }
    }PELocal[NPESY][NPESX];//PELocal1,PELocal2,PELocal3,PELocal4;

  public:
  ProcCentral(){
    Reset();
    }

  ~ProcCentral(){
    }

  void Reset();
  void RecebeMensagem(
    unsigned int * pCXH, unsigned int * pCYH, unsigned int num_elem_h,
    unsigned int * pCXV, unsigned int * pCYV,unsigned int num_elem_v,
    unsigned int total,int num_pe);
  void ProcessaRegioes();
  ProcElem * PEReal[NPESY][NPESX];
  bool PEEnviou[NPESY][NPESX];

};

#include "geral.h"
#include "ProcElem.h"
#include "ProcCentral.h"

//---------------------------------------------------------------------------
```

```
void ProcCentral::RecebeMensagem(
  unsigned int * pCXH, unsigned int * pCYH, unsigned int num_elem_h,
  unsigned int * pCXV, unsigned int * pCYV,unsigned int num_elem_v,
  unsigned int total,int num_pe){

  //Recebe dados do elemento de processamento num_pe
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      if(PEReal[y][x]->numero_do_processador==num_pe){
        PELocal[y][x].Aloca(num_elem_h,num_elem_v,total);
        PELocal[y][x].Copia(pCXH,pCYH,pCXV,pCYV);
        }
  printf("Recebida mensagem do PE%d\n",num_pe);
  }
//---------------------------------------------------------------------------
void ProcCentral::Reset(){
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      PEEnviou[y][x]=false;
      PEReal[y][x]=NULL;
      }
  }
//---------------------------------------------------------------------------
void ProcCentral::ProcessaRegioes(){
  unsigned int contador=1;
  printf("Montando tabela\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      PELocal[y][x].MontaTabela(&contador);
  printf("Adicionando dependencias\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX-1;x++)
      PELocal[y][x].AdicionaDependenciasH(&PELocal[y][x+1]);
  for(int y=0;y<NPESY-1;y++)
    for(int x=0;x<NPESX;x++)
      PELocal[y][x].AdicionaDependenciasV(&PELocal[y+1][x]);
  printf("Processando e substituindo dependencias\n");
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      for(unsigned int posicao=0;posicao<PELocal[y][x].netotal;posicao++){

      PELocal[y][x].ProcessaDependencias(posicao);
      for(int yp=0;yp<NPESY;yp++)
        for(int xp=0;xp<NPESX;xp++)
          PELocal[yp][xp].SubstituiDependencias(PELocal[y][x].dependencias,PELocal[y][x]
.indice,PELocal[y][x].menor);
        }
```

```
      }
  printf("Montando tabelas de saida\n");
  unsigned int * TabelaDeSaida;
  unsigned int tamanho=1;//necessario
  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++)
      tamanho+=PELocal[y][x].netotal;
  TabelaDeSaida=new(unsigned int [tamanho]);

  for(unsigned int i=0;i<tamanho;i++)
    TabelaDeSaida[i]=0;

  unsigned int * TabelaDoProcessador;
  contador=1;

  for(int y=0;y<NPESY;y++)
    for(int x=0;x<NPESX;x++){
      TabelaDoProcessador=new(unsigned int [PELocal[y][x].netotal+2]);
      PELocal[y][x].MontaTabelaDeSaida(TabelaDeSaida,TabelaDoProcessador,&contador);
           TabelaDoProcessador[PELocal[y][x].netotal+1]=PELocal[y][x].n_regioes;//ultimo
elemento=numero real de regioes
                                                                          PEReal[y][x]-
>RecebeMensagemW(TabelaDoProcessador,PELocal[y][x].netotal,RECNUMERACAO);
      delete [] TabelaDoProcessador;
      }

  //devolve numeracao para processadores
  delete[]TabelaDeSaida;
  }
//-------------------------------------------------------------------------------
```