Co-projeto de hardware/software para correlação de imagens

Maurício Acconcia Dias

Co-projeto de hardware/software para correlação de imagens

Maurício Acconcia Dias

Orientador: **Prof. Dr. Fernando Santos Osório**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Agosto de 2011

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Dias, Maurício Acconcia
D541c Co-Projeto de hardware/software para correlação de imagens / Maurício Acconcia Dias; orientador Fernando Santos Osorio -- São Carlos, 2011.
86 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2011.

1. Co-projeto de hardware/software. 2. Processamento de imagens digitais. 3. Correlação cruzada normalizada. 4. Soft-processor. I. Osorio, Fernando Santos, orient. II. Título.

Agradecimentos

Primeiramente gostaria de salientar que gostaria muito de agradecer a todas as pessoas que realmente me ajudaram de forma direta e indireta na realização deste trabalho porém o espaço é curto neste documento. Gostaria também de dizer que se fosse possível, parágrafos estariam um ao lado do outro pois a importância das pessoas não deve ser medida pela ordem em que foram citadas e sim pelo que me fizeram sentir ao prestar o auxílio quando mais precisei.

Agradeço em caráter prioritário à minha família pelo apoio incondicional: minha mãe Patricia e meu pai Antonio Carlos que em momentos de dificuldade sempre estiveram ao meu lado e sempre deram o suporte necessário em todos os sentidos para alcançar este objetivo; agradeço aos meus avós Rafael e Lourdes que sempre me recebem com muito carinho e compreensão ainda que eu estivesse ausente por tempos devido ao trabalho; agradeço à minha tia Fátima e ao meu primo Rafael pelo apoio e principalmente pelas conversas que sempre são ótimas e claro também pela compreensão em relação à ausência; minha prima Naina pelas visitas, por me tirar um pouco do mundo da pesquisa e me levar pra sair a noite ou tomar um café pela manhã e também meus tios Paulo,Célia e Clarice que me recebem sempre tão bem e que, ainda mais que as pessoas anteriormente citadas, precisaram e compreenderam minha ausência. A família sempre é nosso porto seguro.

Agradeço também à minha namorada Larissa que logo no começo do namoro me encontrou em situação de extrema correria e *stress* com o final do trabalho e mesmo assim foi compreensiva e me aguentou, e nunca reclamou de ter que deixar de me ver algum dia por causa do trabalho. Agradeço também à mãe dela Aparecida e à irmã Rebecca que juntamente com ela preencheram um vazio que há algum tempo eu tinha na minha vida ainda que cronologicamente estejam presentes a pouco tempo nela. Obrigado mesmo pelo carinho, compreensão e amor se tornando em pouco tempo parte integrante da minha família e me acolhendo da mesma forma.

Um agradecimento especial ao meu Orientador Fernando Osório que sempre esteve ao meu lado extremamente disposto a ajudar, sempre cobrou de maneira justa e correta e sempre foi compreensivo quando atingi meu limite e precisei me ausentar. Todas as conversas, viagens, congressos e experiências serão levadas para a vida toda e, mais que isso, quando entrei no mestrado ganhei não só um orientador mas também um amigo. Ao Denis e aos outros professores e alunos do laboratório meu agradecimento pelo apoio em momentos difíceis e também pelos momentos de diversão.

Dizem que amigos são a família que escolhemos portanto o segundo agradecimento especial para eles: agradeço inicialmente ao Bruno que divide o apartamento comigo desde o início do trabalho e suportava conversas, nem sempre em momentos oportunos, com paciência e sempre foi companheiro para estudos, artigos, almoços, jantares, drinks e principalmente as conversas intermináveis que foram previstas ainda no tempo de faculdade. São seis anos e meio de amizade que espero que durem muito mais que isso. Agradeço ao Daniel, outro remanescente dos tempos de graduação, que até hoje se mostra uma pessoa excelente e também me aguentou muito no 1,5 ano que está por aqui e que é igualmente companheiro para estudos, artigos, jogos de tenis, conversas e tudo mais. Agradeço também à Lilian, ao Raphael, ao Pessin e a todos os outros amigos, vocês sabem quem são, que realmente formam a família que escolhi.

O desenvolvimento desta dissertação de mestrado não seria possível sem algumas contribuições. Agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro durante o período de desenvolvimento do trabalho. Agradeço também ao INCT-SEC/LRM (Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos / Laboratório de Robótica Móvel - Grupo SEER - USP) por disponibilizarem equipamentos e infra-estrutura necessárias para que este projeto fosse desenvolvido.

Resumo

Este trabalho de pesquisa tem por objetivo o desenvolvimento de um co-projeto de hardware/software para o algoritmo de correlação de imagens visando atingir um ganho de desempenho com relação à implementação totalmente em software. O trabalho apresenta um comparativo entre um conjunto bastante amplo e significativo de configurações diferentes do *soft-processor* Nios II implementadas em FPGA, inclusive com a adição de novas instruções dedicadas. O desenvolvimento do co-projeto foi feito com base em uma modificação do método baseado em *profiling* adicionando-se um ciclo de desenvolvimento e de otimização de software. A comparação foi feita com relação ao tempo de execução para medir o *speedup* alcançado durante o desenvolvimento do co-projeto que atingiu um ganho de desempenho significativo. Também analisou-se a influência de estruturas de hardware básicas e dedicadas no tempo de execução final do algoritmo. A análise dos resultados sugere que o método se mostrou eficiente considerando o *speedup* atingido, porém o tempo total de execução ainda ficou acima do esperado, considerando-se a necessidade de execução e processamento de imagens em tempo real dos sistemas de navegação robótica. No entanto, destaca-se que as limitações de processamento em tempo real estão também ligadas as restrições de desempenho impostas pelo hardware adotado no projeto, baseado em uma FPGA de baixo custo e capacidade média.

Abstract

This work presents a FPGA based hardware/software co-design for image normalized cross correlation algorithm. The main goal is to achieve a significant speedup related to the execution time of the all-software implementation. The co-design proposed method is a modified profiling-based method with a software development step. The executions were compared related to execution time resulting on a significant speedup. To achieve this speedup a comparison between 21 different configurations of Nios II soft-processor was done. Also hardware influence on execution time was evaluated to know how simple hardware structures and specific hardware structures influence algorithm final execution time. Result analysis suggest that the method is very efficient considering achieved speedup but the final execution time still remains higher, considering the need for real time image processing on robotic navigation systems. However, the limitations for real time processing are a consequence of the hardware adopted in this work, based on a low cost and capacity FPGA.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviatuas

/e *Nios II processor model Economic*

/s *Nios II processor model Standard*

/f *Nios II processor model Fast*

ASIC *Application-Specific Integrated Circuit*

ASIP *Application-Specific Instruction-Set Processor*

DSP *Domain Specific Processor (most used)*

DSP *Digital Signal Processing (alternative)*

FNCC *Fast Normalized Cross-Correlation*

FPGA *Field Programmable Gate Array*

FPU *Floating Point Unit*

GCC *GNU Compiler Collection*

GPP *General Purpose Processor*

GPROF *GNU Profiler*

HDL *Hardware Description Language*

I/O *Input/Output*

ICMC Instituto de Ciências Matemáticas e de Computação

IDE *Integrated Development Environment*

LED *Light Emitting Diode*

LIDAR/LADAR *Light Detection And Ranging*

LRM Laboratório de Robótica Móvel do ICMC-USP

NCC *Normalized Cross-Correlation*

Nios II *Altera Soft-Processor (FPGA)*

PLL *Phased-Locked Loop*

PPG Processador de Propósito Geral

RTL *Register Transfer Level*

RTOS *Real-Time Operating System*

SDRAM *Synchronous Dynamic Random Access Memory*

SEER Grupo de Pesquisas em Sistemas Embarcados Evolutivos e Robóticos do ICMC-USP

STOC *Stereo On A Chip*

SOC *System On a Chip*

SOPC *System On a Programmable Chip*

SSRAM *Synchronous Static Random Access Memory*

ULA Unidade Aritmética-Lógica

USP Universidade de São Paulo

VHDL *Very High Speed Integrated Circuit Hardware Description Language*

CAPÍTULO

1

Introdução

Sistemas computacionais estão cada vez mais presentes no cotidiano das pessoas. O processamento rápido e eficiente de informação se tornou uma necessidade atualmente para estes sistemas induzindo o desenvolvimento de sistemas específicos para a execução de tarefas substituindo os sistemas de propósito geral. Estes sistemas específicos são, na maioria das vezes, produzidos em larga escala preocupando-se com requisitos severos de desempenho e custo. Os sistemas de computação desenvolvidos para um propósito específico segundo algumas restrições sendo as principais: tempo (projeto e execução), custo e área ocupada são chamados de sistemas embarcados .Atualmente, os processadores embarcados representam a maior parte do total de vendas no mercado de processadores.

Este trabalho de pesquisa visa o desenvolvimento de um co-projeto de hardware/software para o algoritmo de correlação de imagens visando atingir um ganho de desempenho com relação a implementação totalmente em software e para sistemas de propósito geral do algoritmo. O desenvolvimento do co-projeto utiliza um método baseado em *profiling*. As ferramentas escolhidas são os FPGAs e um *soft-processor* que permite a inclusão de componentes de hardware, através da implementação de instruções customizadas.

Esta dissertação de mestrado foi desenvolvida dentro do contexto do Laboratório de Robótica Móvel (LRM) da Universidade de São Paulo (USP). O laboratório e sua equipe de desenvolvimento estão inseridos no grupo de pesquisa de Sistemas Embarcados, Evolutivos e Robóticos (SEER) (que possui ampla experiência nas áreas de desenvolvimento de hardware e robótica) e também contribui com o INCT-SEC (Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos) através de seus projetos de pesquisa.

1.1 Motivação

O mercado de sistemas computacionais esta forçando o desenvolvimento de novos sistemas com melhor desempenho e mais funcionalidades em um curto período de tempo. Devido às novas funcionalidades, a complexidade dos SoCs (*System-on-Chips*) aumenta juntamente com o número de unidades programáveis contidas nestes chips.

Como previsto por Gordon Moore (Moore (1965)), o número máximo de transistores possíveis dentro de um chip aumenta de forma considerável com o passar do tempo e atualmente os SoCs possuem bilhões de transistores. Este número de transistores permite que os dispositivos sejam aplicáveis a várias situações e com um grande número de funcionalidades.

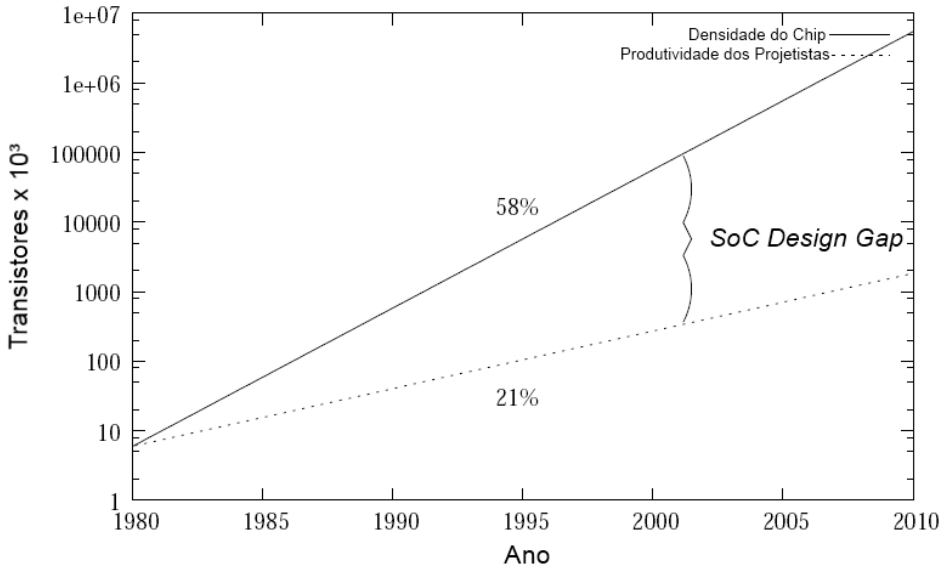

Figura 1.1: SoC Design Gap. Adaptado de Semiconductor Research Corporation.

Apesar deste fato, este número alto de transistores no *chip* causa um grande problema que é a impossibilidade dos projetistas de lidar com o crescimento da complexidade lógica destes chips durante o desenvolvimento de sistemas. Este problema é atenuado com as ferramentas de *design* porém não é completamente resolvido. A figura 1.1 ilustra o fenômeno que é conhecido como lacuna de projetos de SoCs ou *The SoC design gap*[1] (Henkel (2003)).

O problema apresentado pode ser atenuado de forma mais significativa com novas abordagens e métodos de desenvolvimento utilizando SoCs complexos. A utilização do SoC desenvolvido para várias aplicações diminui o custo de desenvolvimento final. O desenvolvimento de SoCs complexos gerou uma demanda crescente por ferramentas e métodos que facilitem o processo

[1] http://www.src.org/#

de otimização dos sistemas com relação a tempo de desenvolvimento, custo, área e consumo de energia.

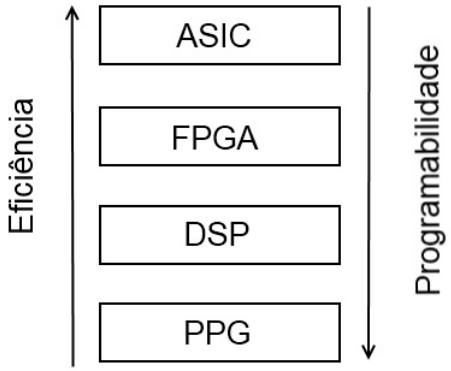

Figura 1.2: Eficiência X Programabilidade.

O desenvolvimento de SoCs para sistemas embarcados possui dois requisitos importantes e conflitantes: Eficiência (atender requisitos como tempo de execução, área ocupada, consumo de energia) e Programabilidade (permitir modificações em sua estrutura e/ou funcionalidade). Existem vários dispositivos que podem ser utilizados para atender estes requisitos porém cada um possui características que não os atendem de forma conjunta. A figura 1.2 mostra a relação entre os requisitos e as principais formas de implementação de sistemas embarcados: ASICs (*Application Specific Integrated Circuit*), DSPs (*Domain Specific Processors*), FPGAs (*Field Programmable Gate Arrays*) e PPG (Processadores de Propósito Geral). As características que colocam os dispositivos nesta ordem serão discutidas posteriormente, a princípio é importante salientar que os FPGAs combinam de uma forma interessante os dois requisitos comparados.

A crescente complexidade de sistemas embarcados resultou na busca por métodos e soluções de projetos mais rápidas e eficientes. A utilização de microprocessadores juntamente com hardware customizado nos projetos de sistemas embarcados se mostrou uma alternativa interessante por possibilitar a integração de hardware e software de uma forma simples e eficiente.

Existem processadores que possuem a opção de incluir hardware customizado como instruções adicionais ao conjunto de instruções básicas do processador, são os *custom-instruction processors* ou processadores de instruções customizadas. Este tipo de processador permite que execuções de funções sejam aceleradas com o desenvolvimento de hardware apenas para execução da função e sua inclusão como uma nova instrução. A figura 1.3 ilustra a situação. Utilizando esta plataforma de desenvolvimento os requisitos de projeto podem ser atingidos de forma mais simples e rápida. Processadores deste tipo estão sendo considerados blocos básicos de desenvolvimento de sistemas embarcados complexos atualmente.

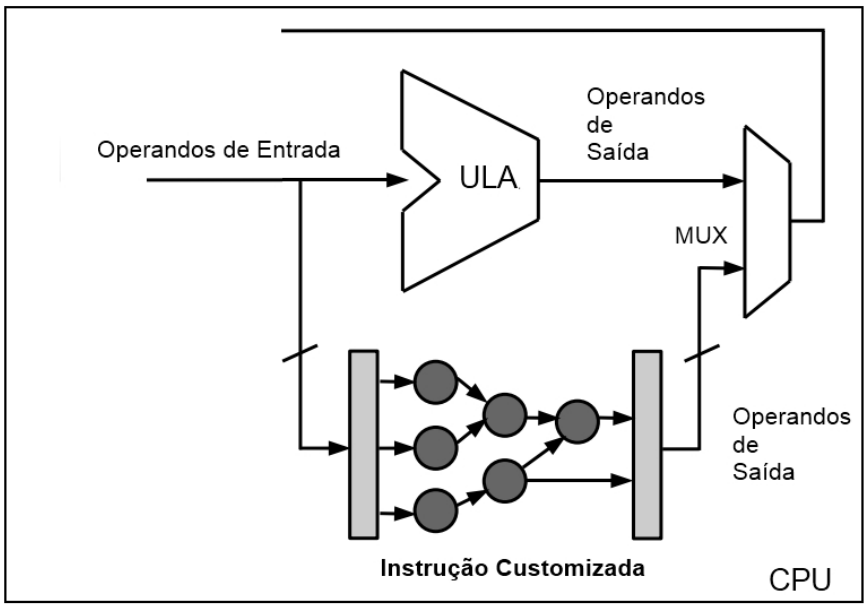

Figura 1.3: Instruções Customizadas. Adaptado de Atasu et al. (2008).

Considerando o contexto de desenvolvimento desta dissertação, o projeto de sistemas embarcados está diretamente ligado à robótica móvel. A robótica é uma área de pesquisa em destaque atualmente devido, dentre outros fatores, ao crescente interesse em automatizar tarefas para que sejam executadas de forma cada vez mais rápida e segura. Um exemplo claro deste interesse é o fato de carros saírem de fábrica com um número cada vez maior de sistemas embarcados que auxiliam o motorista em sua tarefa de dirigir, aumentando a segurança e buscando inclusive um estágio de desenvolvimento em que o carro seja completamente autônomo, ou seja, não necessite de um motorista.

Os sistemas de navegação robótica baseados em visão computacional tem despertado o interesse de pesquisadores durante muito tempo (DeSouza and Kak (2002)). Estes sistemas utilizam sensores visuais (câmeras) e a partir das imagens obtidas extraem as informações relevantes para que possa ser feita a navegação. Existem vantagens e desvantagens com relação à utilização de imagens para a navegação.

Dentre as vantagens de sua utilização pode-se citar: as imagens contém um grande volume de informações sobre o ambiente que será navegado com alto nível de detalhes; imagens relacionam as medições feitas no ambiente com sua estrutura (Dudek and Jenkin (2000)); os sensores visuais possuem um longo alcance e custo cada vez menor e o custo computacional para navegação por imagens comparado ao custo computacional de outros tipos de algoritmos de criação e localização em mapas que dependem de sensores mais sofisticados (e.g. LIDAR) é em geral bem mais baixo.

As principais desvantagens são: o custo computacional dos algoritmos de processamento de imagens utilizados, que mesmo baixo em comparação a

outros algoritmos, é alto; a capacidade de armazenamento necessária para armazenar as imagens; o custo das transferências de dados; fatores como iluminação e transformações nas imagens (rotações, translações, escala) que devem ser tratadas.

O volume de dados obtidos pelos sensores visuais tem aumentado consideravelmente nos últimos anos e juntamente com este fato a necessidade de sistemas que possam processar estas informações mais rapidamente faz com que seja necessário um hardware especializado para atingir a velocidade de processamento desejada.

O desenvolvimento de sistemas embarcados que executem os algoritmos de processamento de imagens é necessário porque existe uma crescente exigência de desempenho sobre estes algoritmos devido ao aumento de sua complexidade e da qualidade das imagens obtidas (aumento de resolução e número de cores) (Russ (2002)). Outra questão importante com relação ao hardware dedicado é o consumo de energia que deve ser eficiente mesmo com o possível aumento que pode acontecer em decorrência de uma maior capacidade de processamento visando um melhor desempenho (Chen et al. (2009)).

Portanto considerando a situação atual de desenvolvimento de sistemas embarcados, suas características e restrições, constata-se a importância e necessidade de desenvolvimento eficiente de sistemas com alta capacidade de processamento, baixo consumo de energia e voltados para uma aplicação inserida diretamente no contexto de desenvolvimento do trabalho: o processamento de imagens. Além disto, existe uma grande demanda por parte da robótica relacionada ao desenvolvimento de sistemas embarcados dotados da capacidade de processamento de imagens, como por exemplo, os robôs móveis e veículos autônomos.

1.1.1 Aplicações em Robótica

O problema do controle e da navegação de robôs com base móvel é ilustrado pela figura 1.4 que mostra um mapa do tipo grade de ocupação (*occupation grid*) juntamente com um robô que irá navegar de um ponto A até um ponto B e os obstáculos. O robô deve usualmente realizar o seu deslocamento de uma posição a outra, seguindo uma trajetória especificada e evitando colidir com os obstáculos. Inicialmente a robótica limitava-se a desenvolver robôs de base fixa para cumprir tarefas específicas mesmo que complexas. A evolução da tecnologia ao longo do tempo despertou a necessidade de locomoção dos robôs iniciando assim o desenvolvimento de robôs com bases móveis.

Dentre as várias maneiras de se navegar através do uso de visão computacional sem possuir um mapa do ambiente estão os *Appearance-Based Methods*. Este tipo de método utiliza o conceito de memorização do ambiente cuja idéia

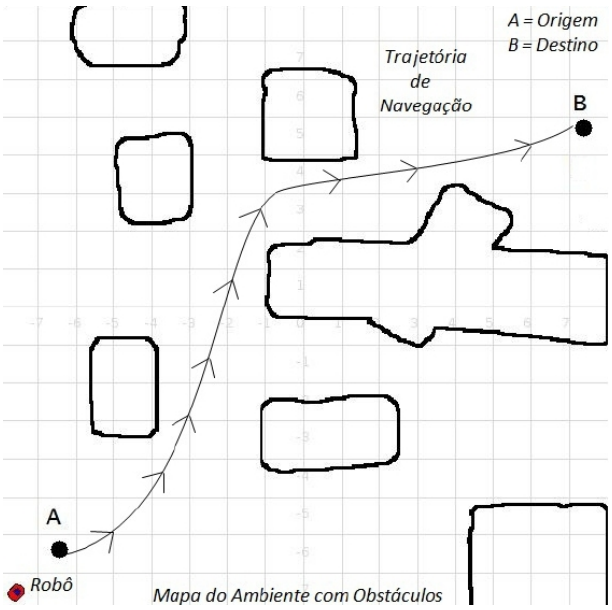

Figura 1.4: Exemplo do problema da navegação robótica

é armazenar imagens ou *templates* do ambiente (memória fotográfica) e associar estas imagens e *templates* a comandos ou controles que irão levar o robô a seu destino final. As características podem ser extraídas da imagem ou encontradas através de algoritmos convencionais ou técnicas de aprendizado de máquina e inteligência computacional. Um dos algoritmos mais utilizados para encontrar a imagem correspondente à imagem atual do robô e identificar qual comando ele deve executar é o cálculo do coeficiente de correlação cruzada normalizado (Gonzalez and Woods (2006)).

Existem alguns trabalhos que servem como referência de utilização deste tipo de técnica na área de navegação robótica: Gaussier (Joulain et al. (1997)) desenvolveu em seu trabalho uma rede neural que analisava as imagens 270-graus (composta por uma junção de várias imagens em sequência captadas por uma câmera em um dispositivo de giro) feitas pela câmera do robô e tomava a decisão de movimento; Matsumoto (Matsumoto et al. (1996)) usou uma técnica de verificação por templates para reconhecer ambientes através de uma sequência de imagens e criou o conceito de VSRR (View-Sequenced Route Representation); Jones (Jones et al. (1997)) seguindo o mesmo conceito de VSRR criou um sistema que o robô recebe as imagens e ações associadas. Jones utilizou a correlação cruzada (NCC - *Normalized Cross Correlation*) onde o robô recupera a imagem do banco de dados que mais parece com a imagem do local e executa a ação relacionada.

A figura 1.5 mostra um esquema de como funciona o método. Após inicialmente ser guiado e armazenar o trajeto em forma de imagens (Memorização da Trajetória), o robô se encontra em uma posição e verifica em sua memória visual qual imagem é mais semelhante a imagem que ele obtém da câmera

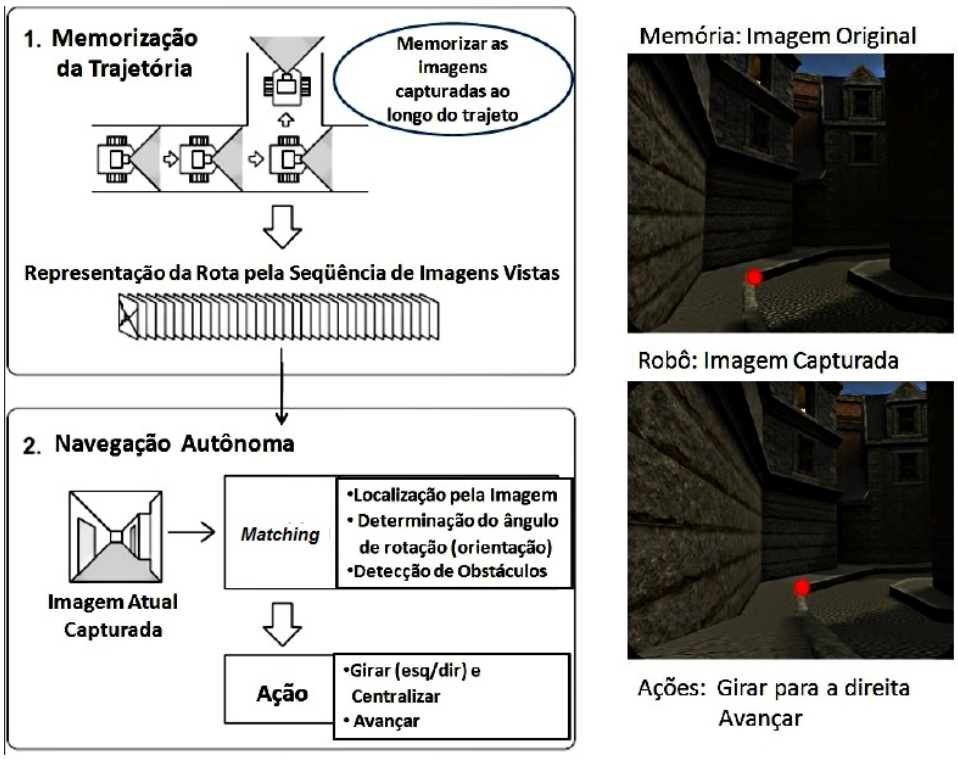

Figura 1.5: Navegação Visual. Adaptado de Matsumoto et al. (1999).

utilizando algum método de *template matching*, como por exemplo, correlação de imagens. Ao selecionar a imagem, o robô ajusta sua orientação e sua posição para que possa se alinhar de forma correta e executar a ação associada a imagem encontrada.

Figura 1.6: Plataformas Robóticas LRM-USP.

Este método de navegação através de câmeras pode ser aplicado diretamente nos robôs móveis do Laboratório de Robótica Móvel (LRM), que atualmente dispõe de robôs para navegação *indoor* e *outdoor* e inclusive carros autônomos como mostra a figura 1.6. Esta é uma importante motivação deste trabalho, visto que com o uso de um hardware customizado e configurado de modo a executar o processamento de imagens de forma mais rápida, os computadores embarcados podem ter seu poder de processamento focado no tratamento de outras importantes tarefas do processo de navegação autônoma (e.g. planejamento de rotas, mapeamento do ambiente, desvio de obstáculos).

1.2 Contribuições do Trabalho

Este projeto de pesquisa enquadra-se nas áreas de desenvolvimento de hardware e de visão computacional. O objetivo principal deste projeto é desenvolver um co-projeto de hardware/software utilizando um *soft-processor* e instruções customizadas visando melhorar o *speedup* da execução do algoritmo em relação ao algoritmo puramente seqüencial. Após a implementação o resultado será avaliado com relação a viabilidade de compor um sistema robótico de navegação visual.

O desenvolvimento do trabalho trouxe principalmente as seguintes contribuições:

- Desenvolvimento de um co-projeto de hardware/software que apresentou um *speedup* de 33,92 comparado com o desempenho do algoritmo de referência puramente sequencial.

- Análise da implementação do algoritmo de correlação cruzada normalizada de imagens no *soft-processor* Nios II com relação as seguintes variáveis: configurações do *soft-processor* Nios II (*economic, standard e fast*), relação de tamanhos de imagem e sub-imagem com o tempo de execução do algoritmo no *soft-processor*, comparação de *profiling* dos algoritmos com relação à gargalos no tempo de execução, utilização de memória *on-chip* e RAM.

- Comparação de dois FPGAs da fabricante Altera (Cyclone II e Cyclone III) em duas plataformas diferentes (respectivamente Terasic DE2-70 e Arrow BeMicro) para a implementação do algoritmo.

- Influência de técnicas de otimização de software para sistemas embarcados no tempo de execução do software e também na usabilidade do código.

- Modificação na metodologia básica de co-projeto de hardware/software baseada em *profiling* para sistemas embarcados fazendo com que o particionamento seja feito de forma ainda mais natural durante o desenvolvimento introduzindo uma divisão por ciclos.

1.3 Organização dos Capítulos

Os capítulos deste documento estão organizados da seguinte maneira: o capítulo 2 descreve a computação reconfigurável, o funcionamento da principal ferramenta utilizada para sua implementação atualmente, os FPGA's (*Field Programmable Gate Arrays*), juntamente com seu fluxo de design e também o conceito de *soft-processors*; o capítulo 3 apresenta conceitos de sistemas embarcados e co-projeto de hardware/software para o desenvolvimento destes sistemas; os tópicos de processamento de imagens e sua conexão com a robótica são tratados no capítulo 4 seguido por uma apresentação das ferramentas que foram utilizadas no desenvolvimento do trabalho no capítulo 5; o capítulo 6 apresenta o desenvolvimento do trabalho proposto nesta dissertação; o capítulo 7 apresenta os resultados e análises, encerrando com as conclusões e trabalhos futuros no capítulo 8. Ao final do documento encontram-se as referências bibliográficas.

CAPÍTULO

2

Computação Reconfigurável

2.1 Definição

Existem basicamente duas formas de se executar um algoritmo, por hardware ou por software (Hauck and Dehon (2007)). No caso do hardware existem os ASICs (*Application Specific Integrated Circuit*) que são projetados para uma aplicação específica e também os DSPs (*Domain Specific Processors*) que são processadores projetados para um domínio específico sendo mais flexíveis em comparação com os ASICs. Apesar de serem específicos e executarem a tarefa para a qual foram projetados de forma eficiente, estes circuitos não podem ser alterados após sua fabricação. Os microprocessadores, que seriam responsáveis pela execução do algoritmo por software, são mais flexíveis na medida em que o software pode ser alterado e outra tarefa pode ser executada sem mudança no hardware. O problema dos microprocessadores é que para que esta flexibilidade seja atingida o desempenho na execução do algoritmo fica geralmente muito abaixo do desempenho de um ASIC (Compton et al. (2000)).

A computação reconfigurável se encontra em um nível entre as formas citadas acima para a execução de um algoritmo como mostrado na figura 2.1, combinando-as e possibilitando assim eliminar grande parte das desvantagens associadas a cada uma das formas de desenvolvimento. Este tipo de computação baseia-se em dispositivos lógicos reprogramáveis que podem atingir desempenho elevado e fornecer flexibilidade de programação simultaneamente. Sendo assim a computação reconfigurável é uma forma de programação que utiliza dispositivos reconfiguráveis, configurados de acordo com descrições de hardware, para desenvolvimento, implementação e testes de cir-

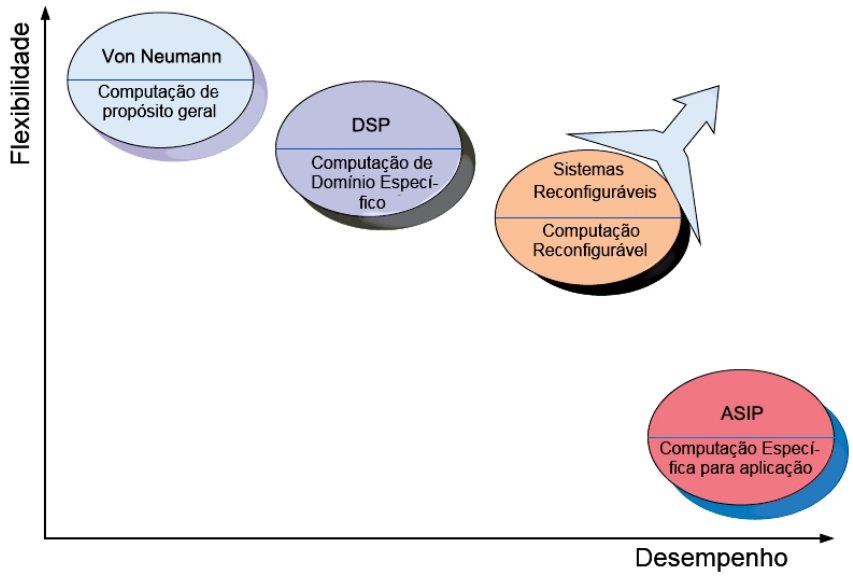

Figura 2.1: Dispositivos de Computação. Adaptado de Bobda (2007).

cuitos eletrônicos (Bobda (2007)). Desde seu início, conceitualmente no final dos anos 50 (Estrin (2002), Bobda (2007)), a computação reconfigurável tem sido cada vez mais utilizada devido as suas características básicas (Skliarova and Ferrari (2003)):

- Flexibilidade: a flexibilidade da computação reconfigurável é consequência do modo pelo qual o hardware é desenvolvido. Geralmente são utilizadas linguagens de descrição de hardware para descrever o hardware desejado. Em seguida o hardware é sintetizado e depois configura um dispositivo reconfigurável, como por exemplo, um FPGA. Desta forma uma modificação no hardware desenvolvido é simplesmente uma mudança na descrição deste hardware. Este fato torna o desenvolvimento mais barato e mais rápido em comparação ao desenvolvimento com ASIC's.

- Custo: o custo de desenvolvimento de hardware utilizando computação reconfigurável é menor pois não é necessário construir um *chip* para cada protótipo e os testes são feitos antes mesmo do hardware configurar o FPGA. Sendo assim, o hardware final possui um número muito menor de erros necessitando ser produzido, na maioria das vezes, uma única vez. Além do custo de desenvolvimento, o custo de uma plataforma para desenvolvimento com esta tecnologia é baixo se comparado com outros métodos de desenvolvimento de hardware.

- Eficiência: considerando as características anteriores, o hardware desenvolvido utilizando computação reconfigurável pode atingir alto nível de eficiência devido a possibilidade maior de otimizações se o tempo de desenvolvimento disponível for o mesmo tempo de desenvolvimento uti-

lizando outros métodos. A ocupação de área do hardware também é otimizada no sentido que ao longo do desenvolvimento as próprias IDEs possuem programas que calculam a melhor utilização de área possível para um determinado hardware dado um dispositivo reconfigurável.

Existem vários dispositivos que podem ser utilizados para implementar a computação reconfigurável como por exemplo as PLAs (*Programmable logic Arrays*), PALs (*Programmable Array Logics*), CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Arrays*). Os principais dispositivos utilizados em desenvolvimentos baseados em computação reconfigurável são as FPGA's devido ao fato de possuírem tantos recursos quanto um ASIC juntamente com a flexibilidade inerente à dispositivos reprogramáveis atingida em grande parte pela existência das linguagens de descrição de hardware (Skliarova and Ferrari (2003)).

2.2 FPGAs

A computação reconfigurável incentivou o desenvolvimento de dispositivos reconfiguráveis. O hardware reconfigurável ideal seria um hardware cujas interconexões entre os milhões de transistores que compõem o chip pudessem ser reconfiguradas para que computassem uma nova função qualquer de uma aplicação, sem restrições de uso de componentes e de conectividade entre estes componentes. Como a implementação de um hardware desta forma ainda não é possível, existem formas simples de implementação que se aproximam de forma considerável do modelo ideal e são utilizadas pelos FPGAs (Yalamanchili (2001)).

Os FPGAs consistem em um conjunto de células lógicas programáveis (*Logic Blocks*) organizadas em um dispositivo de forma a modelar um vetor de recursos de hardware programável. O dispositivo foi denominado *Field Programmable Gate Array* por conter conjuntos de lógicas programáveis em campo, ou seja, o *chip* é programado por seu usuário final após ser produzido. De acordo com a organização destes elementos no chip (posicionamento de blocos lógicos e paradigma de interconexão), que é diferente de acordo com o fabricante, os FPGAs podem ser classificadas em quatro grupos: *symmetrical array*, *row-based*, *hierarchy-based* e *sea of gates*.

O dispositivo que será utilizado no desenvolvimento deste projeto de pesquisa, Altera Cyclone II (figura 2.2), pertence ao grupo das *hierarchy-based* FPGA's. Neste tipo de organização as células são dispostas de forma hierárquica sendo que os elementos de menor granularidade estão na parte mais baixa da hierarquia e são agrupados para formar os elementos de nível superior. Desta forma cada elemento do nível i consiste em um grupo de elementos

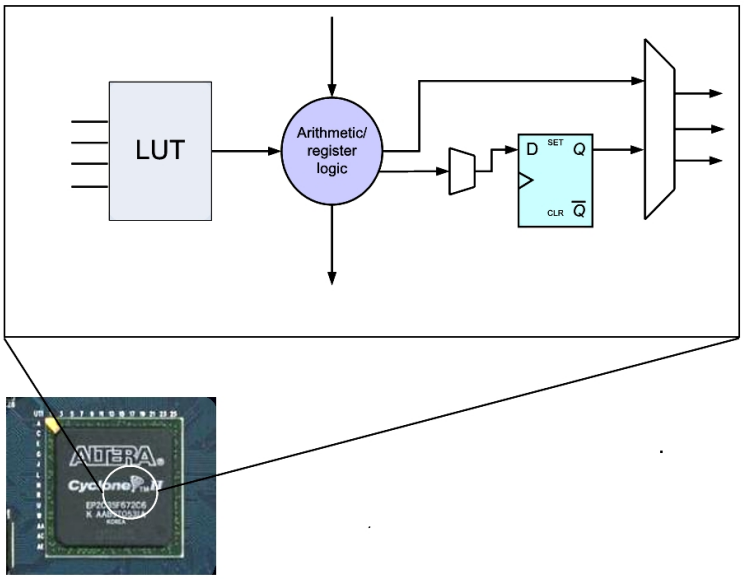

Figura 2.2: Cyclone II FPGA and Logic Block (Bobda (2007)).

do nível i-1 (Bobda (2007)). A figura 2.3 mostra como seria esta organização de forma genérica.

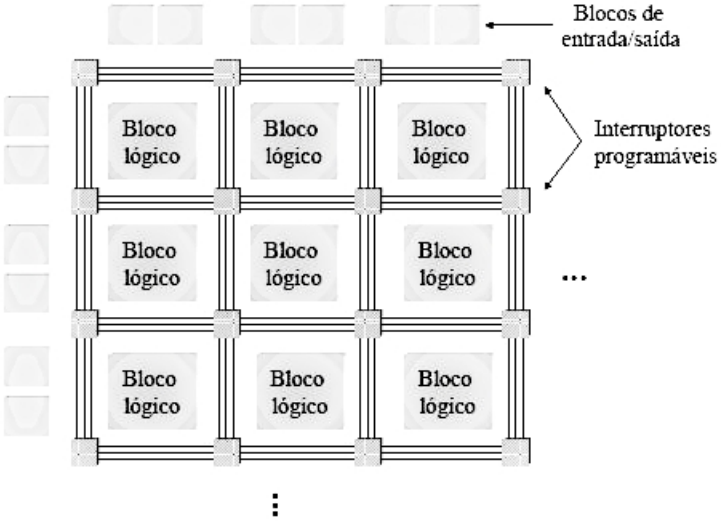

Figura 2.3: Arquitetura típica de um FPGA (Skliarova and Ferrari (2003)).

Os FPGA's da Altera possuem dois níveis básicos de hierarquia, o mais baixo que são as células lógicas e o mais alto que são os vetores de blocos lógicos (*Logic Array Blocs* ou LABs). O FPGA Cyclone II possui dezesseis elementos lógicos por LAB e as conexões de sinais que saem e que chegam nos pinos do dispositivo por uma estrutura de roteamento são chamadas de *MultiTrack*. O *MultiTrack* consiste em uma série de rápidos e contínuos canais interligados em linhas e colunas que ocupam todo o espaço do dispositivo. Os blocos lógicos nos FPGAs estão rodeados por blocos de entrada/saída (Input/Output ou I/O). Estes blocos de I/O são programáveis permitindo desta forma que o desenvolvimento interno configure os pinos como entrada, saída

ou entrada e saída. Cada pino configura um *Tri-State Buffer* que possui um bloco para entrada, um bloco para a saída e um bloco que ativa a saída onde estes pinos também estão ligados ao barramento *MultiTrack*. Os FPGAs, por serem dispositivos complexos, necessitam de um fluxo de *design* para serem utilizados.

2.3 Fluxo de Design em FPGAs

O design em dispositivos reprogramáveis como as FPGAs possui um fluxo que é apresentado na figura 2.4.

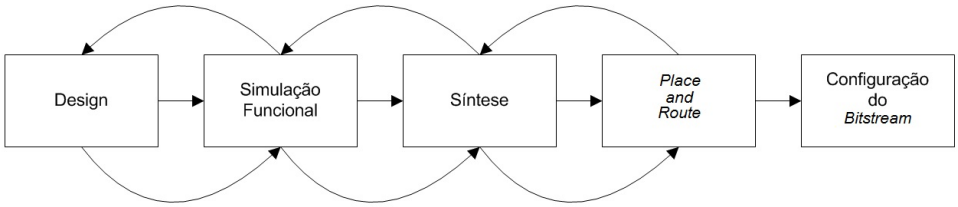

Figura 2.4: Fluxo de *design* em FPGAs. Adaptado de Bobda (2007).

Inicialmente a descrição do hardware é feita através de um esquemático, descritor de máquinas de estados finitos ou de linguagens de descrição de hardware. As linguagens de descrição de hardware (HDLs: *Hardware Description Languages*) são linguagens utilizadas para descrição formal de circuitos eletrônicos e lógica digital. A linguagem permite descrever o modo de operação do sistema, o *design* e a organização. As linguagens de descrição de hardware são similares às linguagens de programação concorrentes devido ao paralelismo intrínseco do hardware possuindo notações explícitas para expressar concorrência (Ghosh (1999)).

Este tipo de linguagem é utilizado para descrever circuitos e com a utilização de simuladores é possível analisar o hardware antes de implementá-lo fisicamente. Após a descrição do hardware, um programa chamado *Synthesizer* atribui operações lógicas de hardware a partir das declarações da linguagem e produz um *netlist* equivalente, de primitivas genéricas de hardware, para implementar o comportamento descrito. Estes programas costumam ignorar as restrições de tempo e utilizar um *clock* para controlar o sistema.

Existem várias linguagens de descrição de hardware por exemplo: AHDL, Handel-C, C-to-Verilog, SystemC, PALASM, Verilog, VHDL, dentre outras. A linguagem escolhida para desenvolvimento deste projeto de pesquisa foi o VHDL. Conforme detalhado a seguir, dentre outros fatores, a escolha por VHDL acontece basicamente por ser uma linguagem que possui um amplo histórico de utilização para desenvolvimento de hardware, é estruturada e

modular, possui vários códigos de dipositivos disponíveis que podem ser reutilizados e suas bibliotecas estão em sua fase madura de desenvolvimento apresentando poucos erros e muito conteúdo (Pedroni (2004),Pellerin and Taylor (1997)).

VHDL (*VHSIC Hardware Description Language; VHSIC: Very High Speed Integrated Circuit*) é uma das linguagens de descrição de hardware mais utilizadas no meio acadêmico. Foi inicialmente desenvolvida pelo Departamento de Defesa dos EUA para documentar o comportamento dos ASICs. A sintaxe de VHDL é muito parecida com a sintaxe da linguagem de programação ADA. Um exemplo de código em VHDL pode ser visto na figura 2.5.

```
-- (this is a VHDL comment)

-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity ANDGATE is
   port (
         IN1 : in std_logic;
         IN2 : in std_logic;
         OUT1: out std_logic);
end ANDGATE;

architecture RTL of ANDGATE is
begin

  OUT1 <= IN1 and IN2;

end RTL;
```

Figura 2.5: Exemplo de código VHDL.

A utilização de linguagens de descrição de hardware, como VHDL, auxilia no desenvolvimento devido a suas características principais (Yalamanchili (2001)):

- Interoperabilidade: VHDL possui um conjunto de construções que podem ser aplicadas em múltiplos níveis de abstração. Este fato expande significativamente o escopo de aplicações da linguagem e promove um modelo padronizado e portável de sistemas eletrônicos.

- Independência de Tecnologia: as descrições de design feitas não estão atreladas a nenhuma metodologia específica de desenvolvimento ou tecnologia alvo. VHDL consegue descrever hardware em nível RTL, conjunto de instruções e *switching transistor*.

- Reutilização de *Design*: várias bibliotecas desenvolvidas em VHDL são compartilhadas entre desenvolvedores.

- Prototipação de Hardware/Software: no caso de HDLs a descrição de hardware pode ser simulada e um software pode ser testado nesta simulação podendo desta forma ser avaliado.

As linguagens de descrição de hardware estão no núcleo das metodologias de desenvolvimento dos sistemas digitais modernos. Este tipo de tecnologia é uma das formas de descrever o hardware que irá configurar as FPGAs.

Após a descrição do hardware, o segundo passo do fluxo de desenvolvimento é a simulação funcional. O objetivo desta simulação é verificar se para uma dada entrada de dados o sistema apresenta a saída desejada. Esta simulação é feita através de softwares e geralmente seu resultado é mostrado em uma interface gráfica. Estes passos descritos anteriormente preparam o sistema para a síntese lógica.

Durante a síntese lógica o sistema é descrito como um conjunto de equações booleanas. As equações booleanas são então mapeadas para funções disponíveis na biblioteca de funções do dispositivo alvo. No caso do FPGA em questão, as funções são mapeadas para tabelas chamadas LUTs (*Look-up Tables*) que são os módulos básicos que irão implementar os operadores booleanos. O resultado da fase de síntese é o chamado *netlist* que descreve os módulos utilizados para implementar as funções e também suas interconexões. Para o *netlist* gerado os operadores precisam ser alocados no dispositivo escolhido e conectados através de um roteamento. Estes passos (*Place and Route*) são geralmente feitos automaticamente por ferramentas disponibilizadas pelos fabricantes dos FPGAs. A configuração gera um código chamado *bitstream* que possui todas as informações deste processo codificadas e prontas para configurarem o dispositivo alvo. O surgimento computação reconfigurável e os FPGAs possibilitou a criação de um novo conceito em computação, os chamados *soft-processors*.

2.4 Soft-Processors

A computação reconfigurável permitiu que o hardware fosse descrito por linguagens de descrição de hardware e configurado em dispositivos reconfiguráveis como os FPGAs. A partir desta possibilidade, hardwares cada vez mais complexos começaram a ser descritos devido ao aumento da capacidade destes dispositivos. Este aumento de capacidade possibilitou que desenvolvedores descrevessem processadores inteiros em hardware reconfigurável e estes processadores são os chamados *soft-processors*.

Para ser considerado um *soft-processor* a única característica necessária é que ele seja 100% configurável. O contexto dos *soft-processors* é diferente comparado ao contexto dos processadores normais por estarem configurados

em hardware reconfigurável. Este fato impacta diretamente em seu desenvolvimento e em suas características. A utilização deste tipo de processadores foi uma alternativa ao crescimento da utilização de microprocessadores no desenvolvimento de sistemas embarcados que era possível apenas com a adição de *hard-processors* (os processadores comuns) nas plataformas de desenvolvimento.

Existem vantagens e desvantagens na utilização de *soft-processors* e *hard-processors* no desenvolvimento de sistemas embarcados, porém a utilização de *soft-processors* tem sido mais vantajosa devido a vários motivos. Alguns dos motivos apontados em Yiannacouras et al. (2005) são listados a seguir:

- Geralmente o número de *hard-processors* disponíveis em uma plataforma de desenvolvimento com FPGAs não irá ser o correto para atender aos requisitos dos projetos, sendo ou muito grande ou muito pequeno.
- A posição das conexões entre os *hard-processors* e o FPGA torna mais difícil a etapa de *Place and Route* durante o desenvolvimento do sistema.
- Utilizando um chip genérico de FPGA, um projetista pode implementar o número de *soft-processors* que for necessário, e possível de acordo com o hardware, para o sistema e as ferramentas de desenvolvimento serão responsáveis por fazer o *Place and Route* dos processadores.
- Estes processadores, por serem implementados em lógica configurável, podem ser modificados com relação a sua complexidade de implementação para atingir exatamente os requisitos do sistema.

Os fabricantes de FPGAs disponibilizam, de forma gratuita ou comercialmente, *soft-processors* prontos para serem configurados em seus dispositivos juntamente com ferramentas de desenvolvimento para que estes processadores sejam programados. Alguns exemplos de *soft-processors* disponibilizados por fabricantes de FPGAs são: Nios II (Altera) e MicroBlaze (Xilinx). Existem também *soft-processors* que não são desenvolvidos por fabricantes de FPGAs, como por exemplo o LEON, que possuem a vantagem de poderem ser utilizados em qualquer FPGA com pequenas modificações.

2.5 Considerações Finais

Este capítulo apresentou os conceitos básicos de computação reconfigurável, os FPGAs (dispositivos utilizados para implementação de hardware reconfigurável) e seu fluxo básico de *design* juntamente com o conceito de *soft-processors*. Nesta dissertação de mestrado o objetivo principal é o desenvolvimento de um co-projeto de hardware/software para o algoritmo de corre-

lação de imagens visando atingir um ganho de desempenho com relação à implementação totalmente software. A escolha da computação reconfigurável utilizando linguagens de descrição de hardware e FPGAs se justifica por ser o modelo de desenvolvimento que apresenta menor custo, consome menos tempo em comparação com outras técnicas de desenvolvimento de hardware e tem apresentado resultados satisfatórios.

O aparecimento da computação reconfigurável e dos FPGAs proporcionou uma mudança no paradigma de desenvolvimento de sistemas embarcados. Anteriormente o desenvolvimento de hardware era feito através de esquemáticos em nível RTL (*Register Transfer Level*) e os *chips* continham apenas hardware devido a complexidade exigida por um sistema que executa um software. Após o conceito de computação reconfigurável ser adotado juntamente com as linguagens de descrição de hardware, projetos mais complexos começaram a ser desenvolvidos e, juntamente com o aparecimento dos *soft-processors*, foram fatores decisivos para que projetos de hardware se tornassem co-projetos de hardware/software. Este tema será abordado no capítulo seguinte.

CAPÍTULO

3

Co-Projeto de Hardware/Software de Sistemas Embarcados

3.1 Sistemas Embarcados

Apresentar uma definição precisa de sistemas embarcados não é uma tarefa trivial (Vahid and Givargis (2001)). Entretanto existem várias definições disponíveis na literatura que quando analisadas de forma conjunta apresentam as características principais que definem estes sistemas. A seguir são apresentadas algumas definições:

- Um sistema embarcado é aproximadamente qualquer sistema de computação que não seja um *desktop* (Vahid and Givargis (2001)).
- Sistemas embarcados são sistemas dedicados a realizar uma tarefa específica (Berger (2001)).
- Sistemas embarcados são sistemas orientados à aplicação e desenvolvidos observando-se restrições relacionadas ao tempo relativo de suas ações (Gupta (1995)).
- Um sistema embarcado é uma combinação de hardware e software, e talvez alguma parte mecânica adicional, desenvolvido para desempenhar uma função específica (Valderrama et al. (2000)).
- Um sistema cuja principal função não é computacional porém é controlado por um computador embarcado no mesmo (Wilmshurst (2006)).

Analisando as definições apresentadas, pode-se definir um sistema embarcado como: um sistema de computação desenvolvido para um propósito específico segundo algumas restrições sendo as principais: tempo (projeto e execução), custo e área ocupada.

Utilizando as definições apresentadas acima, encontrar sistemas embarcados presentes no dia-a-dia das pessoas torna-se trivial. Alguns exemplos de sistemas embarcados podem ser encontrados na figura 3.1, como celulares, câmeras, utensílios domésticos, aparelhos de televisão, caixas automáticos em bancos, sistemas embarcados em veículos, dentre outros.

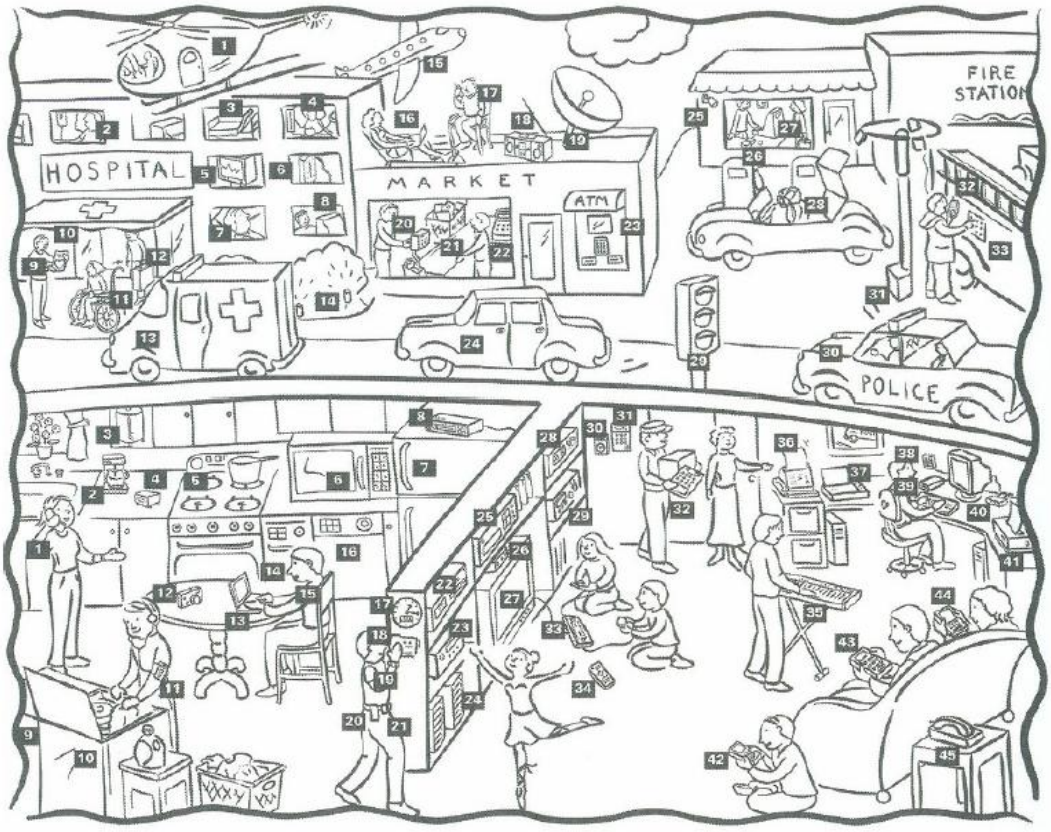

Figura 3.1: Exemplos de Sistemas Embarcados (Vahid and Givargis (2001)).

Existem algumas características que diferenciam os sistemas embarcados e devem ser levadas em consideração durante seu projeto (Berger (2001)):

- Sistemas embarcados possuem o suporte de uma grande variedade de processadores e de arquiteturas de processadores.
- Sensibilidade com relação ao custo do sistema.
- Sistemas embarcados possuem restrições de tempo. Os sistemas podem possuir restrições de tempo normais ou críticas.
- Ao utilizar um sistema operacional em um sistema embarcado este deve preferencialmente ser um RTOS (*Real Time Operating System*).

- As falhas de software em um sistema embarcado são mais graves em comparação às falhas de um software em um *desktop*.
- Sistemas embarcados possuem restrições de consumo de energia.
- Os sistemas embarcados devem conseguir operar em situações extremas.
- Recursos disponíveis em um sistema embarcado são consideravelmente menores em comparação a outros sistemas, portanto usualmente busca-se explorar estes recursos ao máximo.

Ao desenvolver um sistema embarcado, todas as características previamente apresentadas devem ser levadas em conta devido a vários fatores como: o número de unidades a serem produzidas que se relaciona diretamente com o custo do projeto; o tempo de vida esperado para o sistema que deve ser atingido com plena funcionalidade do mesmo e também a confiabilidade do sistema que no caso deve ser tolerante a falhas por estar atuando em situações críticas (Valderrama et al. (2000)).

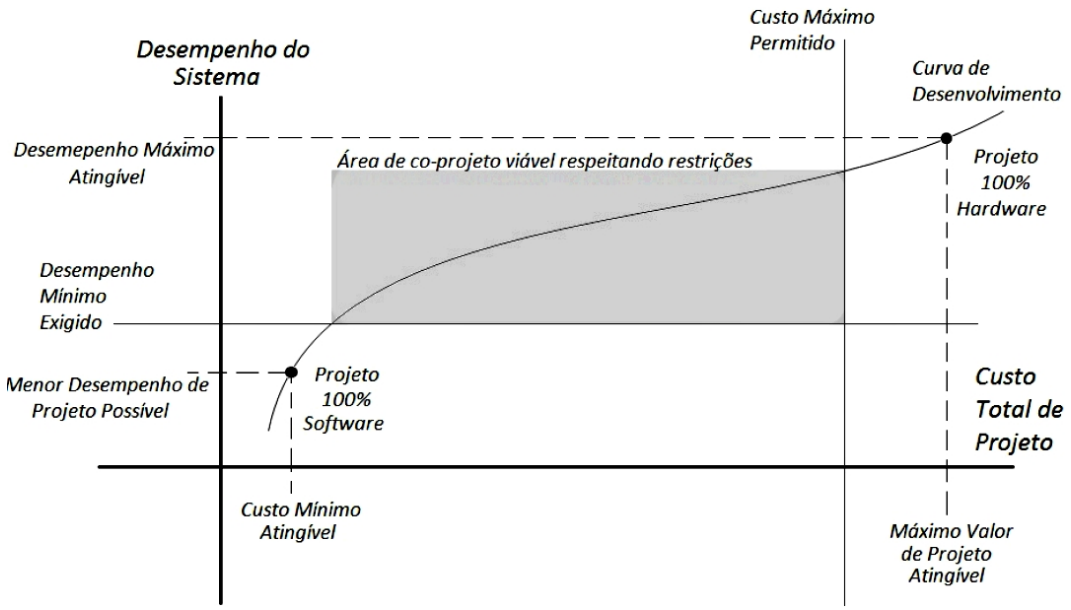

Figura 3.2: Análise de um co-projeto. Adaptado de Gupta (1995).

O fato de sistemas embarcados possuírem objetivos de utilização definidos faz com que o projeto deste tipo de sistema seja uma experiência diferente. Ao desenvolver sistemas computacionais para atingir certos objetivos, percebe-se que um sistema (genérico ou específico) não pode ser utilizado para várias aplicações e atingir de forma ótima estes objetivos (Wolf (2006)). O projetista de sistemas embarcados deve construir uma implementação que cumpra a funcionalidade desejada porém o maior desafio é construir esta implementação de forma a simultaneamente otimizar várias métricas de projeto como

custo, tempo, desempenho, consumo de energia, flexibilidade, *time-to-market*, segurança, manutenibilidade e corretude (Vahid and Givargis (2001)).

O desenvolvimento de sistemas de propósito geral permite que os componentes de hardware e de software possam ser desenvolvidos separadamente e em paralelo. Os sistemas embarcados necessitam de um desenvolvimento conjunto de hardware e de software pois, na maioria dos casos, a melhor forma de atingir os requisitos do sistema de forma eficiente e rápida é desenvolvendo hardware e software concorrentemente e de forma integrada. A área destacada no gráfico da figura 3.2 mostra que o desenvolvimento concorrente consegue se adequar às restrições de sistema mais facilmente. Esta nova necessidade que surgiu juntamente com os sistemas embarcados criou uma nova maneira de projetar sistemas chamada co-projeto de hardware/software.

3.2 Co-Projeto de Hardware/Software

As origens do co-projeto de hardware/software estão nas pesquisas de sistemas embarcados nas quais projetistas geralmente tem utilizado microprocessadores com hardware customizado para obter melhor desempenho. O potencial comprovado destes sistemas combinados com a evolução das plataformas reprogramáveis tem despertado interesse nos sistemas de microprocessadores com co-processadores (Kaplan and Kastner (2000)).

Co-projeto de Hardware/Software (*Hardware/Software Co-Design*) é um paradigma de projetos cujo objetivo consiste no projeto de sistemas que satisfaçam as restrições encontrando um particionamento otimizado e balanceado entre componentes de hardware e software do sistema. Devido à heterogeneidade de componentes, hardware e software devem ser projetados concorrentemente (Valderrama et al. (2000)). Utilizar esta técnica significa unir todos os objetivos do sistema explorando a interação entre os componentes de hardware e software durante o desenvolvimento (Hidalgo and Lanchares (1997)).

No contexto de sistemas embarcados, é importante que o projeto seja otimizado para que o produto final consiga atingir as restrições de forma otimizada. Neste caso a utilização do co-projeto é indicada pois seu método explora as interações entre hardware e software com foco na otimização do sistema e quando as duas partes do sistema são desenvolvidas de forma conjunta o sistema irá conter apenas as características necessárias para seu funcionamento de forma correta (de Micheli and Gupta (1997)). A grande diferença apresentada por este método é a tentativa de otimização de várias partes diferentes do sistema ao mesmo tempo fazendo com que os projetistas utilizem várias ferramentas exaustivamente durante o desenvolvimento (Wolf (2006)).

A figura 3.3 mostra como é um fluxo básico de um co-projeto de hard-

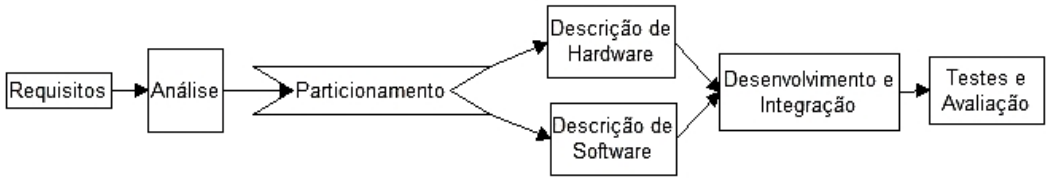

Figura 3.3: Fluxo básico de um co-projeto.

ware/software. Inicialmente os requisitos de um produto são enviados para um equipe de desenvolvimento e são analisados. A análise consiste em verificar como podem ser implementadas as funcionalidades contidas nos requisitos e propor as implementações possíveis. A fase de projeto inicia-se com o particionamento do sistema, que é uma das fases mais importantes porque as decisões de projeto nesta fase irão impactar todas as outras fases, esta fase será discutida mais a fundo nos parágrafos seguintes. Feito o particionamento, as partes que serão implementadas já estão agrupadas em partes de hardware e partes de software. Após o desenvolvimento estas partes deverão ser integradas para que possam ser feitos os testes e avaliações sobre o sistema.

O co-projeto de um sistema embarcado atualmente se apresenta como uma modificação do fluxo básico apresentado pela figura 3.3. Durante e entre as fases apresentadas existe uma grande quantidade de iterações e otimizações ocorrendo. Ao longo do tempo, os projetos utilizando este método mostraram também que o fluxo deveria possuir recorrências pois muitos defeitos e erros de projetos faziam com que o projeto voltasse à fases anteriores. O fluxograma que representa como o co-projeto é feito atualmente é mostrado na figura 3.4.

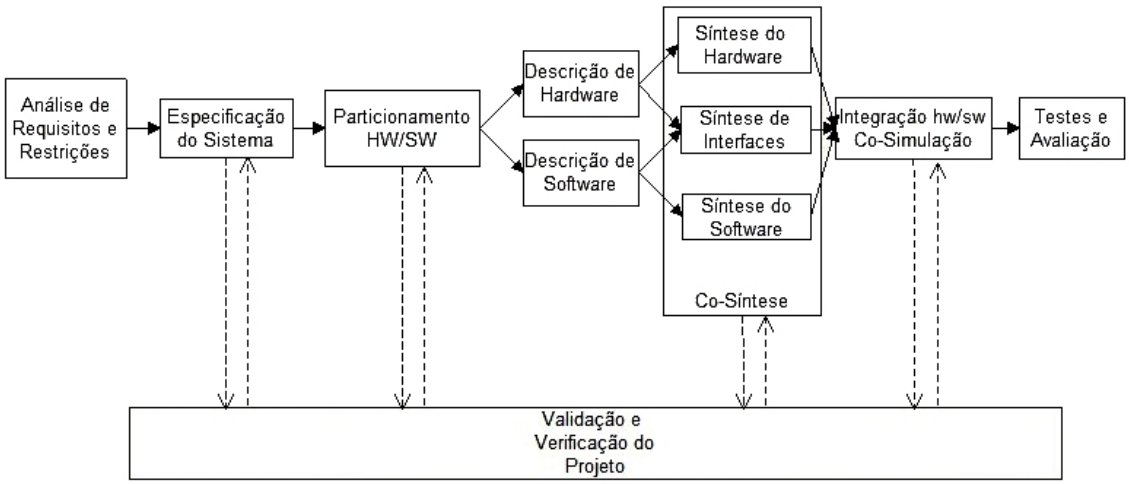

Figura 3.4: Fluxo atual de um co-projeto.

As mudanças básicas com relação ao fluxograma apresentado na figura 3.3 são: o aparecimento da fase de co-síntese, da parte de co-simulação durante a integração do hardware e do software, e também a possibilidade de recor-

rência das fases após feita a validação e verificação do projeto ao longo do desenvolvimento.

A fase de co-síntese compreende a síntese do hardware para um dispositivo físico alvo e também a compilação do software gerando os executáveis para a arquitetura alvo juntamente com um escalonador que permita a interação correta dos módulos do sistema. A síntese de interface gera os softwares de sincronização dos módulos de hardware e de software desenvolvidos.

Após a fase de co-síntese ocorre a integração dos módulos juntamente com a co-simulação. No desenvolvimento de sistemas embarcados vários módulos diferentes são desenvolvidos sendo necessários vários simuladores trabalhando simultaneamente para que o sistema seja simulado de forma correta. Isto significa que muitas vezes é preciso um sistema de validação heterogêneo onde ferramentas diferentes são utilizadas no processo de validação, isto é, uma co-simulação.

3.2.1 Particionamento Hardware/Software

O particionamento hardware/software é a fase do co-projeto onde são decididas quais partes do sistema serão implementadas em hardware e quais partes do sistema serão implementadas em software. Esta decisão geralmente considera métricas como desempenho, consumo de energia, área ocupada em *chip* e custo. Esta fase depende da fase de análise de requisitos, para que os módulos sejam identificados, e impacta diretamente na fase de co-síntese sendo de grande importância para o projeto em sua totalidade. Caso uma decisão seja feita de forma errônea o projeto seguramente irá retornar à fase de particionamento após a verificação ou após a validação. O problema do particionamento remete a um grupo de módulos conectados que deverão ser agrupados de forma a satisfazer os requisitos de sistema e também otimizar o máximo possível de variáveis de projeto.

A fase de particionamento do sistema, como mostrado nas figuras 3.3 e 3.4, deve ser feita de forma que a funcionalidade do hardware seja determinada antes da fase de síntese (Kaplan and Kastner (2000)) mesmo que o diagrama de desenvolvimento apresente ciclos. Anteriormente, o particionamento era feito pelos próprios projetistas utilizando sua experiência em projetos. Com o aumento da complexidade dos sistemas embarcados atuais e do hardware disponível para implementá-los, este processo de particionamento levaria um tempo que não é compatível com o *time-to-market* atual de desenvolvimento. A solução para este problema é utilizar alguma das ferramentas disponíveis no mercado ou então desenvolver o sistema de particionamento. Por se tratar de um problema NP-Difícil (Aratò et al. (2003)), as ferramentas de particionamento geralmente utilizam heurísticas (como Algoritmos Genéticos e Redes

Neurais Artificiais (Dias and Lacerda (2009))) para resolver o problema já que se trata de um espaço de projeto de tamanho exponencial.

Basicamente os sistemas de particionamento utilizam o processo de partir de uma solução inicial e modificá-la de acordo com os requisitos de sistema. As primeiras soluções consideravam a solução totalmente em hardware e depois modificavam para software os módulos do sistema aos poucos. A medida que os algoritmos foram evoluindo e os co-projetos começaram a ser mais comuns e conseqüentemente analisados, a solução inicial começou a ser criada totalmente por software e os módulos são modificados para hardware segundo os requisitos de sistema (Kaplan and Kastner (2000)).

Existem algumas características comuns aos sistemas de particionamento hardware/software (Gajski et al. (1997)):

- Nível de abstração da especificação: esta característica está relacionada a como os dados serão modelado ao serem utilizados no particionamento. Estes dados podem ser por exemplo grafos orientados à fluxo de dados, grafos acíclicos direcionados, conjunto de tarefas, transferência de registradores.

- Granularidade: define o nível de decomposição da especificação inicial em objetos funcionais e caracteriza-se pela quantidade de informação contida na especificação de cada objeto funcional.

- Métricas e estimadores: são atributos que permitem classificar um particionamento como bom ou ruim e apresentar o quão bom ou quão ruim está o particionamento. Estes atributos são por exemplo consumo de energia, tempo de execução, custo do sistema final, área ocupada do chip.

- Algoritmo de particionamento: o algoritmo pode ser incremental que inicia a solução vazia depois resolve de forma incremental ou pode iniciar de uma solução aleatória e modificá-la.

- Formato do resultado: o resultado pode ser representado por uma lista, por um grafo, por uma estrutura de dados específica porém o importante é que ele possa ser utilizado diretamente para as próximas fases do co-projeto.

COSYMA (Ernst et al. (2002)), VULCAN (Gupta and Micheli (1993)), LYCOS (Madsen et al. (1997)), SpecSynth (Vahid and Gajski (1995)) e PURE (Eles et al. (1997)) são algumas ferramentas conhecidas de particionamento automático enquanto POLIS (Chiodo et al. (1996)), COSMOS (Ismail et al. (1994)) e CASTLE (Camposano and Wilberg (1996)) são ferramentas de particionamento interativo, no caso, que permitem intervenção do usuário.

3.2.2 Profiling

Os algoritmos utilizados por ferramentas de particionamento e também os métodos de desenvolvimento de co-projeto de hardware/software necessitam de informações sobre desempenho dos módulos antes de particionarem o sistema. As ferramentas de *profiling* são ferramentas CAD (*Computer-aided Design*) que medem o desempenho do sistema baseado no tempo necessário para a execução de funções. A utilização destas ferramentas proporciona a detecção de gargalos de desempenho e auxilia na otimização durante o desenvolvimento de sistemas embarcados (Tong and Khalid (2008)).

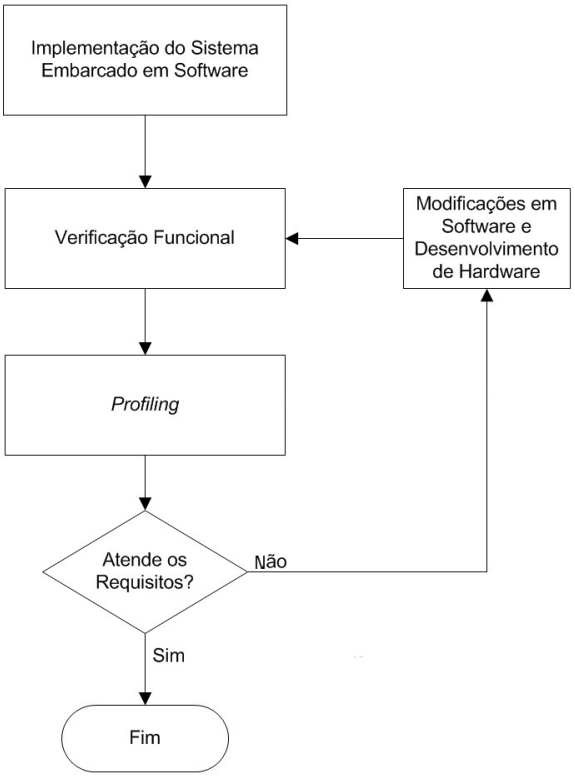

Figura 3.5: Fluxo básico de *Profiling-Based Methods*

A utilização de ferramentas de *profiling* como auxílio durante o desenvolvimento de co-projetos de hardware/software deu origem aos chamados métodos baseados em *profiling* ou *Profiling-Based Methods*. Estes métodos adicionam um passo de *profiling* de código ao fluxo do co-projeto, modificando-o de forma a desenvolver o sistema acelerando por software e por hardware ao mesmo tempo as funções identificadas como gargalos do algoritmo. A figura 3.5 apresenta o fluxograma básico dos métodos de co-projeto baseados em *profiling*.

As ferramentas de profiling disponíveis para utilização, tanto comercialmente como de forma gratuita, podem ser classificadas em três tipo básicos (Tong and Khalid (2008)): baseadas em software, baseadas em hardware e baseadas em FPGA (figura 3.6).

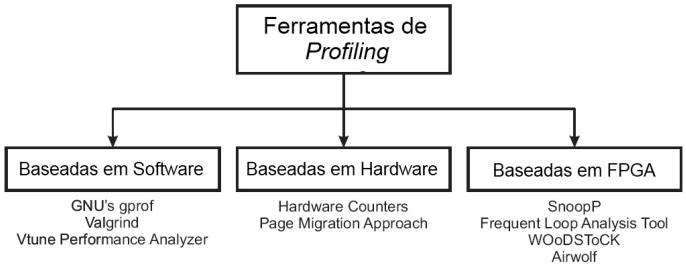

Figura 3.6: Ferramentas de *Profiling*. Adaptado de Tong and Khalid (2008).

Ferramentas de *profiling* baseadas em software podem atuar de duas maneiras diferentes: simulando a execução do software em um processador e colhendo dados desta execução, ou através da inserção de código de instrumentação no software o que possibilita a verificação de desempenho durante a real execução do programa em um microprocessador. As ferramentas baseadas em hardware, que estão presentes também em processadores comerciais, funcionam através de contadores de hardware que monitoram eventos como acesso a memória, *cache miss*, tipos de instrução executadas e bolhas no *pipeline*. Por fim, as ferramentas baseadas em FPGA utilizam execução de software em *soft-processors* para as medições adquirindo as informações através de um hardware *on-chip* de *profiling*.

Tabela 3.1: *Ferramentas de Profiling. Adaptado de Tong and Khalid (2008).*

Característica	gprof	ISS	VTune	Purify	HWC	PMA	SnoopP
Código de Instrumentação	X		X	X			
Amostragem	X		X	X	X	X	
Precisão de Ciclos de Clock							X
Overhead de Desempenho	X	X	X	X		X	
Simulação		X	X				
Baseada em Hardware					X	X	
Baseada em Software	X	X	X	X			
Baseada em FPGA							X
Profile Funcional	X	X					X
Profile de Memória				X	X	X	

A tabela 3.1 apresenta uma comparação de características desejáveis entre algumas ferramentas de *profiling*. Várias ferramentas possuem características interessantes e algumas se destacam como a SnooP que é a única a possuir precisão de ciclos de clock e o *Gnu Profiler* (gprof) que possui várias características interessantes, acompanha o *Gnu C Compiler* (GCC) e também o compilador C do *soft-processor* Nios II.

3.3 Considerações Finais

Este capítulo apresentou as definições básicas e conceitos relacionados aos sistemas embarcados, os métodos utilizados durante o desenvolvimento destes sistemas segundo o paradigma chamado co-projeto de hardware/software e, por fim, definiu e comparou as ferramentas de *profiling* situando-as ao longo do desenvolvimento dos co-projetos. A computação reconfigurável proporciona uma ampla e eficiente implementação de um co-projeto pois permite o agrupamento de componentes de hardware e de software em um único dispositivo, os FPGAs. Além disto, por serem reconfiguráveis, os FPGAs são mais adequadas ao ciclo de refinamento do particiomanento e do co-projeto, uma vez que tanto o software quanto o hardware podem ser revisados e alterados mais facilmente.

Sendo assim, a implementação do sistema proposto nesta dissertação adota o método de co-projeto de hardware/software baseado em *profiling* utilizando FPGAs. A escolha justifica-se pelo fato de que este método de co-projeto exige um tempo de desenvolvimento geralmente menor do que um projeto puramente baseado em hardware e, também, porque a aplicação em questão exige muito processamento e exige que este processamento seja feito de forma rápida.

CAPÍTULO

4

Correlação de Imagens

4.1 Processamento de Imagens

A área de processamento de imagens surge principalmente para suprir as necessidades de duas área de aplicação: melhoria da informação visual para interpretação humana e processamento de dados de cenas para percepção automática através de máquinas (Gonzalez and Woods (2006)). Inicialmente o processamento de imagens foi utilizado para melhorar a qualidade de imagens digitalizadas para jornais e revistas e posteriormente aplicado em várias áreas do conhecimento para solução de grande variedade de problemas. O processamento de imagens é uma área de pesquisa de crescente relevância atualmente.

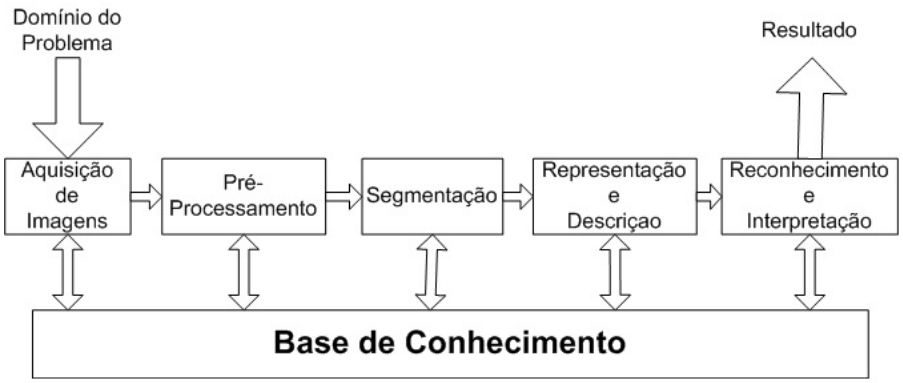

Figura 4.1: Passos Fundamentais (Adaptado de Gonzalez and Woods (2006)).

O processamento de imagens digitais envolve as áreas de hardware, software e fundamentos teóricos. A figura 4.1 apresenta os passos fundamentais em processamento de imagens digitais. Inicialmente é feita a aquisição de

imagens através de um sensor óptico. O sensor é escolhido de acordo com a aplicação ou então de acordo com a disponibilidade de recursos que se tem no sistema, como no caso por exemplo de um sistema embarcado.

A imagem após sua aquisição é representada como uma função $f(x, y)$ como mostra a figura 4.2. A função $f(x, y)$ representa uma imagem digital que é composta por uma matriz de elementos de imagem ou *pixels*. Para ser adequada para o processamento computacional a função precisa ser digitalizada espacialmente e em amplitude para que a imagem seja representada de forma digital. A amostragem da imagem é a digitalização das coordenadas espaciais (x,y) e a quantização em níveis de cinza é a digitalização da amplitude. O processo de amostragem e quantização pode ser uniforme (valor fixo de resolução espacial) ou não uniforme (utilização de esquema adaptativo onde o processo de amostragem depende das características da imagem) (Gonzalez and Woods (2006)).

Formalmente a amostragem pode ser entendida como uma partição do plano xy em uma grade, onde cada cruzamento da grade é um par de elementos obtidos de Z^2 que é o conjunto de todos os pares ordenados (a, b) onde a e $b \epsilon Z$. Portanto $f(x, y)$ é uma imagem digital se (x, y) forem elementos de Z^2 e f uma função que atribui um valor de nível de cinza a cada par (x, y) distinto. Esta atribuição funcional é a quantização da imagem. A resolução (grau de detalhes discerníveis) da imagem está diretamente ligada ao tamanho da matriz de amostragem (NxM) e ao número de níveis de cinza utilizados, parâmetros que, a medida que são aumentados, aumentam diretamente a resolução da imagem. Embora relacionada com a resolução, a qualidade de uma imagem digital é um parâmetro subjetivo e altamente dependente da aplicação (Gonzalez and Woods (2006)).

Após a aquisição da imagem o passo de pré-processamento consiste de algoritmos que irão preparar a imagem de forma a auxiliar o processamento digital propriamente dito. Estes algoritmos geralmente tratam de brilho, intensidade, extração de ruído, seleção de regiões de interesse da imagem dentre outros.

Feito o pré-processamento a imagem pode ser segmentada. Nesta fase a imagem é dividida em partes ou objetos que constituem a imagem. Os algoritmos de segmentação quando bem implementados facilitam na solução de vários problemas de processamento, em contrapartida quando mal implementados estes algoritmos tendem a piorar estes problemas.

Ao passar pelo estágio de segmentação, os dados são agrupados como pixels e serão representados em fronteiras ou em regiões completas. Após esta etapa o processo de descrição extrai características que podem resultar em informações quantitativas de interesse ou que sejam básicas para discriminação

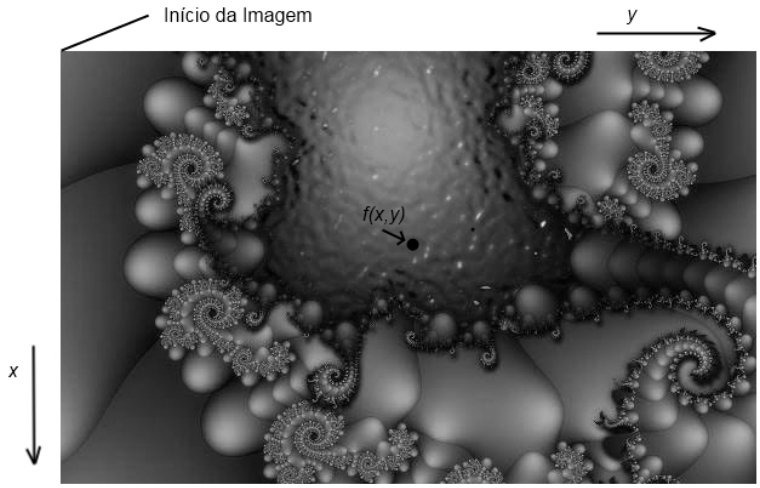

Figura 4.2: Representação da Imagem.

dos objetos (Gonzalez and Woods (2006)).

A última etapa consiste no reconhecimento que atribui rótulos aos objetos e também na interpretação que atribui significados à grupos de objetos. A base de conhecimento é extremamente importante pois auxilia todos os processos descritos facilitando sua execução e melhorando seu resultado. Um exemplo é saber a forma do objeto que é procurado pois facilitaria na fase de segmentação ou também se é melhor identificar as bordas ou regiões completas para o resultado final. Os resultados de cada fase podem ser visualizados durante o processamento pois resultam em imagens.

Os passos fundamentais de processamento de imagens digitais utilizam algoritmos que possuem geralmente um custo computacional elevado, pois costumam percorrer a imagem aplicando filtros ou buscando pontos de interesse ou características, e conseqüentemente dependem do tamanho da imagem utilizada como entrada. O desenvolvimento na área de sensores ópticos está evoluindo rapidamente e as imagens captadas estão cada vez maiores (em termos de amostragem e quantização) tornando os algoritmos ainda mais custosos. Aliando o aumento do custo computacional dos algoritmos ao fato de que a imagem pode usualmente ser percorrida e processada de forma paralela é justificável a aplicação do desenvolvimento de hardware para a execução otimizada destes algoritmos.

Um dos algoritmos utilizados em processamento de imagens, que é muito aplicado na principal área de pesquisa do laboratório de robótica móvel onde este trabalho está inserido, é a correlação de imagens. Métodos de navegação robótica baseados em visão utilizam o algoritmo de correlação de imagens com vários objetivos distintos. Exemplos de aplicações onde o algoritmo de correlação apresenta resultados satisfatórios são: a localização de "marcas

de referência"(*landmarks*) para que o robô encontre sua localização e também saiba seu deslocamento com relação a uma imagem de referência, ou também, o *matching* de imagens para *Appearance-Based Methods*. Este capítulo apresenta o algoritmo de correlação de imagens e suas principais aplicações em robótica móvel.

4.2 Correlação de Imagens

As imagens digitais podem ser representadas como funções. A correlação de imagens baseia-se na correlação de funções onde, dadas duas funções *f(x,y)* e *g(x,y)*, sua correlação denotada por $f(x,y) \circ g(x,y)$ é definida pela relação:

$$f(x,y) \circ g(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\alpha, \beta) g(x+\alpha, y+\beta) d\alpha d\beta. \tag{4.1}$$

Nesta relação, *f(x,y)* e *g(x,y)* são deslocados na integração de α e β entre $-\infty$ a $+\infty$ para cada valor de *x* e *y*. A propriedade interessante da correlação de funções é que ela atinge seu valor mais elevado para as funções que são semelhantes, atingindo seu valor máximo para funções exatamente iguais.

Levando em consideração esta propriedade e sabendo que as imagens são representadas como funções pode-se encontrar a posição de partes de uma imagem na própria imagem ou encontrar a imagem com maior grau de semelhança em comparação com outras, considerando um grupo de imagens. Existem algumas implementações do algoritmo de correlação de imagens e duas delas que se mostram mais robustas (Martin and Crowley (1995)) são apresentadas a seguir.

4.2.1 Correlação Cruzada

O algoritmo de correlação cruzada calcula o coeficiente de correlação entre duas imagens ou entre uma imagem e uma sub-imagem. O cálculo utiliza a forma discreta da equação 4.1. Inicialmente a discretização da equação 4.1 resulta na seguinte equação (Gonzalez and Woods (2006)):

$$\gamma(s,t) = \sum_{y=1}^{N} \sum_{x=1}^{M} f(x,y) w(x-s, y-t). \tag{4.2}$$

As relações entre as variáveis da equação 4.2 estão representadas na figura 4.3. No caso desta equação de cálculo da correlação $\gamma(s,t)$, *f(x,y)* representa a imagem de tamanho *N*x*M* e *w(x,y)* representa a sub-imagem que será buscada através da correlação de tamanho *J*x*K*. Os pontos *s* e *t* representam a coordenada *(x,y)* do template na imagem.

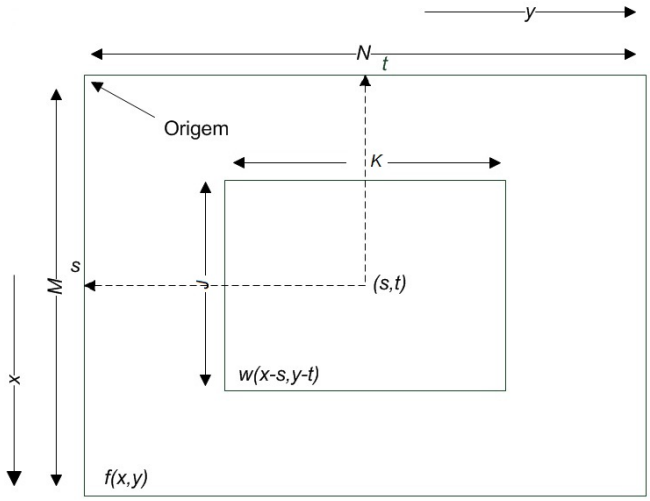

Figura 4.3: Variáveis da Equação 4.2 (Gonzalez and Woods (2006)).

A figura 4.4 apresenta um exemplo de utilização do calculo do coeficiente de correlação para encontrar a sub-imagem selecionada previamente em uma imagem atual. Inicialmente é escolhida uma sub-imagem, como definido na janela da esquerda, e então o algoritmo faz o cálculo e encontra a região da imagem atual que apresenta o maior coeficiente de correlação, ou seja, o maior grau de semelhança com a parte selecionada da imagem anteriormente obtida.

Figura 4.4: Exemplo da utilização do coeficiente de correlação.

O cálculo do coeficiente de correlação desta forma não apresenta elevado custo computacional em comparação com outros algoritmos de processamento de imagem porém apresenta problemas, como por exemplo a variação de brilho e a escala na imagem atual com relação à imagem anteriormente obtida. Afim de contornar estes problemas uma das forma mais utilizadas de cálculo do coeficiente de correlação cruzada é a forma normalizada. Normalizando o cálculo do coeficiente, este torna-se invariante à escala e brilho tanto para a máscara quanto para a imagem.

4.2.2 Correlação Cruzada Normalizada

A equação que descreve o cálculo do coeficiente de correlação cruzada normalizado, seguindo as variáveis definidas pela figura 4.3, é a seguinte:

$$\gamma(s,t) = \frac{\sum_{y=1}^{N}\sum_{x=1}^{M}[f(x,y) - \bar{f}(x,y)][w(x\text{-}s,y\text{-}t) - \bar{w}]}{\{\sum_{y=1}^{N}\sum_{x=1}^{M}[f(x,y) - \bar{f}(x,y)]^2[w(x\text{-}s,y\text{-}t) - \bar{w}]^2\}^{\frac{1}{2}}} \tag{4.3}$$

Nesta equação o coeficiente de correlação cruzada normalizado $\gamma(s,t)$ é calculado considerando-se os valores médios da janela $\bar{w}$ e região da imagem $\bar{f}$ na posição *(s,t)*, tanto no numerador (que é semelhante á equação 4.2), quanto no denominador que foi adicionado juntamente aos valores médios visando a normalização. Este coeficiente apresenta vários desafios para implementação em hardware como as operações de ponto flutuante, a raiz quadrada e também as potências. Entretanto, os somatórios que acessam as imagens podem ser implementados paralelamente apresentando um ganho no tempo de execução.

A complexidade de tempo para cálculo do coeficiente normalizado é maior em comparação à equação 4.2, porém os benefícios obtidos pela normalização são bem vindos principalmente por evitar muitas vezes o pré-processamento da imagem que teria que ser utilizado para realizar um ajuste de brilho ou escala. Existe uma variação para o algoritmo NCC chamada correlação cruzada normalizada rápida (*Fast Normalized Cross-Correlation* ou FNCC) que será definida a seguir.

4.2.3 Correlação Cruzada Normalizada Rápida

O custo computacional do cálculo do coeficiente de correlação normalizado é alto e existe um grande interesse em algoritmos que sejam capazes de calcular este coeficiente de forma mais eficiente. Para que fosse possível o cálculo mais rápido do coeficiente foram propostas algumas modificações e foi definida então a correlação cruzada normalizada rápida ou FNCC. A definição do algoritmo apresentada a seguir pode ser vista em Briechle and Hanebeck (2001) e também em Tsai and Lin (2003).

A idéia básica da modificação é analisar o numerador e o denominador do coeficiente de correlação inicialmente de forma separada. Para simplificar o cálculo do denominador, são utilizadas duas tabelas de somas $s(u,v)$ e $s^2(u,v)$ uma na imagem $f(x,y)$ e outra na energia da imagem $f^2(x,y)$.

$$s(u,v) = f(u,v) + s(u-1,v) + s(u,v-1) - s(u-1,v-1) \tag{4.4}$$

$$s(u,v) = f^2(u,v) + s^2(u-1,v) + s^2(u,v-1) - s^2(u-1,v-1) \tag{4.5}$$

O cálculo das tabelas de soma considera as variáveis s e t da equação 4.3 como u e v. Este cálculo é relativamente simples, porém a aceleração do algoritmo encontra-se na utilização destes valores já armazenados no cálculo do coeficiente e armazenar estes valores para uma imagem de tamanho considerável em hardware ocupa uma quantidade de memória nem sempre disponível.

$$\bar{t}(x,y) = \sum_{i=1}^{K} k_i t_i(x,y) \tag{4.6}$$

A parte de modificação no cálculo do numerador é reescrevê-lo de forma a agregar o conceito e expandir a função do *template* de média zero para uma soma ponderada de *K* funções de bases retangulares atingindo uma aproximação da função *template*. Todos os passos das demonstrações, reduções e modificações no coeficiente podem ser verificados em Briechle and Hanebeck (2001). Com a utilização das tabelas das equações 4.4 e 4.5 juntamente com a função do template de média zero de forma expandida representado pela equação 4.6 o cálculo do coeficiente de forma aproximada é dado por:

$$\bar{\gamma}(x,y) = \frac{\sum_{i=1}^{K} k_i (s(x_u^i + u, y_u^i + v) - s(x_u^i + u, y_l^i + v - 1) - s(x_l^i + u - 1, y_u^i + v) + s(x_l^i + u - 1, y_l^i + v - 1))}{\sqrt{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})^2 \sum_{x,y}(t(x-u, y-v) - \bar{t})^2}} \tag{4.7}$$

Este novo coeficiente apresentou bons resultados com relação ao tempo de execução tanto em Briechle and Hanebeck (2001) quanto em Tsai and Lin (2003). O problema destas mudanças é que, numa análise feita para implementação em sistemas embarcados, o fato de armazenar a tabela após calculada consome muita memória e os cálculos das tabelas também consomem tempo de execução. Além disso, a modificação no cálculo do numerador da equação 4.3 fez com que o cálculo do coeficiente resultasse em uma aproximação e não em um valor exato e este fato pode prejudicar o cálculo em algumas situações mesmo que pontuais por ser uma aproximação.

Neste trabalho optou-se por utilizar o coeficiente de correlação cruzada normalizado por suas características robustas com relação à variações em escala e brilho tanto para a imagem quanto para a sub-imagem (o que apresenta uma vantagem sobre o coeficiente sem a normalização) e também por não apresentar um consumo elevado de memória e de pré-processamento (vantagens com relação ao FNCC).

4.3 Trabalhos Relacionados

As principais áreas de pesquisa descritas por este referencial teórico são abordadas em trabalhos recentes, alguns trabalhos de destaque serão apresentados a seguir.

Chati et al. (2007) apresenta um co-projeto de hardware/software para detecção de pontos chave na imagem utilizando FPGA. Este trabalho utiliza o novo paradigma de projeto devido ao aumento da complexidade dos circuitos e busca por melhores resultados. O desenvolvimento de um co-projeto pos-

sibilitou a utilização de um processador de 100MHz cujo processamento de uma imagem relativamente pequena (320x240) consumia um tempo também muito pequeno de 0,8 ms. O trabalho de Assumpção et. Al. (Assumpcao et al. (2007)) utilizou um FPGA para desenvolver um sistema de visão para desvio de obstáculos. O baixo custo das câmeras aliado ao bom desempenho do FPGA justificam o desenvolvimento.

Bonato et. Al. (Bonato et al. (2008)) desenvolveu uma arquitetura de hardware paralelo para um sistema de detecção de características de imagens. Este trabalho também utilizou técnicas de co-projeto de hardware/software e utilizou o *soft-processor* NIOS II em seu desenvolvimento. A arquitetura baseada em FPGA utiliza multi-câmeras e foi desenvolvida para o problema de localização e mapeamento simultâneos em um sistema embarcado, implementando em hardware o filtro de Kalman estendido e detecção de características baseada em algoritmos SIFT.

Para melhorar a utilização das câmeras vários trabalhos de hardware para processamento de imagens foram desenvolvidos, como por exemplo o trabalho de Kalomiros & Lygouras (Kalomiros and Lygouras (2008)) que apresenta um co-processador em hardware para detecção de profundidade em imagens *stereo*. Kalomiros & Lygouras (Kalomiros and Lygouras (2009)) desenvolveram uma arquitetura reconfigurável para detecção de de características pontuais e utilizam em navegação robótica.

O *journal* IEEE *Transactions on Circuits and Systems for Video Technology* em novembro de 2009 apresentou uma edição especial focada apenas na co-exploração de arquiteturas e algoritmos de visão computacional em plataformas atuais de desenvolvimento. O editorial produzido por Chen et. Al. (Chen et al. (2009)) apresentou os artigos e, seguindo uma abordagem semelhante, são citados e descritos apenas os artigos relevantes à este projeto de pesquisa.

Lee et. Al. (Lee et al. (2009)) apresentou um trabalho que analisa o desenvolvimento, o avanço, e o futuro das tecnologias de hardware e software relacionadas a computação visual. Considerando as áreas de codificação de vídeos, processamento de vídeo, visão computacional e computação gráfica o artigo apresenta uma análise das características básicas dos algoritmos e arquiteturas que serão modificadas para acompanhar o avanço tecnológico. A conclusão dos autores indica que a otimização dos algoritmos e das arquiteturas para trabalharem de forma concorrente e não seqüencial e a interação entres os dois componentes do sistema será de grande importância juntamente com a utilização de multicores híbridos. O trabalho de Hubert & Stabernack (Hubert and Stabernack (2009)) apresenta uma abordagem de co-exploração baseada em *profilings*. O trabalho compara a utilização de vários tipos de ferramentas de *profiling* para encontrar os principais pontos críticos do sistema,

com relação às restrições de projeto, e particionar o sistema de forma correta. O desenvolvimento do artigo concentra-se no gargalo apresentado na utilização de memória durante o desenvolvimento de um SoC para codificação de vídeos.

Diretamente relacionado ao trabalho proposto nesta dissertação, o trabalho de Xiantao & Xingbo (Wang and Wang (2009)) apresenta duas arquiteturas para o cálculo do coeficiente de correlação normalizado. No entanto os principais problemas do cálculo do coeficiente não são descritos de forma satisfatória como o cálculo da raiz quadrada e a divisão em ponto flutuante. Os resultados apresentados são animadores porém o hardware não pode ser analisado ou mesmo reproduzido de forma completa sem estas informações e também faltam informações da precisão do *matching*.

A criação de arquiteturas para o cálculo do coeficiente de correlação são apresentadas nos trabalhos de Yonghong (Yonghong (2010)) e Tao et. Al. (Tao et al. (2010)) e apresentam bons resultados, o problema é a perda de precisão numérica que pode afetar o desempenho do algoritmo em casos críticos. As arquiteturas não configuram co-projetos por serem implementadas totalmente em hardware e também o cálculo das raízes não é bem definido. Aplicações recentes da utilização de correlação cruzada, não apenas para o processamento de imagens mas também para outras aplicações, podem ser vistas em Fernandes et al. (2010), Hongcheng and Liu (2010) e Pei et al. (2010).

4.4 Considerações Finais

Considerando o contexto do laboratório de pesquisa onde o desenvolvimento deste trabalho está inserido, este trabalho de pesquisa possui várias aplicações na área de robótica móvel, onde está sendo desenvolvido visando sua utilização junto a métodos de navegação robótica baseados em visão computacional chamados de *Appearance-Based Methods* (Jones et al. (1997), Se et al. (2002), Matsumoto et al. (1996)). Este trabalho foi desenvolvido baseando-se nos conceitos e técnicas apresentadas neste referencial teórico. O algoritmo de cálculo do coeficiente de correlação cruzada normalizado (NCC) foi adotado e mapeado em um co-projeto de hardware/software buscando melhorar o tempo de execução do algoritmo implementado utilizando hardware reconfigurável, no caso, os FPGAs.

O desenvolvimento do co-projeto de hardware/software utilizando o método e o dispositivo escolhidos baseia-se na utilização de várias ferramentas. A tecnologia evoluiu ao longo do tempo resultando no surgimento de várias ferramentas, com intuito de auxiliar os desenvolvedores, como: plataformas que utilizam FPGAs, IDEs para desenvolvimento de hardware e programação

de *soft-processors*, bibliotecas em linguagem de alto nível para simplificar a implementação de algoritmos, dentre outras. O capítulo seguinte apresenta as ferramentas e técnicas que foram utilizadas no desenvolvimento deste projeto.

CAPÍTULO

5

Método e Desenvolvimento

Este projeto de pesquisa tem por objetivo o desenvolvimento de um co-projeto de hardware/software para o algoritmo que calcula a correlação cruzada normalizada de uma imagem digital visando obter um *speedup* com relação ao algoritmo implementado seqüencialmente. Para atingir este objetivo um método de desenvolvimento para co-projeto de hardware/software modificado foi utilizado e o sistema desenvolvido utilizando diferentes plataformas de implementação com FPGAs, um *soft-processor* e seus softwares de desenvolvimento.

5.1 Método Utilizado

Existem vários métodos propostos para execução de um co-projeto de hardware/software baseados no fluxograma apresentado nas figuras 3.4 e 5.1. O método utilizado para o desenvolvimento deste trabalho utiliza uma modificação do método tradicional de co-projeto de hardware/software (DIAS et al. (2010a) DIAS et al. (2010b)),e pode ser visualizado no fluxograma da figura 5.2.

A modificação do método para co-projeto foi utilizada para que o desenvolvimento pudesse se manter adequado ao *time-to-market* e às restrições de projeto. Para este objetivo ser alcançado, o diagrama não deve ser recorrente até o seu início devido ao tempo gasto caso seja necessário voltar e também só deve atingir o nível de desenvolvimento de hardware caso as restrições não sejam atendidas durante o desenvolvimento de software. Inicialmente o algoritmo que representa problema escolhido é implementado em uma linguagem de alto nível em um computador de propósito geral e é obtido seu tempo de

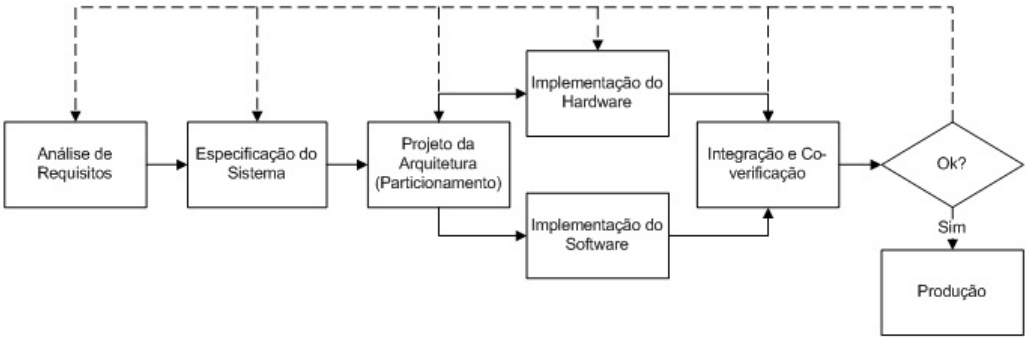

Figura 5.1: Método tradicional para co-projeto de hardware/software.

execução. O hardware do computador de propósito geral utilizado é composto por um processador Intel® Core i3 2.13GHz, 3Gb de memória RAM DDR3 no sistema operacional Microsoft Windows® 7 Home Premium.

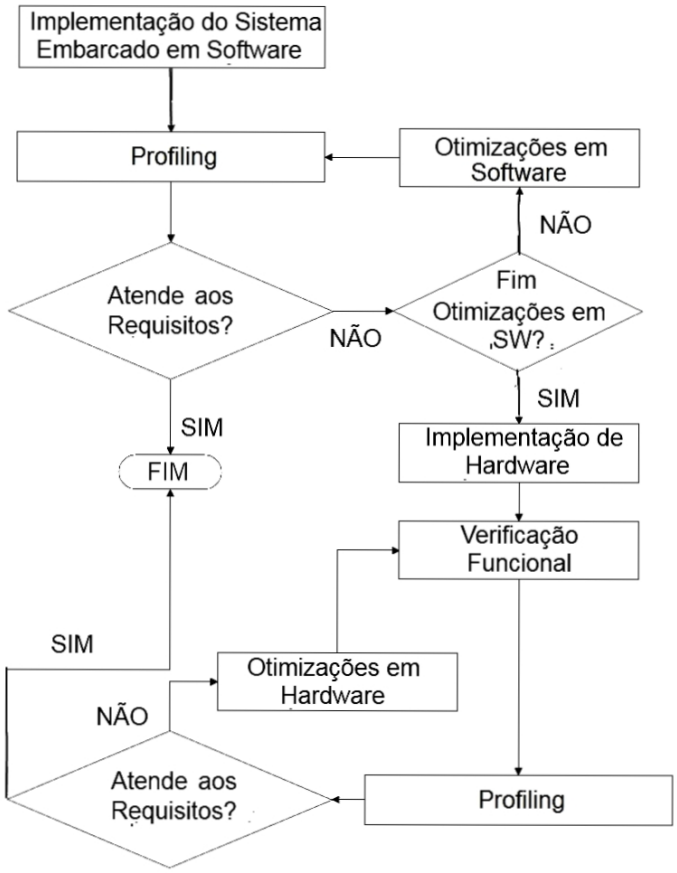

Figura 5.2: Método Modificado (DIAS et al. (2010a) DIAS et al. (2010b)).

Para a implementação do algoritmo no computador de propósito geral foi escolhida a IDE `Code Blocks`[1] em linguagem de programação C. A IDE escolhida utiliza os compiladores da GCC (`Gnu Compiler Collection`) que no caso da linguagem de programação escolhida possui disponível e incluída a

[1]http://www.codeblocks.org/
[2]http://www.gnu.org/software/binutils/

ferramenta de *profiling* `Gnu Profiler` ou Gprof[2]. O objetivo desta primeira implementação é o desenvolvimento do software de forma otimizada para que possa ser implementado diretamente no *soft-processor* consumindo o mínimo de recursos e executando da forma mais rápida possível.

O *profiling* resultante do software é avaliado e algumas modificações já podem ser feitas nesta primeira fase do projeto com objetivo de otimizar o software. Este passo pode ser considerado um pré-desenvolvimento pois ainda não configura a implementação do sistema embarcado em software como sugere o primeiro passo do método. Após modificar o algoritmo, é necessária a escolha de uma plataforma de desenvolvimento que possibilite a implementação do algoritmo em um *soft-processor* e também o desenvolvimento de hardware caso seja necessário.

O primeiro passo do método de desenvolvimento escolhido é a implementação do sistema embarcado totalmente seqüencial em software e avaliação de seu *profiling*. Este passo torna-se menos árduo considerando a implementação do algoritmo previamente no computador de propósito geral devido ao fato de que o *soft-processor* escolhido (Nios II) permite ser programado em linguagem C. Após a execução do algoritmo seu *profiling* é analisado e este é o segundo passo do método. O primeiro ciclo do método consiste em ajustes no próprio software para otimizar o tempo de execução e, caso não seja mais possível otimizar o tempo de execução apenas com modificações no software (e.g. modificações de tipos de variáveis, mudança em utilização de operadores), o desenvolvimento de hardware é iniciado.

As otimizações em hardware, caso sejam necessárias, são feitas baseadas no resultado do *profiling* do software. O resultado mostra o tempo de execução para cada linha do algoritmo analisado e os gargalos são então identificados. O segundo ciclo do método consiste então no desenvolvimento de hardware para atacar os gargalos do algoritmo visando melhorar seu tempo de execução e todo hardware desenvolvido será incluído como uma ou mais instruções customizadas do *soft-processor*. Caso os requisitos de tempo de execução sejam atingidos o ciclo se encerra e o sistema então pode ser considerado terminado. O tempo de duração de cada ciclo durante o desenvolvimento pode ser modificado de acordo com as restrições do próprio tempo de desenvolvimento do sistema. A seguir serão apresentadas as técnicas e as ferramentas utilizadas durante o desenvolvimento do projeto.

5.1.1 Técnicas e Ferramentas

A avaliação do algoritmo desenvolvido incluirá o efeito da modificação do tamanho das imagens e sub-imagens no sistema implementado no *soft-processor* Nios II. Primeiramente serão analisados os tempos de execução para a me-

mória *On-Chip* (equivalente a memória *cache*) do processador escolhido e em seguida, caso seja atingido um tempo de execução aceitável, o sistema irá incluir a implementação de uma memória RAM. A utilização de uma memória RAM implica em um atraso no tempo de execução causado pelo atraso no acesso aos dados em comparação com a memória *On-Chip* que possui tempo de acesso reduzido por estar dentro do processador.

O método escolhido para o desenvolvimento deste projeto é um método de co-projeto de hardware/software baseado em *profiling*. Analisando as ferramentas de *profiling* apresentadas no capítulo 3 deste documento quanto a viabilidade de utilização, flexibilidade, tipo de profiling disponíveis dentre outras características a ferramenta escolhida foi o `Gnu Profiler`. Esta ferramenta de *profiling* possui várias características desejáveis e a principal delas é estar disponível tanto no compilador GCC para computadores de propósito geral quanto para os softwares executados no *soft-processor* Nios II que são compilados por uma versão modificada do mesmo compilador.

No caso do método de *profiling* que será utilizado, *Flat Profile*, apresenta o tempo de execução de cada linha de código do software permitindo assim uma análise dos gargalos do algoritmo e conseqüentemente indica como melhorar o tempo de execução. Todos os tempos de execução são mostrados em segundos e são analisadas inclusive funções de sistema. A figura 5.3 mostra o exemplo de um *Flat Profile*.

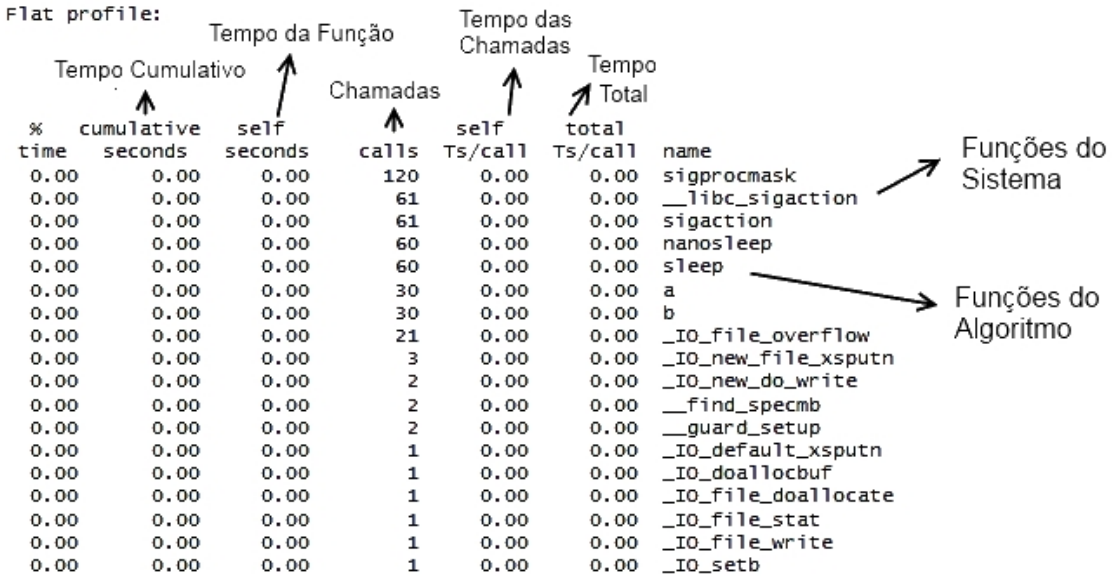

Figura 5.3: Exemplo de *Flat Profile*.

Após descobrir quais os gargalos do algoritmo, iniciam-se as otimizações do software. As otimizações utilizadas consistem em técnicas conhecidas e aplicadas em co-projetos de sistemas embarcados em geral (Wolf (2006)). Compiladores atualmente possuem em sua implementação técnicas avançadas de

otimização de software que são utilizadas mesmo quando as opções de otimização não são selecionadas, porém a modificação no código previamente também pode atingir resultados satisfatórios.

Após o desenvolvimento do algoritmo foram selecionadas as seguintes técnicas para otimização de software: (i) utilizar operações básicas ao invés de funções da linguagem de programação (modificar $pow(x, 2)$ por $x * x$); (ii) utilizar operadores comuns para atribuições simples (modificar $x+ = y$ por $x = x + y$); (iii) sempre que possível utilizar variáveis sem sinal; (iv) utilizar a técnica conhecida como *loop unrolling* que consiste em desmembrar o loop de forma que ao invés de uma operação por incremento sejam feitas duas ou mais e, por fim, (v) utilização de vetores ao invés de matrizes para armazenamento da imagem. Após aplicadas as técnicas descritas no software o desenvolvimento do hardware será feito.

Foram escolhidas para este projeto de pesquisa duas plataformas contendo FPGAs da fabricante Altera (figura 5.4): Terasic DE2-70[3] (Altera Cyclone II) e Arrow BeMicro[4] (Altera Cyclone III). A escolha se deu pelos seguintes motivos: (i) no caso plataforma DE2-70 existe um número considerável de entradas e saídas que possibilitam inclusive a aquisição de imagens por um sensor óptico que pode ser adicionado à plataforma, e no caso da plataforma Arrow BeMicro a alimentação é feita através da porta USB sendo uma alternativa interessante com relação ao consumo de energia que por padrão é 5V; (ii) ambas as plataformas são de baixo custo; (iii) os FPGAs escolhidos da fabricante Altera permitem a configuração do *soft-processor* Nios II de forma gratuita e este processador permite a adição de até 256 instruções customizadas, programação em linguagem C e possui a ferramenta de *profiling* Gprof implementada para ser utilizada em softwares desenvolvidos para o processador; (iv) o laboratório onde será desenvolvido o projeto (LRM-USP) possui todos estes equipamentos disponíveis. Outra característica importante da plataforma DE2-70 é o fato de que existem vários exemplos de desenvolvimento de hardware para seus dispositivos de entrada e saída (tanto os presentes na plataforma quanto uma câmera[5] e um *display touch-screen*[6]) devido ao tempo que a plataforma se encontra no mercado.

A utilização do *soft-processor* NIOS II apresenta algumas vantagens dentre elas: existem vários sistemas operacionais desenvolvidos para o processador, inclusive o mLinux que é um sistema baseado em linux de tempo real (RTOS - *Real Time Operational System*); possibilita a utilização de vários núcleos de processamento em um FPGA caracterizando *multi-cores*; possui acesso aos pe-

[3]http://csg.csail.mit.edu/6.375/handouts/other/DE2Manual.pdf
[4]http://www.arrownac.com/offers/altera-corporation/bemicro/getting-started.html
[5]http://www.terasic.com.tw/attachment/archive/282/TRDB_D5M_UserGuide.pdf
[6]http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=213

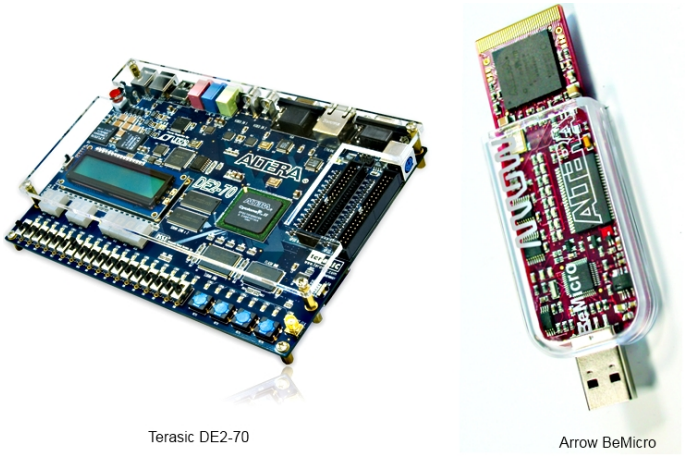

Figura 5.4: Plataformas de Desenvolvimento.

riféricos como *câmera e touch-screen*; permita avaliar o desempenho do código com várias ferramentas.

O *soft-processor* Nios II é disponibilizado em três versões diferentes pela fabricante Altera que são: *economic* (Nios II /e), *standard* (Nios II /s) e *fast* (Nios II /f). Os códigos VHDL da versão Nios II /e foram disponibilizados pela fabricante em dezembro de 2009. Existem algumas diferenças básicas entre os três processadores e a tabela 5.1 apresenta algumas destas diferenças.

Tabela 5.1: Comparativo de processadores Nios II

	Nios II /e	Nios II /s	Nios II /f
Licença para Utilização Comercial	Livre	Paga	Paga
Jtag Debug	Sim	Sim	Sim
Instruções Customizadas	256	256	256
Multiplicação, Divisão e Deslocamento em HW	Não	Sim	1-Ciclo
Previsão de Desvio	Não	Estático	Dinâmico
Pipeline	Não	Cinco Estágios	Seis Estágios
Cache	Não	Instrução	Dados e Instrução
Unidade de Ponto Flutuante	Não	Não	Não

A otimização dos processadores ocorre basicamente no tempo de execução dos algoritmos porém a área e elementos lógicos utilizados aumentam significativamente, onde para o processador Nios II /e (mais simples) são utilizados bem menos elementos lógicos, apenas algo em torno de 700 LEs. O barramento principal do processador chamado de AVALON é baseado em endereços e, ao contrário da forma que os barramentos são normalmente implementados, permite que vários dispositivos mestres atuem simultaneamente devido a presença de um árbitro.

Todos os processadores mencionados apresentam a possibilidade de inclusão de hardware customizado que pode ser feita de 3 maneiras diferentes. A primeira delas é quando o hardware é desenvolvido totalmente fora do processador e se comunica com a CPU utilizando o barramento AVALON. A segunda maneira é quando o hardware desenvolvido é colocado dentro do processador, porém fora da ULA, e também utiliza o barramento AVALON para comunicação. A terceira e mais eficiente maneira de incluir um hardware no processador Nios II é através de instruções customizadas (Joost and Salomon (2006)).

A última opção cria um barramento de dados paralelo a Unidade Aritmética Lógica (UAL) da CPU (figura 5.5) e durante a geração do sistema, instruções especiais de acesso ao hardware são adicionadas ao conjunto de instruções do processador. O fato do hardware estar presente dentro do processador diminui consideravelmente o tempo de comunicação e permite um ganho considerável de desempenho. As principais limitações são o barramento de dados que é de 32 bits e também o fato de que o hardware desenvolvido deverá ser adaptado caso seja incluído em outros projetos que não utilizarem o Nios II.

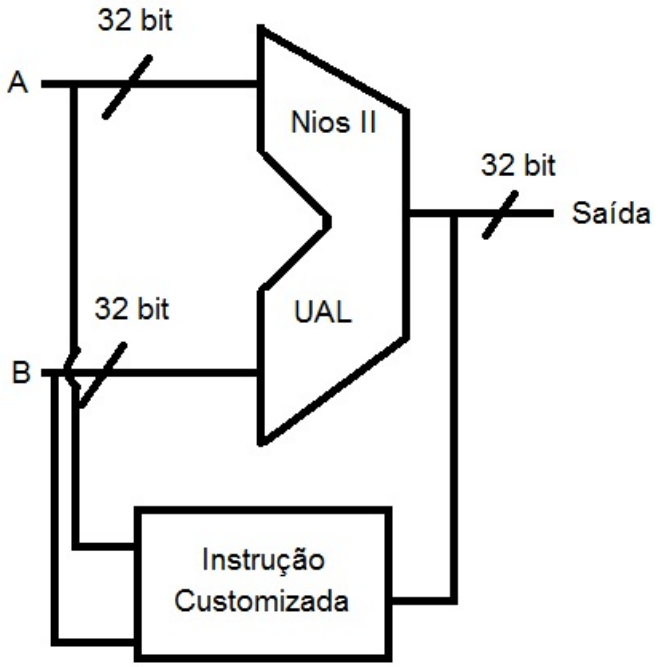

Figura 5.5: Instrução Customizada.

A fabricante Altera também disponibiliza duas IDEs para o desenvolvimento de hardware em seus FPGAs: `Quartus II`[7] e `Nios II` IDE[8]. A IDE Quartus II possui todas as ferramentas necessárias para o desenvolvimento de hardware para os FPGAs e também, até a versão utilizada neste projeto (9.1), possui um simulador de formas de onda. Toda a configuração do hardware do *soft-processor* Nios II também é feita no Quartus porém por uma IDE

[7]http://www.altera.com/literature/lit-qts.jsp
[8]http://www.altera.com/literature/lit-nio2.jsp

chamada `SOPC Builder`. Neste software é feito todo o fluxo de design em FPGAs descrito no capítulo 2. No caso do *design* e da simulação funcional o usuário é responsável pelo desenvolvimento, e as etapas de síntese, *place and route*, configuração do *bitstream* e programação do dispositivo são feitas de forma automática. As operações que são feitas de forma automática podem ser configuradas para otimização de variáveis como por exemplo área utilizada e tempo de execução.

O `Nios II` IDE agrega todas as ferramentas necessárias para o desenvolvimento de software para o *soft-processor* Nios II que for configurado pelo `SOPC Builder`. As fases de programação, compilação, simulação, comunicação com o FPGA e execução do software no hardware configurado é feita através deste software. Todo material relacionado aos softwares de desenvolvimento, download dos softwares e informações sobre como adquirir as plataformas podem ser encontradas no site da fabricante[9].

5.2 Considerações Finais

Atualmente os co-projetos de hardware/software possuem várias ferramentas e métodos que auxiliam o seu desenvolvimento, e este capítulo apresentou as ferramentas e o método escolhido para o desenvolvimento deste projeto de pesquisa. Sendo assim, as características apresentadas justificam a utilização destas ferramentas, além do fato de que o Laboratório de Robótica Móvel (LRM) possui todas estas ferramentas e infra-estrutura à disposição para o desenvolvimento do projeto. O capítulo seguinte apresenta os resultados de todas as etapas do desenvolvimento desde as primeiras análises do algoritmo durante sua implementação no computador de propósito geral até o *speedup* final após o desenvolvimento do hardware. Os resultados são apresentados e analisados simultaneamente e são seguidos pelas conclusões do trabalho.

[9] http://www.altera.com

CAPÍTULO

6

Resultados

6.1 Introdução

Os resultados dos experimentos realizados seguem o fluxograma de experimentação ilustrado na figura 6.1

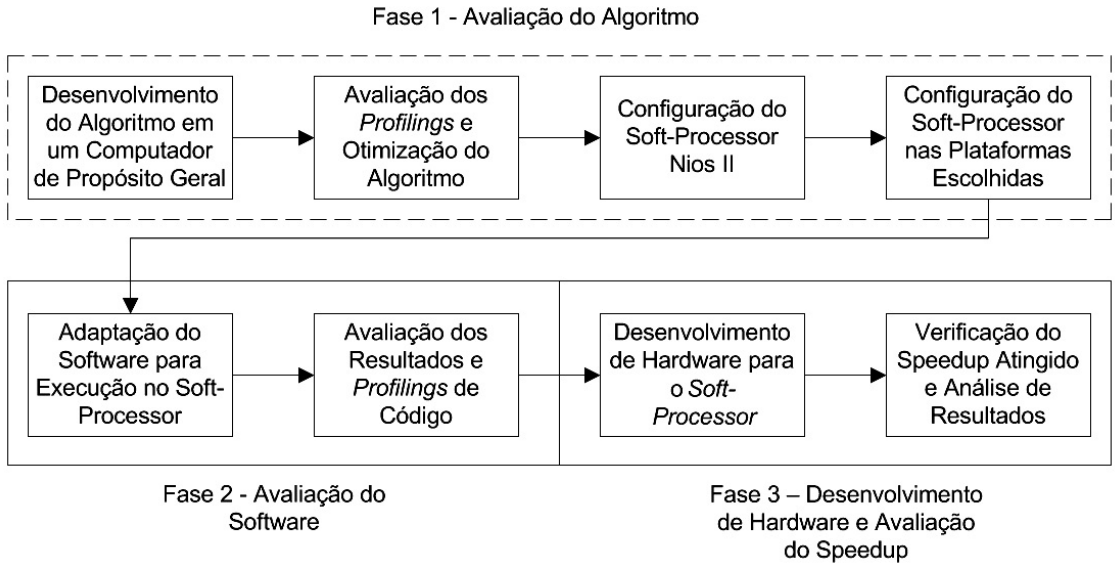

Figura 6.1: Fluxograma de Experimentação.

Inicialmente o algoritmo foi desenvolvido para o processador de propósito geral. Esta implementação permitiu a análise do *profiling* do algoritmo e o teste de otimizações de software. Após as análises iniciais, experimentos foram feitos com relação à influência da variação do tamanho da imagem e da sub-imagem no tempo de execução do algoritmo.

Após a fase inicial, o algoritmo foi desenvolvido para o *soft-processor* Nios

II e as mesmas otimizações de software foram analisadas através do *profiling* do algoritmo. Vários processadores foram configurados inicialmente (Nios II /e Economic, Nios II /s Standard, Nios II /f Fast, Nios II /*f FPU, Nios II /*fd FPU + Divisão em Hardware) e o profiling do algoritmo foi feito para cada um deles nas duas plataformas escolhidas com as otimizações de software. O tempo de execução determinou a utilização da plataforma DE2-70 e então as otimizações em hardware foram desenvolvidas nesta plataforma (inclusão da unidade de ponto flutuante, aumento da memória através da inclusão de memória SDRAM, aumento do *clock* através da PLL e a inclusão de um hardware dedicado) e o *speedup* das implementações foi calculado através do *profiling* nos *soft-processors* configurados.

A seções seguintes descrevem os passos do fluxograma de experimentação adotado para este projeto apresentando os resultados bem como sua análise. A síntese dos processadores utilizou apenas a otimização para atingir o clock desejado e o softwares foram compilados com otimização para o tamanho final do executável baseado em testes anteriores que não apresentaram mudanças significativas quando otimizados para tempo de execução no caso do GCC para o *soft-processor* Nios II compilando o código desenvolvido.

6.2 Desenvolvimento para o Processador de Propósito Geral

6.2.1 Implementação de Software

A implementação do algoritmo da correlação cruzada normalizada foi feita em linguagem C para proporcionar maior portabilidade ao ser transferido para o *soft-processor* Nios II. A imagem é considerada uma matriz de pixels em escala de cinza (*grayscale*) com valores de 0 a 255. Esta representação, ilustrada pela figura 6.2, utiliza 8-bits por pixel o que proporciona também um valor aceitável para armazenamento e processamento em hardware.

A sub-imagem utilizada para o cálculo da correlação é posicionada no canto superior esquerdo da imagem e varre toda a imagem, pixel a pixel. O problema das bordas foi resolvido de forma a não expandir a imagem portanto a janela limita-se a área da imagem. Os métodos de navegação robótica que utilizam este algoritmo também utilizam um banco de dados de imagens sendo que a probabilidade da sub-imagem ou imagem a ser encontrada esteja deslocada de forma que prejudique a abordagem proposta (posicionada parcialmente e apenas na borda da imagem) é baixa. A primeira implementação do algoritmo foi feita em sua forma clássica para que fosse possível o estudo do *profiling* e das implementações de otimizações de software. Considerando que o resul-

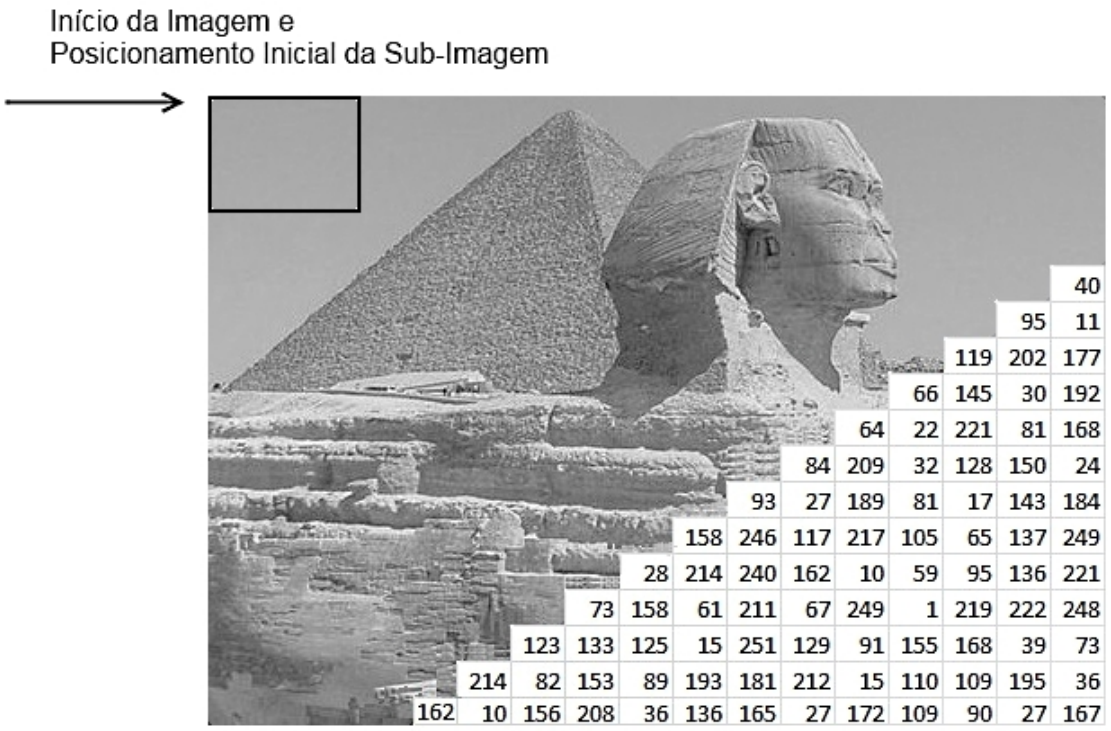

Figura 6.2: Exemplo de imagem de entrada.

tado do algoritmo já foi exaustivamente testado na literatura, este trabalho busca a otimização de seu tempo de execução.

6.2.2 Análises e Otimizações

A implementação inicial do software foi desenvolvida, como descrito no capítulo anterior, em um computador de propósito geral. Ao analisar o *profiling* dos códigos gerados pela ferramenta `Gprof` foi possível perceber que o gargalo do algoritmo era exatamente o cálculo do coeficiente que possui partes da função que envolviam cálculos em ponto flutuante. A figura 6.3 demonstra o *profiling* inicial do código onde os cálculos fixos são as médias dos valores da imagem utilizadas no cálculo do coeficiente.

Os resultados da execução do algoritmo para vários tamanhos de imagens e sub-imagens demonstrou que existe uma curva crescente que relaciona o tempo de execução do algoritmo com a variação do tamanho da imagem para um tamanho fixo de sub-imagem. Os resultados demonstram ainda que a curva que relaciona o tempo de execução do algoritmo fixando-se a imagem e variando o tamanho da sub-imagem possui um limiar onde o tempo de execução que era crescente passa a ser decrescente.

A existência do limiar justifica-se pelo fato de que o número de operações para percorrer a imagem diminui com o crescimento da sub-imagem. Os gráficos apresentados nas figuras 6.4 e 6.5 demonstram estes comportamen-

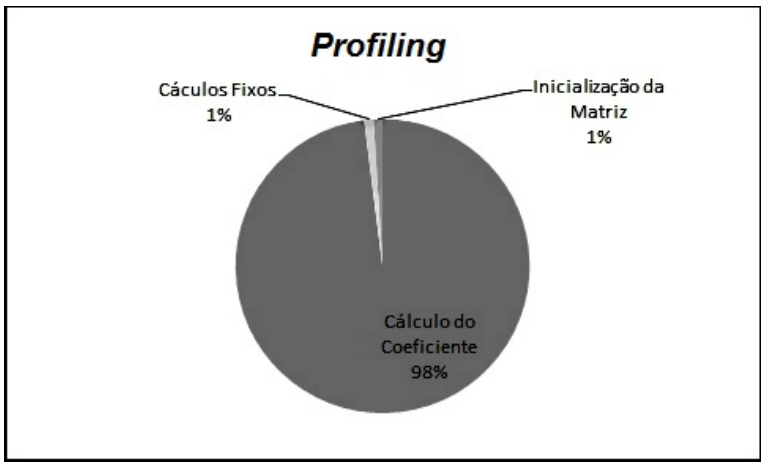

Figura 6.3: *Profiling* do código.

tos. Enquanto o comportamento do gráfico 6.5 apresenta-se como esperado, o comportamento do gráfico da figura 6.4 apresenta uma queda acentuada após um certo limiar. Este limiar representa o momento em que a sub-imagem apresenta um tamanho que favorece o cálculo do coeficiente tanto em número de operações como em número de vezes que o cálculo é efetuado.

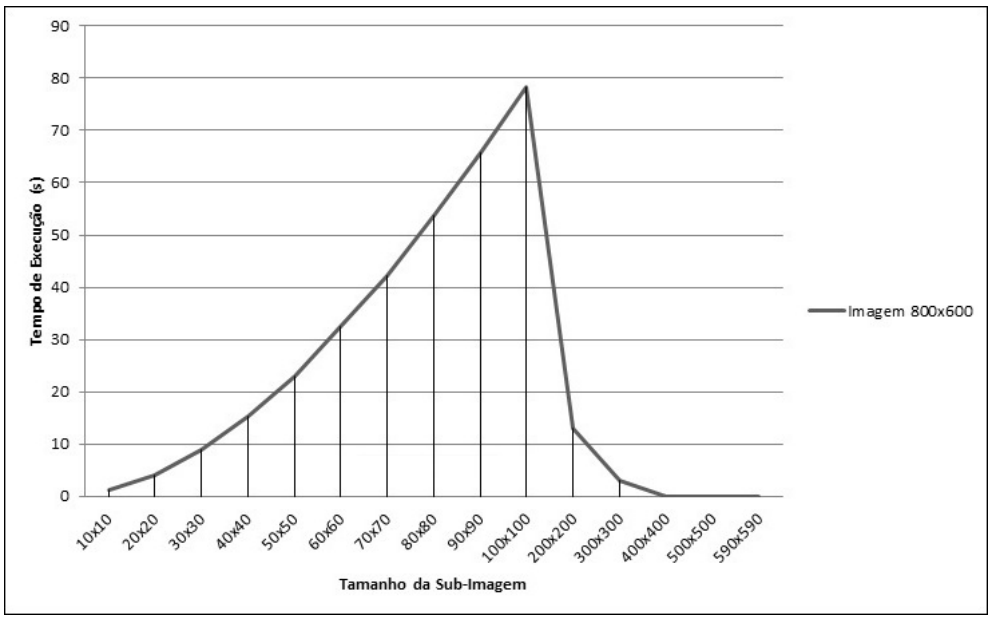

Figura 6.4: Execuções fixando o tamanho da imagem e variando o tamanho da sub-imagem.

Considerando que o hardware do computador de propósito geral não pode ser modificado, foram implementadas otimizações de software com intuito de diminuir ainda mais o tempo de execução conseguindo atingir um *speedup* ainda nesta etapa do desenvolvimento. Existem otimizações de software que são aplicáveis no desenvolvimento para sistemas embarcados como descrito em Wolf (2006), e que foram consideradas neste trabalho.

Após a análise de código e estudo das otimizações propostas as seleciona-

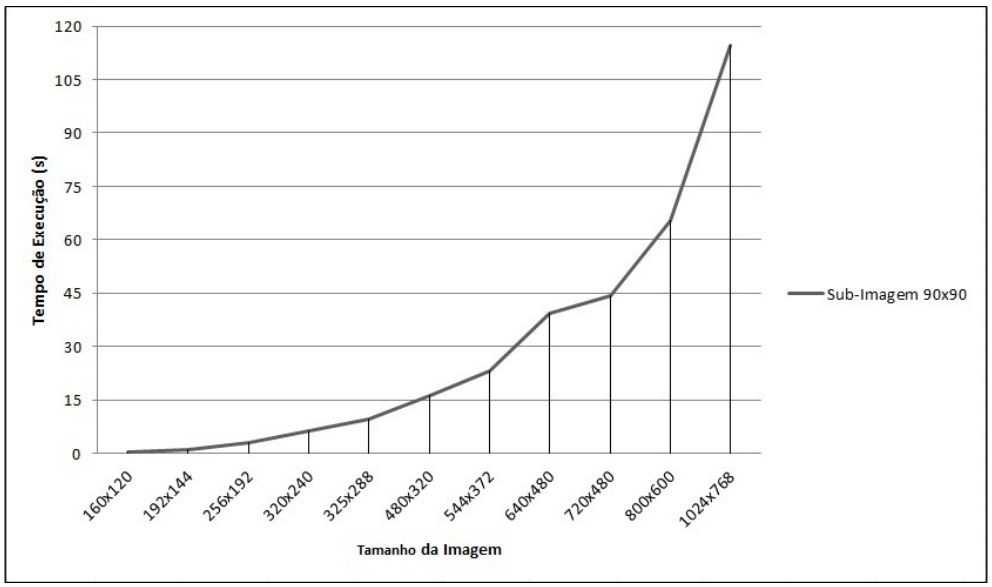

Figura 6.5: Execuções fixando o tamanho da sub-imagem e variando o tamanho da imagem.

das para implementação foram:

- Variáveis sem sinal.
- Utilização de expressões equivalentes a funções, como por exemplo, $x * x$ ao invés de $pow(x, 2)$.
- Mudança na forma de escrita de operadores entre $x+ =$ e $x = x+$, verificando se a alteração afeta o desempenho;
- *Loop Unrolling*. Executar mais de uma operação do loop por iteração.

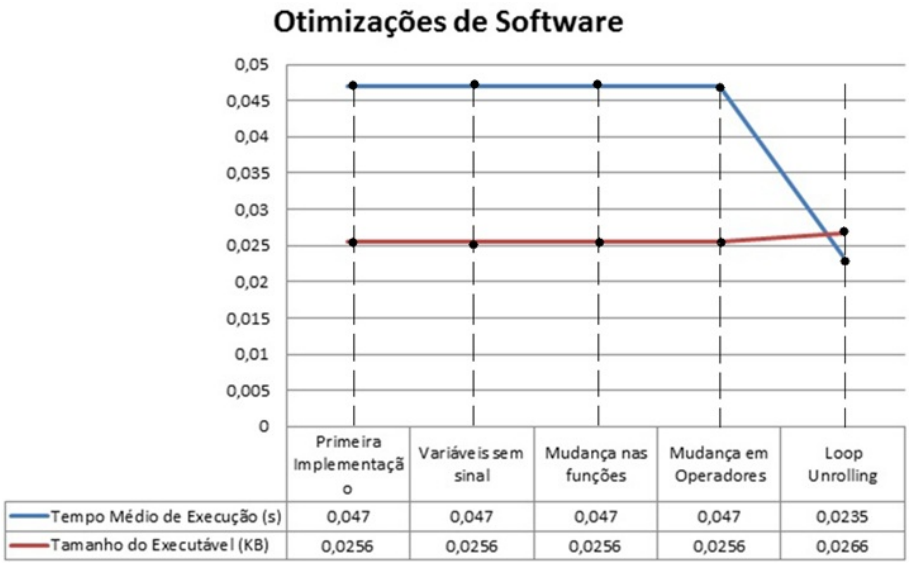

Figura 6.6: Otimizações de software.

O gráfico da figura 6.6 apresenta os tempos de execução após as otimizações de software. Dentre as otimizações escolhidas, a única que apresentou um resultado relevante foi o *loop unrolling*. A técnica diminuiu o tempo de execução do algoritmo em 50% e aumentou o tamanho do executável em 0,001KB. O aumento no consumo de memória não é significativo com relação ao ganho no tempo de execução justificando a utilização da técnica. As outras técnicas não apresentaram resultados significativos pois o compilador já implementa este tipo de otimizações automaticamente. Por fim, a tabela 6.1 apresenta os tempos de execução para variados tamanhos de imagens e de janelas. Na tabela, as linhas são o tamanho da imagem, as colunas são o tamanho da sub-imagem e os tempos são representados em segundos.

Tabela 6.1: Tempos de Execução para GPP, onde as linhas são o tamanho da imagem e as colunas são o tamanho da sub-imagem, e os tempos são representados em segundos.

	40x40	50x50	60x60	70x70	80x80	90x90	100x100	130x130
160x120	0,35	0,45	0,51	0,53	0,47	0,4	0,3	
192x144	0,59	0,78	0,92	1,02	1,07	1,02	0,93	0,01
256x192	1,19	1,64	2,11	2,57	2,85	3,18	3,31	0,53
320x240	2,04	2,99	3,75	4,71	5,55	6,25	7,15	1,4
352x288	2,84	4,04	5,42	7,03	8,17	9,5	10,68	4,47
480x320	4,46	6,51	8,82	11,4	13,83	16,35	18,83	8,37
544x372	6,01	8,9	12,28	15,77	19,47	23,22	27,11	37,19
640x480	9,43	14,12	19,46	26,03	32,07	39,29	46,08	110,79
720x480	10,63	16,11	22,2	29,55	36,67	44,35	52,8	130,48
800x600	15,19	22,93	32,14	42,33	53,24	65,49	78,25	162,32
1024x768	25,71	39,09	54,18	73,17	92,54	114,55	873,41	> 1000

Esta parte do método tem por objetivo guiar o desenvolvimento inicial do hardware que será utilizado para implementar o sistema. A proposta do trabalho é utilizar o *soft-processor* Nios II para implementar o método de co-projeto descrito no capítulo anterior. A seção seguinte descreve a arquitetura dos processadores configurados para as duas plataformas escolhidas para este trabalho.

6.3 Desenvolvimento para o Soft-Processor

6.3.1 Configuração dos Soft-Processors

As plataformas de desenvolvimento BeMicro[10] e DE2-70[11] escolhidas para este trabalho possuem FPGAs da fabricante Altera. Existem comparativos[12] entre os FPGAs desenvolvidos pela fabricante e analisando os comparativos a família Stratix e Hard Copy são as famílias de alto desempenho e alto custo

seguidas pela família Cyclone que é de baixo desempenho e baixo custo também.

O FPGA presente na plataforma Arrow BeMicro é o Cyclone III da fabricante Altera. Este FPGA não pertence a uma linha de ponta portanto possui recursos limitados. Juntamente com este fato, aliado ao tamanho e ao tipo de alimentação e comunicação (USB), a plataforma foi desenvolvida com recursos limitados apresentando por exemplo um oscilador com clock máximo de 16MHZ e apenas 4MB de memória SRAM disponível. Estas características fazem com que o sistema desenvolvido para a plataforma BeMicro contenha componentes diferentes em comparação com o sistema desenvolvido para a plataforma DE2-70. Presente na plataforma DE2-70, o FPGA Cyclone II possui maior número de elementos lógicos e estão disponíveis dois *chips* com 32MB de memória SRAM e também um oscilador com clock máximo de 50MHZ.

Considerando as características das plataformas escolhidas serão apresentadas as arquiteturas básicas utilizadas na configuração dos *soft-processors* para a execução do software.

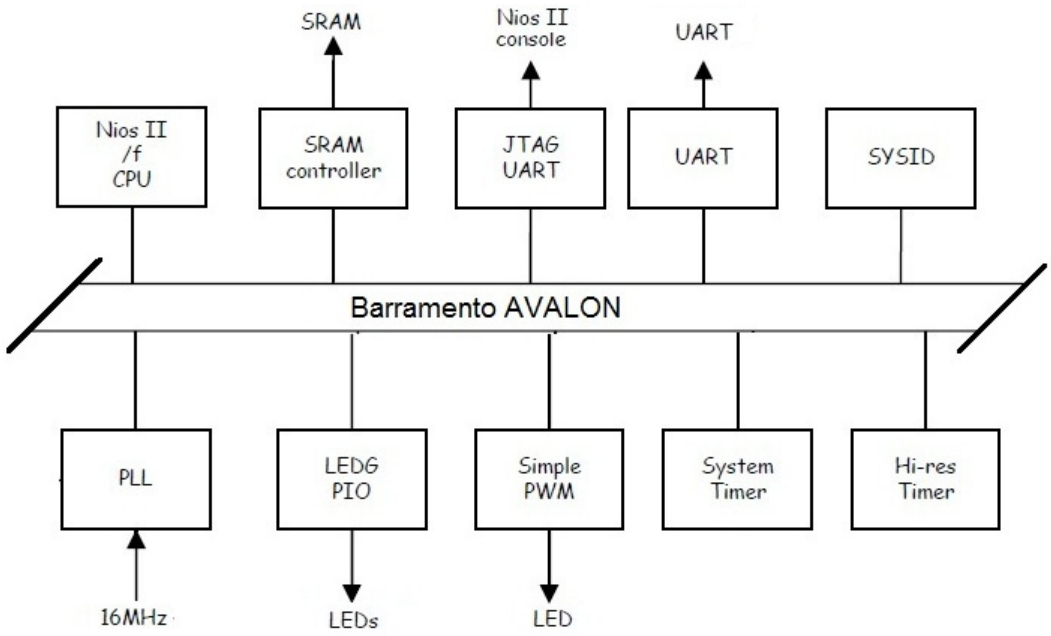

Figura 6.7: Arquitetura dos processadores para a plataforma BeMicro.

A figura 6.7 apresenta os componentes utilizados na arquitetura dos processadores da plataforma BeMicro. A arquitetura apresenta primeiramente o núcleo do processador neste caso representado pela versão *fast* do *soft-processor* Nios II (/f). Existe também um controlador para a utilização da memória RAM (*SRAM controller*), o dispositivo de comunicação do *soft-processor*

[10]http://www.altera.com/b/nios-bemicro-evaluation-kit.html
[11]http://www.altera.com/education/univ/materials/boards/de2-70/unv-de2-70-board.html
[12]http://www.altera.com/literature/ds/ds_nios2_perf.pdf

com o computador (JTAG UART), um dispositivo de comunicação serial (UART), a identificação do sistema para programação (SYSID), um dispositivo *Phase-Locked Loop* (PLL) para que o clock atinja os 50MHZ, para controlar os LEDs um dispositivo de entrada e saída paralela (LEDs PIO), dois *timers* sendo um para o *clock* do sistema e um para medições de tempo de execução (*System Timer* e *Hi-res Timer*) e, por fim, um dispositivo de controle de largura de pulso ligado a um LED (*Simple* PWM).

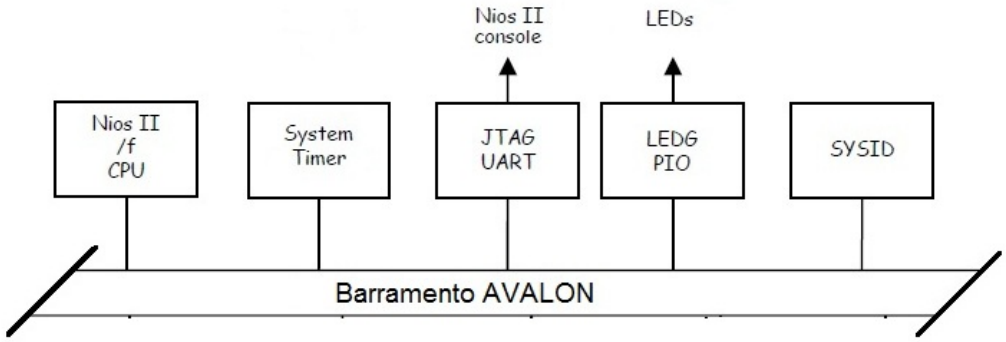

Figura 6.8: Arquitetura dos processadores para a plataforma DE2-70.

As arquiteturas desenvolvidas para a plataforma DE2-70 utilizam os componentes presentes na figura 6.8. Esta arquitetura apresenta: o núcleo do processador neste caso representado pela versão *fast* do *soft-processor* Nios II (/f), o dispositivo de comunicação do *soft-processor* com o computador (JTAG UART), a identificação do sistema para programação (SYSID), para controlar os LEDs um dispositivo de entrada e saída paralela (LEDs PIO) e um timer para o *clock* do sistema.

A diferença dos componentes para as duas plataformas é consequência dos modos de comunicação necessários para sua utilização e também pelo fato de que, considerando que o *clock* atingível pela plataforma BeMicro é 50MHz, a comparação dos sistemas deve ser feita inicialmente com o mesmo *clock*. Considerando o tempo de execução do sistema, é possível melhorar o clock incluindo um dispositivo *Phase-Locked Loop* (PLL) na arquitetura da plataforma DE2-70 elevando seu clock para 100MHz caso o caminho crítico do hardware permita esta modificação. Os dispositivos ligados aos LEDs são implementados para verificação através de saída de dados.

Seguindo as arquiteturas-base definidas acima, os hardwares foram configurados e programados nas plataformas. Através do conhecimento prévio do problema adquirido na fase de implementação em um computador de propósito geral foi possível diminuir o tempo de desenvolvimento com a implementação prévia de 9 diferentes tipos de processadores. Os 9 tipos são constituídos por: 3 processadores básicos (Nios /e, /s e /f), os mesmos 3 processado-

Tabela 6.2: Comparação dos dados de síntese (elementos lógicos (LE), blocos de memória (MB), pinos (P)) entre as duas plataformas BeMicro (*b*) e DE2-70 (*d*).

	*b*LE	*b*MB	*b*P	Clock	*d*LE	*d*MB	*d*P
Nios II /e	2240	0,26	77	50	2052	0,79	6
Nios II /s	3250	0,30	77	50	2736	0,82	6
Nios II /f	4160	0,31	77	50	2736	0,84	6
Nios II /ef	3520	0,26	77	50	2736	0,79	6
Nios II /sf	4960	0,30	77	50	4104	0,82	6
Nios II /ff	5600	0,31	77	50	4789	0,84	6
Nios II /efd	8480	0,26	77	50	7525	0,79	6
Nios II /sfd	-	-	-	50	8894	0,82	6
Nios II /ffd	-	-	-	50	9578	0,84	6

res com unidades de ponto flutuante (*Floating-Point Unit* ou FPU) para suprir a necessidade das operações de adição, subtração e multiplicação em ponto flutuante (denotados por /ef, /sf e /ff) e, por fim, novamente os mesmos 3 processadores com a diferença que a unidade de ponto flutuante inclui a divisão de ponto flutuante em hardware (denotados por /efd, /sfd e /ffd). Estes processadores são configurados através da ferramenta `SOPC Builder`[13] disponibilizada pela fabricante dos FPGAs Altera.

A tabela 6.2 apresenta comparação dos dados de síntese e os 9 processadores (Nios II /e,/s e /f; Nios II /xf - que representa a utilização da FPU e Nios II /xfd - que representa a utilização da FPU com o divisor de hardware). O caminho crítico de todos os processadores apresentou resultados próximos de *clock* a 100MHz com variações entre 101MHz e 109MHz permitindo portanto a utilização de um clock de até 100MHz.

Durante a implementação dos processadores ocorreu um problema com a plataforma BeMicro. A ferramenta responsável pela parte de *fitting* do software Quartus II não foi capaz de gerar o *netlist* com os elementos lógicos disponíveis no FPGA Cyclone III para a versão *fast* (/f) do *soft-processor* que incluía a unidade de ponto flutuante. Sendo assim as duas últimas versões presentes na tabela 6.2 só possuem dados de síntese para a plataforma DE2-70.

Os dados de síntese presentes na tabela 6.2 sugerem uma análise com relação a alguns pontos relevantes. O processador *economic* (/e) consome uma quantidade consideravelmente menor de área porém seu tempo de execução tende a ser pior em comparação aos outros dois tipos de processadores. Considerando ainda o consumo de área entre o mesmo FPGA, os processadores /s e /f também possuem um consumo de área com diferença significativa o que sugere a utilização da versão /s caso não haja diferença significativa nos tempos de execução.

[13]http://www.altera.com/literature/ug/ug_sopc_builder.pdf

A porcentagem ocupada do FPGA Cyclone III (BeMicro) dificilmente irá permitir o desenvolvimento de um hardware de alta complexidade como instrução customizada pra ser incluído no processador. A inclusão do hardware para divisões em ponto flutuante aumenta o consumo de área e elementos lógicos de forma que caso o tempo de execução não seja significativamente mais rápido, sua utilização pode ser desconsiderada. Apesar dos dados de síntese de maior porcentagem de ocupação, a plataforma BeMicro possui a vantagem de ser alimentada com apenas 5V (padrão USB) enquanto a plataforma DE2-70 é alimentada com uma fonte de 12V.

Após as configurações iniciais dos soft-processors que serão utilizados durante o co-projeto, o próximo passo é a transição do código desenvolvido para o computador de propósito geral para o *soft-processor* Nios II.

6.3.2 Implementação de Software

O *soft-processor* Nios II, dentre as vantagens apresentadas no capítulo anterior, possui um compilador de software baseado no compilador GCC. Este fato auxilia de forma considerável na transição de códigos desenvolvidos para processadores de computadores de propósito geral. O compilador desenvolvido para o *soft-processor* Nios II utiliza a linguagem C no padrão ANSI/ISO aumentando ainda mais a portabilidade de código.

A consequência direta das características descritas foi uma transição sem incompatibilidades, inclusive na parte de entrada e saída pois o software Nios II IDE possui uma interface com o processador (que estabelece a comunicação através do componente JTAG UART) que permite a utilização de funções de entrada e saída de dados normalmente.

Para a execução do software nos *soft-processors* configurados foi necessário modificar o tamanho máximo das imagens considerando o tamanho da memória disponível no hardware e também para que a análise do tempo de execução inicialmente pudesse ser feita em menor tempo. A questão de alocação de memória não foi analisada pois toda a memória utilizada é alocada estaticamente.

O tempo de execução do algoritmo, como sugere o método de co-projeto adotado neste trabalho, foi medido através do *profiling* das execuções do código. O sistema considerado inicial para o método é o *soft-processor* Nios II /e sem implementação de hardware para ponto flutuante. Desta maneira podem ser analisados os impactos das melhorias de desenvolvimento dos *soft-processors* (com relação a inclusão de *pipeline*, aceleradores de hardware para operações de ponto fixo dentre outras características apresentadas na tabela 5.1) juntamente com o *speedup* proporcionado pela FPU apresentando ou não o hardware responsável pela divisão de ponto flutuante. Desta forma os re-

sultados analisados são válidos para *soft-processors* em geral e podem auxiliar inclusive em seu desenvolvimento. A análise do *profiling* da execução do algoritmo nos *soft-processors* é apresentada na seção seguinte.

6.3.3 Análise e Otimizações do Software

A execução e análise do software embarcado inicialmente indica qual o tempo de execução de cada processador e a viabilidade de utilização de cada processador para o problema abordado em ambas as plataformas. Outra informação importante da análise é o comportamento do algoritmo no processador embarcado com relação ao aumento da imagem e ao aumento da sub-imagem, dois fatores impactantes para o algoritmo no processador de propósito geral que podem ter seus efeitos atenuados ou elevados. Este estágio equivale ao início da fase 2 do fluxograma de experimentação da figura 6.1 e ao primeiro ciclo de desenvolvimento referente às otimizações de software do diagrama presente na figura 5.2, que descreve a metodologia proposta nesta dissertação. É importante lembrar que os testes foram executados para vários tamanhos de imagens e sub-imagens sendo que os exemplos apresentados foram escolhidos de forma aleatória.

Utilizando o código implementado no processador de propósito geral foi realizada no *soft-processor* Nios II /e a primeira execução possibilitando a análise inicial do *profiling* do código. O resultado do *profiling* é demonstrado pelo gráfico da figura 6.9 que apresenta a porcentagem do tempo de execução.

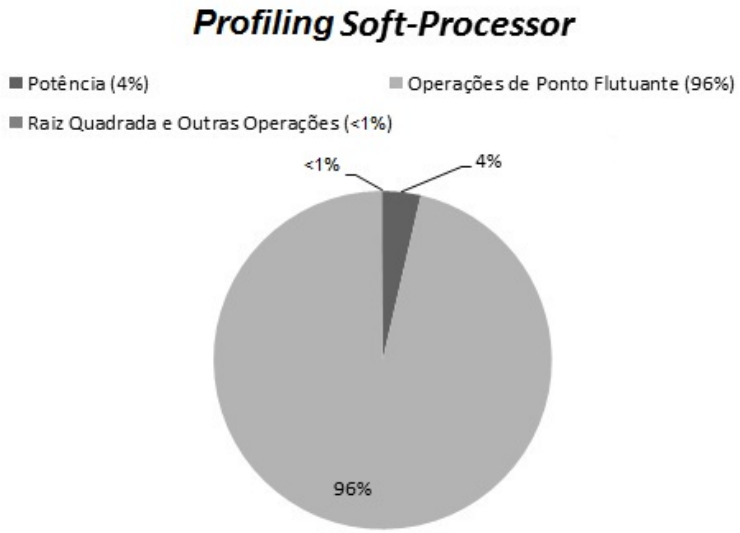

Figura 6.9: *Profiling* do código no *Soft-Processor*.

O procedimento anterior à utilização do método proposto apresentou como principal gargalo do algoritmo as operações de ponto flutuante. Confirmando a análise prévia, o *profiling* do código ao ser executado no *soft-processor* manteve as características notadas previamente e também apresentou um tempo

de execução aceitável para as outras funções, dentre elas, potenciação e raiz quadrada. Inicialmente foram feitas otimizações em software e depois em hardware como apresentado a seguir.

O software executado inicialmente no soft-processor apresentava as mesmas características do software desenvolvido inicialmente para o processador de propósito geral. Para analisar os efeitos das otimizações de software no *soft-processor* foram aplicadas as mesmas otimizações de software propostas anteriormente:

- Variáveis sem sinal.
- Utilização de expressões equivalentes a funções, como por exemplo, $x * x$ ao invés de $pow(x, 2)$.
- Utilização de vetores ao invés de matrizes.
- *Loop Unrolling*. Executar mais de uma operação do loop por iteração.

A principal otimização em software para o *soft-processor* foi a utilização de variáveis *unsigned char* (figura 6.10), o que reduziu consideravelmente o tamanho do programa gerado em memória. No caso das expressões equivalentes o tempo de execução simplesmente foi transferido das funções para outras operações sem apresentar melhorias no tempo total de execução, mesmo comportamento apresentado pelos operadores comuns. A utilização de vetores ao invés de matrizes considera a alocação da memória pelo sistema operacional, otimização que neste caso não influenciou o tempo de execução.

O gráfico apresentado na figura 6.10 mostra que a utilização das variáveis *unsigned char*, que neste caso é possível devido à escolha de imagens em escala de cinza que necessitam de apenas 8-bit por *pixel*, consumiu uma quantidade menor da memória do programa. A economia de memória é interessante pois a implementação do hardware desenvolvido futuramente em aplicações robóticas considera o armazenamento de imagens em um banco de dados disponível junto ao dispositivo embarcado.

A implementação da técnica do *loop unrolling* não apresentou mudança no tempo de execução. Devido a não existência de uma hierarquia de memória (que neste caso acelera o processo considerando a localidade espacial de dados) e também da ausência de hardware para a execução do *unrolling* de forma mais rápida, a técnica não acelerou o tempo de execução. A diferença dos resultados ao compararmos a implementação no processador de propósito geral é consequência, em grande parte, dos fatos citados. Desta forma as experiências com otimizações de software foram consideradas suficientes e as otimizações em hardware foram iniciadas, como sugere o método proposto.

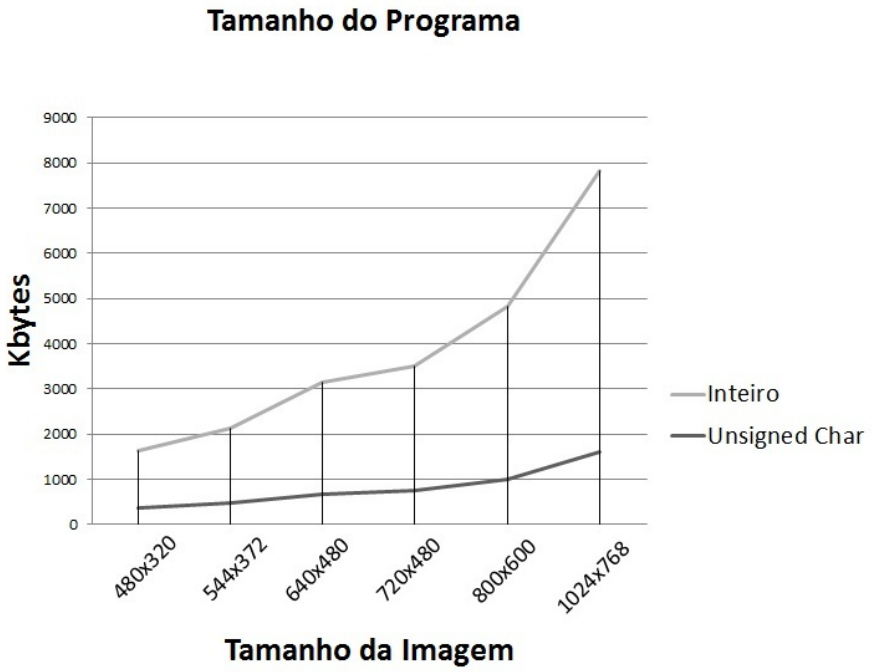

Figura 6.10: Análise do tamanho do programa em Kbytes.

6.3.4 Otimizações em Hardware

A primeira comparação entre tempos de execução foi feita entre as plataformas escolhidas. Após a análise do *profiling* inicial do código, a necessidade da utilização de hardware para operações de ponto flutuante foi confirmada. Portanto no caso do *soft-processor* Nios II a utilização da unidade de ponto flutuante como instrução customizada é considerada a primeira otimização de hardware iniciando assim a fase 3 do fluxograma da figura 6.1. A otimização de hardware também aparece indicada na figura da metodologia proposta, conforme apresentado na figura 5.2.

A unidade de ponto flutuante, no caso do *soft-processor* Nios II, sendo incluída como instrução customizada, acelera a comunicação com o processador diminuindo o tempo de execução já que o hardware é adicionado à unidade aritmética-lógica do processador. Existem várias unidades de hardware para operações em ponto flutuante disponíveis para download e utilização, porém algumas delas não são compatíveis com o *soft-processor* utilizado (largura de barramentos e padrão de interpretação de bits) ou apresentam um caminho crítico demasiado longo prejudicando consideravelmente o *clock* final e consequentemente o tempo de execução do sistema.

Alguns testes foram feitos com unidades disponíveis em um conhecido repositório de hardware chamado *Opencores*[14]. Duas unidades disponíveis fo-

[14]http://opencores.org/
[15]http://opencores.org/project,fpu100

ram escolhidas: a primeira[15] apesar de compatível com o *soft-processor*, após a síntese apresentou um caminho crítico de 3MHz tornando sua utilização inviável; a segunda unidade[16] possuía barramento de 64bits (*double-precision*) e, após conversão para 32bits ocasionando perda de precisão, seu tempo de execução apresentou-se também acima do aceitável sendo descartada da mesma forma. Após os testes, a opção foi pela utilização da unidade que é disponibilizada pela fabricante dos FPGAs totalmente compatível com o *soft-processor* utilizado.

A unidade de ponto flutuante com dupla precisão seguia o padrão IEEE 754 assim como o *soft-processor* Nios II. Ao analisar o padrão, é possível converter de 32 bits para 64 bits apenas considerando as partes mais significativas dos bits ocasionando uma perda de precisão, tanto na mantissa como no expoente. As entradas de 32 bits foram estendidas para 64 bits pelo primeiro conversor e a saída do cálculo da unidade de 64 bit foi reduzida pelo segundo conversor. As figuras 6.11 e 6.12 apresentam o hardware final que foi testado para a conversão. Após feita a conversão, o caminho crítico da unidade ao ser compilada caiu para um *clock* de menos de 50MHz, onde isto ainda assim irá prejudicar o *clock* geral do sistema.

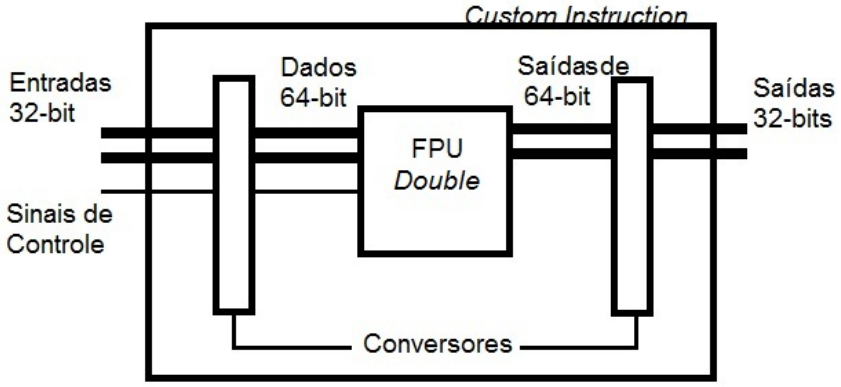

Figura 6.11: Adaptação da Unidade de Ponto Flutuante.

Utilizando a unidade de ponto flutuante do fabricante dos FPGAs presentes em 10 das 16 configurações prévias do *soft-processor*, o mesmo código foi executado em ambas as plataformas com imagens de tamanho reduzido, consequência da pequena quantidade de memória disponível na plataforma BeMicro. A maioria das sub-imagens foram utilizadas em tamanhos quadrados por conveniência para os testes. Utilizando os dados de cálculo de uma imagem de 160x120 pixels e o cálculo de uma janela de apenas 10x10 pixels apresentados na tabela 6.3 é possível analisar os resultados obtidos com os primeiros processadores configurados.

[16]http://opencores.org/project,fpu_double

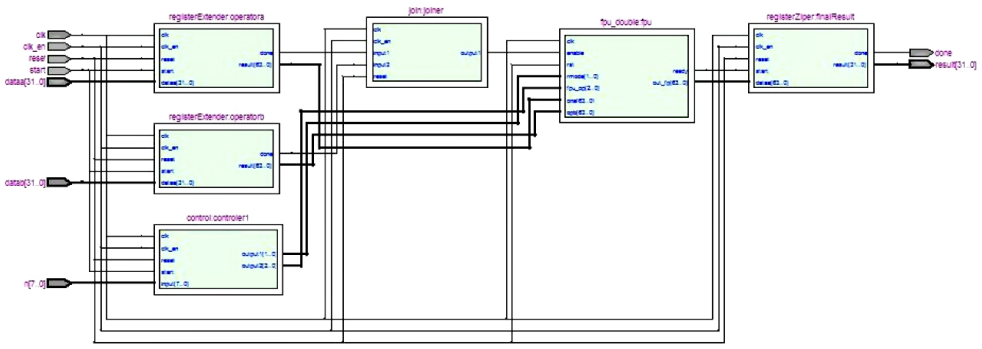

Figura 6.12: Diagrama de Blocos Quartus II IDE.

Tabela 6.3: Comparação entre processadores configurados.

	Execução BeMicro (s)	Execução DE2-70 (s)
Nios II /e	5607,7	1560,16
Nios II /s	1263,58	474,58
Nios II /f	980	371,97
Nios II /ef	4057	1299,37
Nios II /sf	980	348,88
Nios II /ff	-	276,19
Nios II /efd	3888	1219
Nios II /sfd	856,7	375,69
Nios II /ffd	-	293,1

As entradas da tabela 6.3 que não possuem valores correspondem as configurações do *soft-processor* que extrapolaram a capacidade do FPGA Cyclone III. As colunas da tabela possuem uma diferença considerável entre o tempo de execução para as plataformas de desenvolvimento escolhidas. Esta diferença tão grande é explicada por um conjunto de fatores dentre eles: a utilização da memória SDRAM no caso da plataforma BeMicro; a necessidade de um hardware PLL para gerar o clock de 50 MHz; devido ao fato do projeto ocupar maior parte do FPGA o posicionamento do hardware e a distribuição dos pinos faz diferença e também a forma como foram construídos o barramento nas plataformas. Os resultados demonstram que a plataforma BeMicro apesar das vantagens de tensão de alimentação e tamanho não apresenta um desempenho aceitável para ser utilizada como plataforma de execução, sendo descartada neste estágio do desenvolvimento.

Após a análise inicial, foi decidido utilizar apenas a plataforma DE2-70 e o FPGA Cyclone II. Os primeiros processadores configurados para esta plataforma possuíam apenas a memória *On-Chip*, ou seja, a memória de acesso rápido interna ao processador equivalente à memória *cache*. Esta memória, levando em conta os componentes utilizados no processador, possuíam entre 100 e 120 Kbytes. Visando a implementação do sistema de uma forma flexível

com relação ao tamanho das imagens e sub-imagens utilizadas, as memórias SSRAM e SDRAM foram incluídas no projeto assim como os dispositivos de entrada e saída como LEDs e chaves (*Switches*).

A inclusão desta memória nesta plataforma não é uma tarefa trivial. O hardware da memória SDRAM necessita de um atraso de 3ns no *clock* geral do sistema de forma que o hardware para esta ligação específica precisa ser criado. O suporte para esta operação dentro das IDEs `Quartus` e `Sopc Builder` só apareceu como padrão em suas versões mais novas (10.2 e 11) o que dificultou sua implementação no caso da versão utilizada (9.1). O atraso de 3ns é atingido pela inclusão de um hardware PLL internamente ao processador. Desta forma a nova arquitetura do processador configurado é apresentada na figura 6.13.

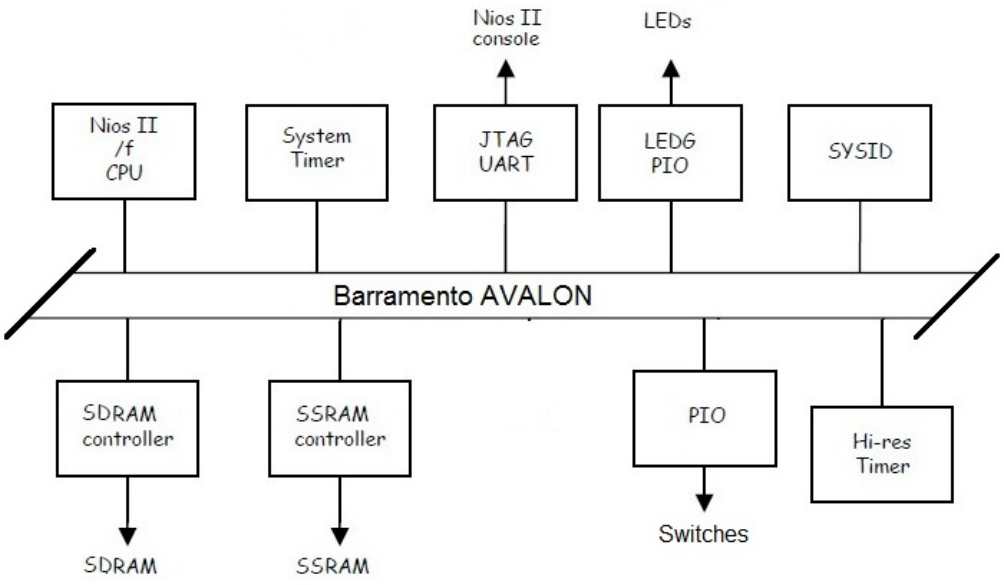

Figura 6.13: Arquitetura dos processadores para a plataforma DE2-70.

Comparando os resultados do tempo de execução entre os *soft-processors* que não incluíam as memórias SDRAM e SSRAM e a nova arquitetura é possível realizar a análise do impacto (*overhead*) da utilização da memória já que a diferença entre as configurações, além das memórias, é minima e está sendo utilizado o mesmo FPGA na mesma plataforma. Portanto outros 9 soft-processors foram configurados incluindo as memórias, onde a tabela 6.4 apresenta os resultados para imagens de 160x120 pixels e sub-imagem 10x10. A diferença de tempos de acesso à memória presentes nesta tabela é ocasionada principalmente pela necessidade de um controlador de memória para utilização de memória SDRAM e da transferência dos dados, juntamente com a forma que o próprio *soft-processor* acessa estes dados na memória que não se mostrou suficientemente eficiente.

Tabela 6.4: Comparação entre utilização de memórias.

	DE2-70 com SSRAM e SDRAM (s)	DE2-70 com On-Chip (s)
Nios II /e	4583.66	1560,16
Nios II /s	782.05	474,58
Nios II /f	454.42	371,97
Nios II /ef	3964.96	1299,37
Nios II /sf	609.58	348,88
Nios II /ff	437.25	276,19
Nios II /efd	3718.01	1219
Nios II /sfd	641.12	375,69
Nios II /ffd	343.54	293,1

A diferença apresentada pelos valores da tabela pode ser considerada principalmente como consequência da utilização das memórias. Embora haja um aumento no tempo de execução, a quantidade de memória disponível foi modifica de 120 KB para 70 MB. Esta quantidade de memória possibilita o armazenamento de um grupo de imagens ou até mesmo a utilização de imagens de tamanhos maiores o que justifica diretamente sua utilização. Nesta fase do desenvolvimento, após analisar em várias situações o desempenho de todos os *soft-processors* configurados e ponderar variáveis como: (i) consumo de energia; (ii) área ocupada e (iii) tempo de execução, foram descartados: os testes posteriores com a plataforma BeMicro, e também a utilização dos *soft-processors* /e e /s (mesmo com FPU e hardware para divisão) juntamente com a versão /f e /f com a unidade de ponto flutuante sem o divisor de hardware.

O método proposto para o desenvolvimento do hardware, considerando-se sua adoção em aplicações de robótica, deverá realizar o *matching* de imagens inteiras. Este tipo de operação possui um tempo de execução menor em comparação com o mesmo algoritmo executando sub-imagens pequenas. Para a configuração proposta do *soft-processor* utilizando as memórias SDRAM e SSRAM os tempos de execução para o cálculo do coeficiente entre imagens é apresentado no gráfico da figura 6.14.

Os resultados dos tempos de execução demonstram um tempo ainda alto de execução tanto para comparação de imagens quanto para sub-imagens pequenas. O próximo passo do desenvolvimento de hardware envolve o aumento do *clock* de execução do processador dentro do caminho crítico alcançado pelos algoritmos de *fitting* e *place and route* para que o poder de processamento seja elevado ao máximo. O aumento do *clock* irá impactar diretamente na área do processador final que irá incluir uma outra PLL para geração do clock, e também no consumo de energia já que com o aumento da frequência do clock do processador naturalmente aumenta seu consumo.

O *soft-processor* escolhido para receber a PLL e ter seu *clock* elevado para 100MHz é o processador Nios II /f com a unidade de ponto flutuante incluindo

Figura 6.14: Cálculo do Coeficiente entre Imagens de mesmo tamanho.

a divisão em hardware e que utiliza as memórias SSRAM e SDRAM (denotado como Nios II /ffd ram). Este processador apresentou após o processo de síntese uma frequência máxima de 131,06 MHz segundo os dados do caminho crítico. Considerando a utilização das memórias e o hardware a ser desenvolvido futuramente o clock escolhido foi de 100MHz.

Como pode ser observado no gráfico da figura 6.15, que mostra os dados de cálculo entre imagem e sub-imagem de mesmo tamanho, a diferença no tempo de execução entre o clock de 50MHz e o de 100MHz é muito significativa e a medida em que as imagens aumentam de tamanho a diferença aumenta sendo que é realmente necessário que o processador trabalhe em uma frequência mais alta considerando que o objetivo é diminuir o tempo de execução. O aumento na área final é aceitável porém o consumo de energia caso fosse possível ser medido iria aumentar.

A segunda parte do desenvolvimento de hardware escolhida para este problema foi acelerar as operações de soma que existem no cálculo do coeficiente executando-as de forma paralela. Considerando que a aritmética de ponto flutuante já foi tratada com a inclusão da unidade de ponto flutuante e que as funções que executam as operações de exponenciação e raiz quadrada não ocuparam um porcentagem alta do tempo de execução a aceleração das operações de soma foi escolhida.

Considerando a abordagem proposta de utilização de instruções customizadas para inclusão de hardware devido ao baixo custo de comunicação com o *soft-processor*, as instruções possuem como entrada dois operandos de 32 bits cada e uma saída de 32 bits. Desta forma, considerando que estão sendo utilizados valores de 8 bits para representar os pixels da imagem, poderão ser

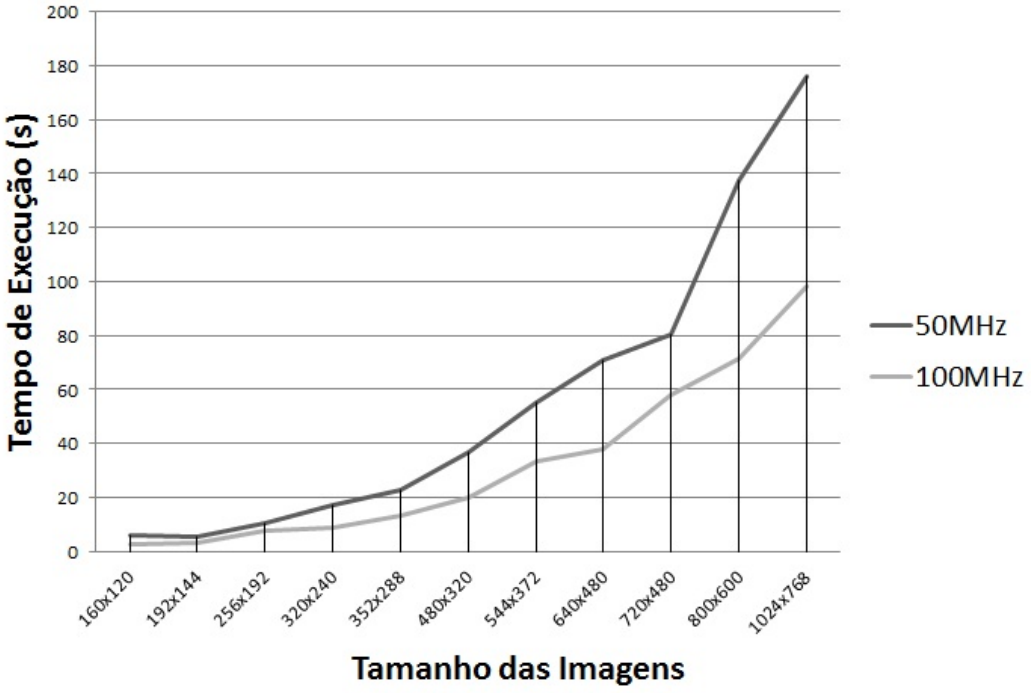

Figura 6.15: Comparação entre as frequencias de *clock*.

somados quatro valores das imagens de uma só vez. Tendo 4 valores de 8 bits somados com outros 4 valores de 8 bits os 32 bits da saída serão suficientes para acomodar os bits da soma e serão considerados um número inteiro. Esta limitação se deve ao fato da abordagem escolhida neste trabalho, sendo que se uma estrutura de hardware fosse incluída de outra forma no soft-processor mais somas poderiam ser feitas simultaneamente porém o tempo de comunicação seria mais alto. Uma característica importante da estrutura desenvolvida é que é possível reutilizá-la em qualquer parte do algoritmo que fizer a soma dos pixels da imagem.

As figuras a seguir (6.16 a 6.18) apresentam uma ilustração de como é o hardware desenvolvido, uma parte do diagrama *Register Transfer Level* (RTL) correspondente gerado pela síntese da IDE `Quartus II` e as formas de onda de uma execução de soma respectivamente. Trata-se de uma estrutura que agrega somadores em cascata de forma a somar simultaneamente 4 pares de valores de 8 bits resultando em uma saída de 11 bits extendida para 32 bits devido ao barramento do *soft-processor*.

A inclusão deste hardware como instrução customizada do processador não modificou o caminho crítico do processador de forma a diminuir seu clock máximo significativamente e também não aumentou consideravelmente os dados de síntese do processador. O software necessitou de algumas mudanças básicas nas funções onde a instrução customizada será utilizada e algumas macros foram inseridas para sua utilização. Os detalhes básicos do desenvolvimento das instruções e as formas de utilização em software podem ser

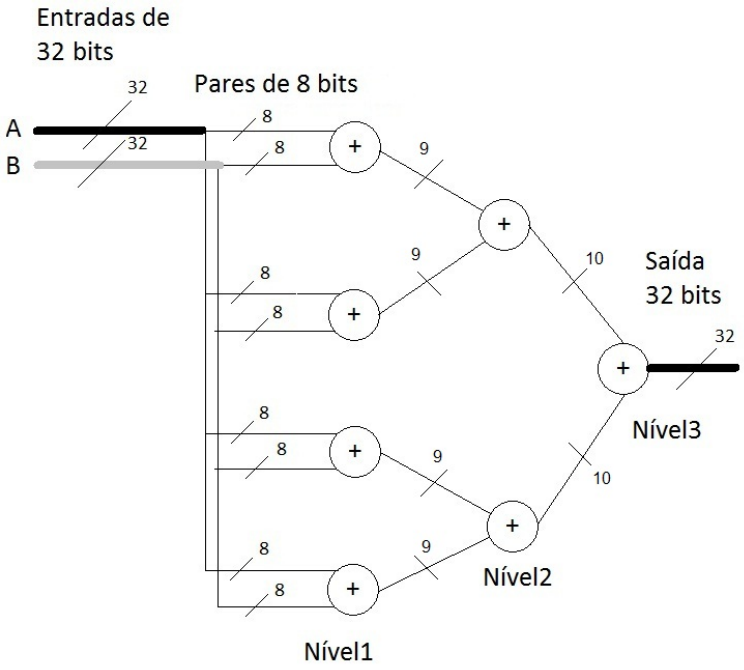

Figura 6.16: Hardware Desenvolvido.

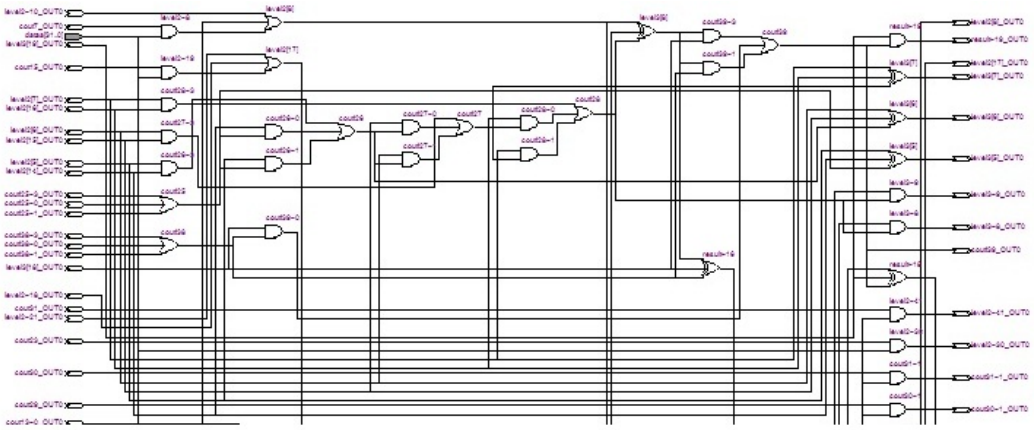

Figura 6.17: Parte do digarama RTL correspondente ao nível 2.

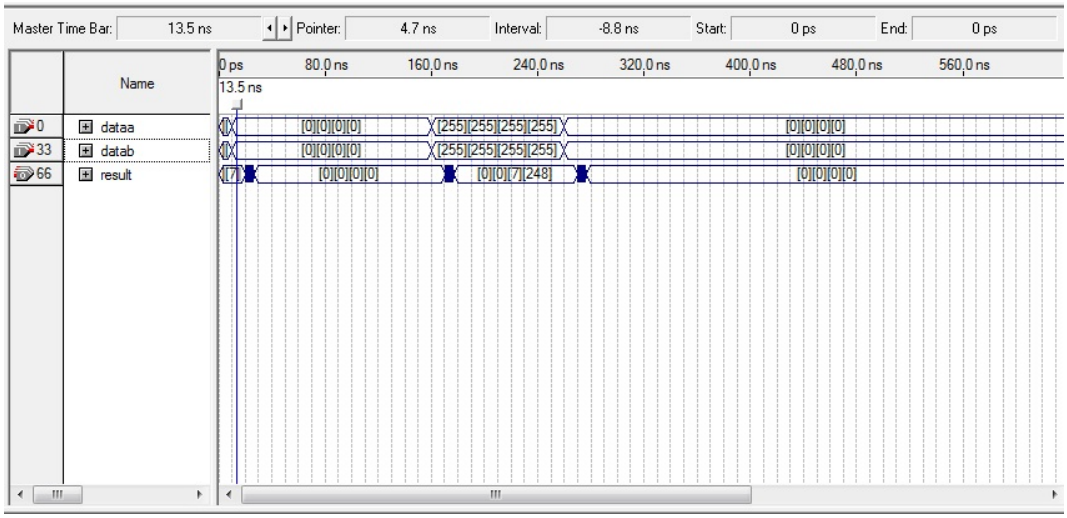

Figura 6.18: Formas de onda da instrução customizada.

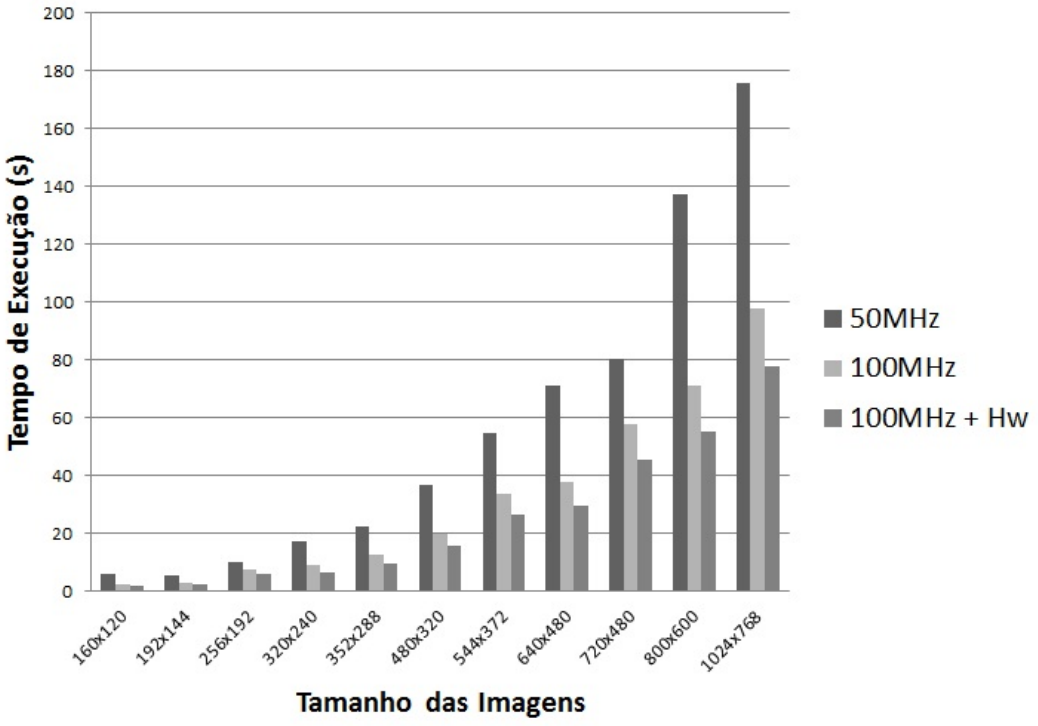

Figura 6.19: Instrução Customizada.

encontradas no manual[17] do *soft-processor*.

A instrução customizada desenvolvida pode ser entendida como uma forma de fazer um *loop unrolling* em hardware. A idéia de sua utilização foi consequência também do impacto desta técnica no processador de propósito geral. Considerando que sua utilização envolveu mudanças no código e também existe o tempo de comunicação entre a instrução e o processador o fato de se executar 4 operações de soma por instrução reduziu o tempo de execução como demonstra o gráfico da figura 6.19 que considera o cálculo do coeficiente para imagem e sub-imagem de mesmo tamanho. A instrução foi incluída no *soft-processor* Nios II /f com a unidade de ponto flutuante completa com clock de 100MHz (denominado de Nios II /ffd ram pll ci).

Após as otimizações em software e o desenvolvimento de hardware seguindo o método proposto, os tempos de execução juntamente com as características do software e do hardware foram analisados atingindo portanto a etapa final da fase 3 do diagrama de experimentação proposto no início deste capítulo presente na figura 6.1. Os resultados finais e a análise do *speedup* são descritos a seguir.

[17] http://www.altera.com/literature/ug/ug_nios2_custom_instruction.pdf

Tabela 6.5: Comparação entre processadores configurados.

Soft-Processor	EL	Pinos	Memória	PLL	Clock
Nios II /e	2052	7	0,94	0	117,0
Nios II /s	2736	7	0,82	0	95,36
Nios II /f	3420	7	0,84	0	110,0
Nios II /ef	2736	7	0,94	0	118,99
Nios II /sf	4104	7	0,82	0	88,75
Nios II /ff	4789	7	0,84	0	116,43
Nios II /efd	7525	7	0,79	0	111,22
Nios II /sfd	8894	7	0,82	0	90,88
Nios II /ffd	9578	7	0,84	0	116,85
Nios II /eram	3420	350	0,1	1	81,25
Nios II /sram	4789	350	0,11	1	105,93
Nios II /fram	5473	350	0,13	1	138,01
Nios II /efram	4789	350	0,1	1	83,05
Nios II /sfram	6157	350	0,11	1	65,07
Nios II /ffram	6841	350	0,13	1	132,77
Nios II /efdram	9578	350	0,1	1	80,83
Nios II /sfdram	10946	350	0,11	1	71,61
Nios II /ffdram	11630	350	0,13	1	131,06
Nios II /ffdpll	9578	7	0,84	1	115,41
Nios II /ffdrampll	12314	99	0,12	1	119,22
Nios II /ffdrampllci	12314	99	0,12	1	108,72

6.4 Análise dos Resultados e Speedup

Durante o desenvolvimento do trabalho foram configuradas 21 versões diferentes do *soft-processor* onde várias características foram modificadas e cada geração de sistema e processo de síntese consumiam aproximadamente 10 minutos. Apesar do alto tempo de geração e síntese do sistema o tempo de desenvolvimento sem as ferramentas seria consideravelmente mais alto. A tabela 6.5 contém os dados de síntese para cada processador configurado na plataforma DE2-70 para o FPGA Cyclone II.

A tabela 6.5 apresenta os dados de síntese de elementos lógicos (EL), pinos (Pinos), bits de memória (Memória), utilização de PLLs (PLL) e a frequência máxima de *clock* alcançada (Clock Max). A diferença nos pinos ocorre por utilização de chaves e *leds* da plataforma, os bits de memória estão maiores em configurações onde não há memória SDRAM e SSRAM devido ao fato de que toda a memória *on-chip* disponível foi utilizada e todos os *designs* que utilizaram PLL fizeram uso de 1 das 4 disponíveis. A última configuração que contém tudo que foi desenvolvido (Nios II /f + FPU + PLL + Instrução Customizada) não apresenta dados de síntese altos de forma que não justifiquem sua utilização. O processador Nios II /s obteve as piores frequências máximas para todos os casos de configurações similares.

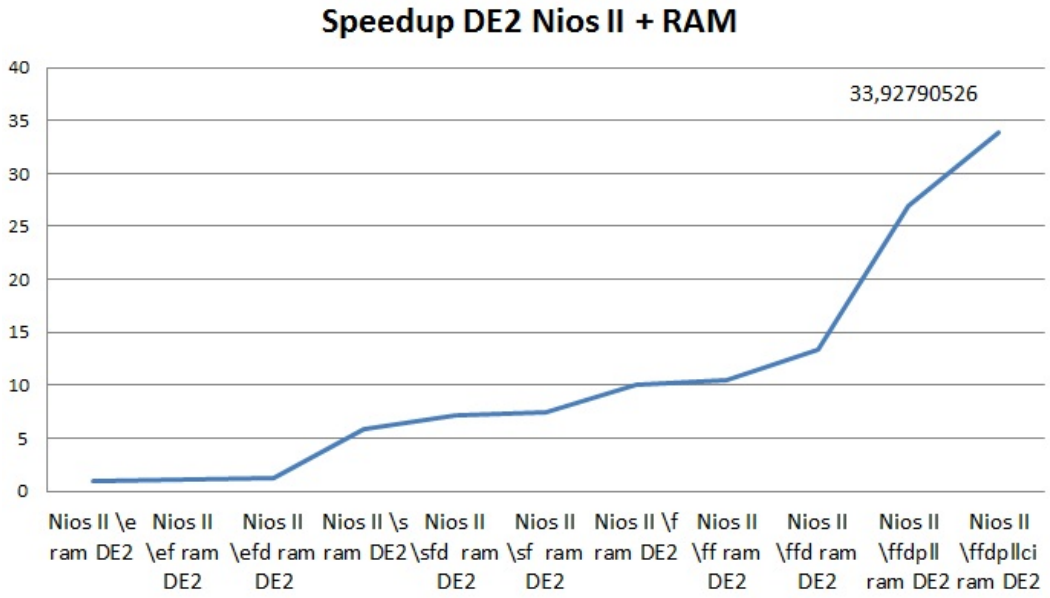

Figura 6.20: *Speedup* com utilização de memória.

A análise do *speedup* pode ser feita de várias maneiras levando em consideração todas as etapas do desenvolvimento. A primeira análise (figura 6.22) consiste em comparar o primeiro tempo de execução obtido da plataforma BeMicro, utilizando a versão /e do *soft-processor* Nios II desprovida de hardware para aritmética de ponto flutuante, com o último tempo de execução obtido que é proveniente da execução na versão /f do *soft-processor* com unidade de ponto flutuante completa, *clock* de 100MHz, instrução customizada para hardware e memórias SDRAM e SSRAM na plataforma DE2-70. Este resultado demonstra desde a influência da plataforma utilizada e da comunicação do FPGA, passando pela utilização ou não de recursos como *pipelining* e hardwares para acelerar operações aritméticas, até a influência da mudança no *clock* e inclusão de um hardware específico para o problema em questão. O problema desta avaliação é que as versões iniciais dos *soft-processors* não permitiam a utilização de imagens maiores sendo que a análise foi feita com imagens de 160x120 *pixels* e janelas de 10x10 *pixels*.

Considerando as configurações que permitiram a análise de imagens do tamanho desejado, ou seja, as que possuíam as memórias SDRAM e SSRAM as análises podem ser feitas desde a versão /e sem FPU até a versão final do *soft-processor* novamente (figura 6.20). O real *speedup* atingido, de 33.92, é demonstrado pelo gráfico da figura (figura 6.20) pois apresenta a otimização do hardware para a plataforma que apresenta o melhor resultado desconsiderando a plataforma BeMicro utilizada em uma fase inicial do desenvolvimento. Para descartar da análise os efeitos de mudanças básicas de hardware como *pipelining* por exemplo, é necessário analisar apenas os valores do tempo de execução entre os *soft-processors* Nios II /f (figura 6.21).

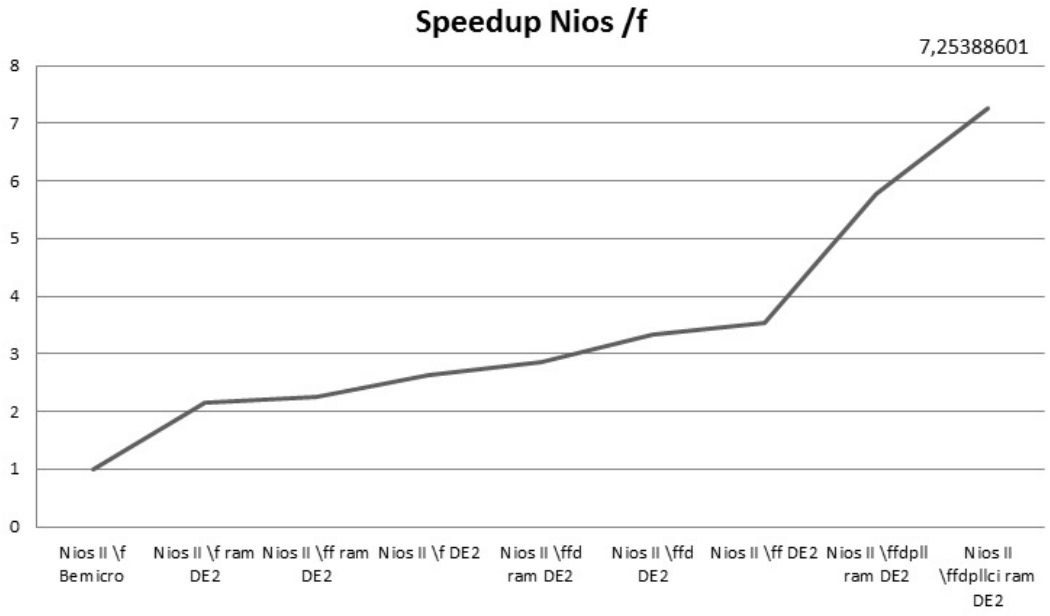

Figura 6.21: *Speedup* para Nios II /f.

Os resultados apresentados permitiram analisar o comportamento do sistema proposto criado baseado em um método de co-projeto de hardware/software baseado em *profiling* para executar o algoritmo de correlação cruzada normalizada (NCC). As conclusões obtidas a partir da análise dos resultados são apresentadas no capítulo seguinte.

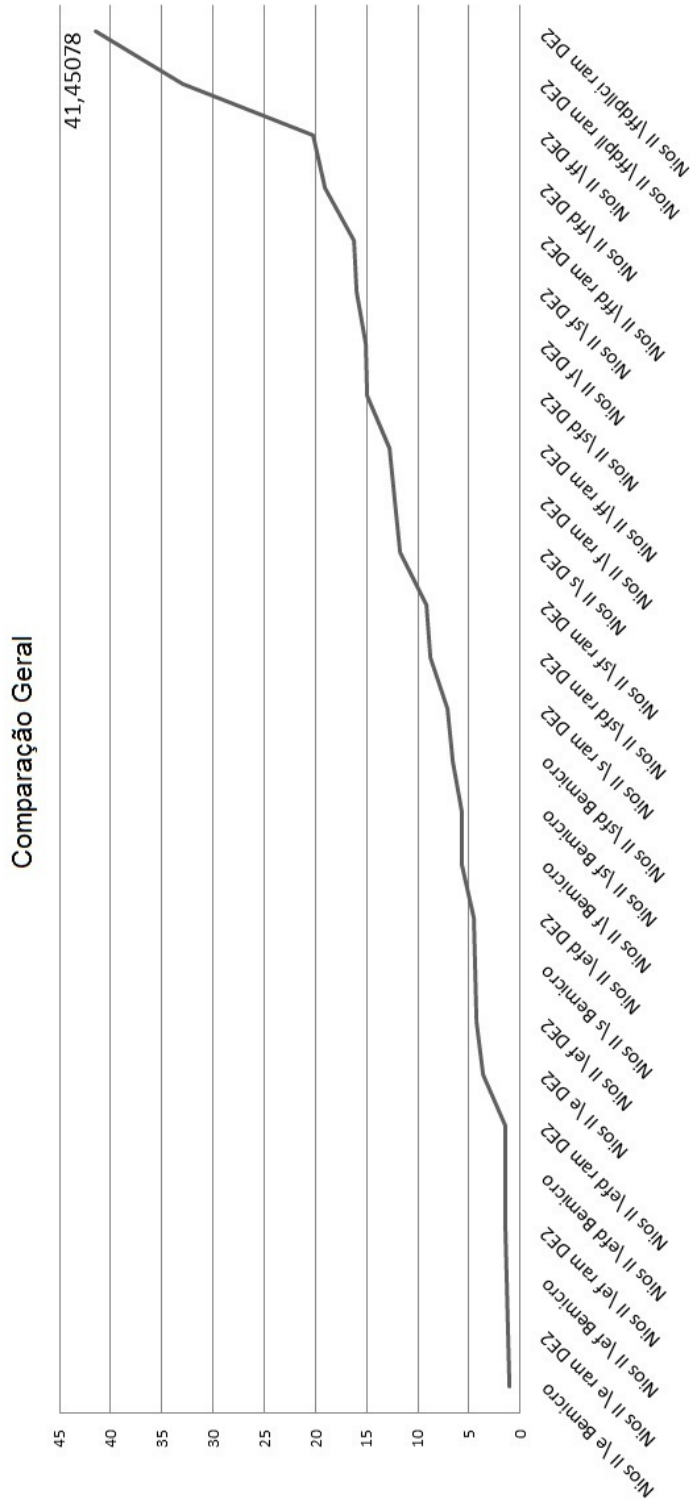

Figura 6.22: Comparação geral entre abordagens.

CAPÍTULO

7

Conclusão

Esta dissertação apresentou um co-projeto de hardware/software baseado em *profiling* para correlação cruzada normalizada de imagens. Os resultados apresentados na seção anterior demonstram que o objetivo principal do trabalho, atingir um *speedup* significativo no tempo de execução do algoritmo através do desenvolvimento do método proposto, foi atingido. Durante o desenvolvimento do trabalho foram configurados 21 *soft-processors*, ou seja, 21 versões diferentes do Nios II onde o tempo de execução do algoritmo foi avaliado para vários tamanhos de imagens e de sub-imagens. O desenvolvimento do trabalho também apresentou o efeito da utilização de técnicas de otimização de software para sistemas embarcados tanto em computadores de propósito geral quanto no *soft-processor*. A configuração de um alto número de processadores permitiu a análise de vários fatores com relação ao desenvolvimento de hardware e sua utilização.

O método utilizado durante o desenvolvimento, apresentado como uma modificação dos métodos de co-projeto de hardware/software baseados em *profiling* por possuir dois ciclos de desenvolvimento onde o desenvolvimento de software precede o desenvolvimento de hardware, apresentou resultados bastante satisfatórios. A melhora no desempenho da execução do algoritmo foi atingida ainda que o sistema como um todo não tenha apresentado um tempo de execução aceitável para aplicações robustas de robótica.

Durante o desenvolvimento deste trabalho o método também foi aplicado em outros problemas (DIAS et al. (2010b)DIAS et al. (2010a)) apresentando resultados de melhoria no tempo de execução igualmente satisfatórios. O sucesso do método apresentado também é consequência da utilização de dispositivos reconfiguráveis durante seu desenvolvimento que permitiram a prototi-

pação rápida e análise do desempenho. Apesar do tempo de execução atingido não ser adequado para aplicações robóticas de tempo real, ainda é possível utilizar FPGAs de alto desempenho adotando a metodologia proposta e obter um *speedup* ainda maior.

Os resultados deste trabalho demonstram que a plataforma BeMicro apesar de possuir as vantagens de alimentação e tamanho possui um FPGA pouco robusto e, no caso do problema tratado nesta dissertação, não apresentou desempenho aceitável. Estes resultados sugerem ainda que esta plataforma seja utilizada preferencialmente para desenvolvimento de aplicações simples e/ou didáticas. A segunda plataforma de desenvolvimento, DE2-70, apresentou melhores resultados e uma possibilidade maior para o desenvolvimento de hardware customizado para ser adicionado ao *soft-processor*. Apesar dos dados de síntese apresentados no capítulo anterior e do alto número de interfaces de entrada e saída de dados disponíveis na plataforma, o FGPA presente que não é de alto desempenho apresentou limitações, como por exemplo, a pequena quantidade de memória *on-chip* que pode ser alocada ocasionando a necessidade de utilização de memórias SDRAM e SSRAM para a execução do algoritmo mesmo com imagens de tamanho reduzido.

As ferramentas de desenvolvimento escolhidas para o trabalho (Quartus II IDE, Nios II EDS, Nios II soft-processor, SOPC Builder) possibilitaram a exploração e análise de vários perfis e características de hardware em tempo de desenvolvimento aceitável. É importante ressaltar que a documentação disponibilizada pelo fabricante, aliada ao alto número de fóruns de discussão e de desenvolvedores de hardware que utilizam as plataformas escolhidas, foi um fator determinante para o sucesso do desenvolvimento. Porém a grande quantidade de problemas que estas ferramentas apresentam e o número alto de versões disponibilizadas em curtos períodos de tempo causam contratempos durante o desenvolvimento. Existem também manuais que contém os erros mais comuns encontrados e formas de tentar corrigi-los, porém estes manuais não são facilmente encontrados na página do fabricante e nem sempre os erros são possíveis de serem corrigidos e neste caso o fabricante sugere que seja esperada a próxima versão do software.

Apesar das características apresentadas anteriormente o *soft-processor* Nios II não pode ser considerado um processador que apresenta resultados satisfatórios para este tipo de desenvolvimento de co-projeto de hardware/software principalmente considerando uma comparação com os processadores de propósito geral não embarcados. No caso de aplicações robóticas, geralmente faz-se uso de um computador embarcado que possui um processador de propósito geral. Ao desenvolver um co-projeto de propósito específico é esperada uma diminuição no consumo de energia, de área e também no tempo de execução

do algoritmo. O tempo de execução do algoritmo com a utilização do *soft-processor* para processamento com uma frequência de 100MHz dificilmente consegue competir diretamente com o tempo de execução de um processador de propósito geral (com um *clock* na ordem de 2.4 a 3.0 GHz) embora o consumo de energia e a área de *chip* ocupada diminuam consideravelmente. Este problema pode ser parcialmente contornado com o desenvolvimento das instruções customizadas em hardware para aceleração do algoritmo. No caso do problema escolhido o tempo de execução diminuiu, mas ainda não atingiu o nível desejado para inclusão em um sistema de navegação robótica.

O estado da arte no desenvolvimento de soluções para o problema da correlação cruzada pode ser dividido em três grupos: desenvolvimento de software, desenvolvimento de hardware dedicado e desenvolvimento de co-projetos de hardware/software. No caso do desenvolvimento de software existem várias aplicações onde os algoritmos NCC e FNCC estão sendo utilizados e otimizados como os trabalhos de Fernandes et al. (2010) e Hongcheng and Liu (2010). Exemplos de arquiteturas de hardware dedicadas desenvolvidas para o algoritmo podem ser vistas no trabalho de Yonghong (2010) e também de Tao et al. (2010). As arquiteturas desenvolvidas apresentam um tempo de execução aceitável, porém nenhuma foi incluída ainda em um sistema real robótico além de apresentarem alta complexidade de desenvolvimento. O trabalho apresentado encontra-se situado na parte de desenvolvimento de co-projetos utilizando métodos baseados em *profiling* e *soft-processors* sendo que em sua busca o autor não encontrou nenhuma referência que apresentasse o desenvolvimento baseado nas mesmas características.

O desenvolvimento do trabalho trouxe principalmente as seguintes contribuições:

- O desenvolvimento de um co-projeto de hardware/software que apresentou um *speedup* de 33,92 comparado com o desempenho do algoritmo de referência puramente sequêncial.

- Análise da implementação do algoritmo de correlação cruzada normalizada de imagens no *soft-processor* Nios II com relação as seguintes variáveis: configurações do *soft-processor* Nios II (*economic, standard e fast*), relação de tamanhos de imagem e sub-imagem com o tempo de execução do algoritmo no *soft-processor*, comparação de *profiling* dos algoritmos com relação à gargalos no tempo de execução, utilização de memória *on-chip* e RAM.

- Comparação de dois FPGAs da fabricante Altera (Cyclone II e Cyclone III) em duas plataformas diferentes (respectivamente Terasic DE2-70 e Arrow BeMicro) para a implementação do algoritmo.

- Influência de técnicas de otimização de software para sistemas embarcados no tempo de execução do software e também na usabilidade do código.

- Modificação na metodologia básica de co-projeto de hardware/software baseada em *profiling* para sistemas embarcados fazendo com que o particionamento seja feito de forma ainda mais natural durante o desenvolvimento introduzindo uma divisão por ciclos.

Este trabalho envolveu diversas área do conhecimento tais como: computação reconfigurável utilizando FPGAs, co-projeto de hardware/software, projeto de hardware com HDL, robótica, visão computacional e arquitetura de computadores. No apêndice A são apresentadas as publicações originadas deste trabalho de pesquisa e também as publicações anteriores relacionadas a este trabalho.

7.1 Trabalhos Futuros

A análise dos resultados obtidos e as considerações colocadas anteriormente sugerem como trabalhos futuros: a implementação do sistema apresentado em uma plataforma que possua um FPGA de alto desempenho como a família Stratix do mesmo fabricante dos FPGAs utilizados neste trabalho; o desenvolvimento de um hardware dedicado para solução do problema utilizando o *soft-processor* apenas como *glue logic* para facilitar a implementação ou também sem a utilização do *soft-processor*; o desenvolvimento da interface de comunicação entre o co-projeto desenvolvido e o sensor óptico para que o sistema possa ser avaliado em execução e assim o tempo total de execução considerando a aquisição da imagem e a execução possa ser avaliado; análise de outros algoritmos de *matching* de imagens para serem implementados como co-projetos de hardware/software e sua comparação; estudo e implementação de algoritmos de correlação cruzada cujo resultado seja aproximado (não exato), como o Fast NCC e métodos baseados no uso apenas de unidades de ponto fixo, onde para tanto teria que ser avaliado não somente o resultado em termos de *speedup*, mas conjuntamente a qualidade do *match* (questão não abordada neste trabalho, uma vez que a qualidade do *match* não estava em questão); explorar ainda mais o "*loop unrolling*"e blocos maiores em *custom instruction* (mais somadores e mais operações); utilizar técnicas de correlação de imagens piramidal a exemplo da técnica utilizada em outro trabalho desenvolvido junto ao LRM/USP (Facchinetti and Osório (2010)); explorar o consumo de energia dos vários soft-processors configurados. Todos os trabalhos futuros apresentados visam obter uma melhoria no tempo de execução

do algoritmo e uma análise de viabilidade de utilização do produto final em um sistema de navegação robótica.

Referências Bibliográficas

Aratò, P., S.Juhàsz, Mann, . Z., Órban, A., and Paap, D. (2003). Hardware-software partitioning in embedded system design. *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*.

Assumpcao, J., Wolf, D. F., and Marques, E. (2007). Towards a hardware accelerated obstacle avoidance system for mobile robots using monocular vision. In *International Symposium on Industrial Embedded Systems*, pages 365–368.

Atasu, K., Ozturan, C., Dundar, G., Mencer, O., and Luk, W. (2008). Chips: Custom hardware instruction processor synthesis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(3):528–541.

Berger, S. A. (2001). *Embedded Systems Design: An Introduction to Processes, Tools and Techniques*. CMP Books, Berkley, CA, USA.

Bobda, C. (2007). *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*. Springer Publishing Company, Incorporated.

Bonato, V., Marques, E., and Constantinides, G. A. (2008). A parallel hardware architecture for image feature detection. In *ARC '08: Proceedings of the 4th international workshop on Reconfigurable Computing*, pages 137–148, Berlin, Heidelberg. Springer-Verlag.

Briechle, K. and Hanebeck, U. D. (2001). Template Matching Using Fast Normalized Cross Correlation. In *Proceedings of SPIE: Optical Pattern Recognition XII*, volume 4387, pages 95–102.

Camposano, R. and Wilberg, J. (1996). Embedded system design. *Des. Autom. Embedded Syst.*, 1(1-2):5–50.

Chati, H., Muhlbauer, F., Braun, T., Bobda, C., and Berns, K. (2007). Hardware/software co-design of a key point detector on fpga. In *Field-*

Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on, pages 355–356.

Chen, Y.-K., LEE, G. G. C., Mattavelli, M., and Jang, E. S. (2009). Special issue: Algorithm/architecture co-exploration of visual computing on emerging platforms. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1573–1575.

Chiodo, M., Engels, D., Giusto, P., Hsieh, H., Jurecska, A., Lavagno, L., Suzuki, K., and Sangiovanni-Vincentelli, A. (1996). A case study in computer-aided co-design of embedded controllers. *Des. Autom. Embedded Syst.*, 1(1-2):51–67.

Compton, K., Hauck, S., and Compton, K. (2000). An introduction to reconfigurable computing. *IEEE Computer*.

de Micheli, G. and Gupta, R. K. (1997). Hardware/software co-design. In *Proceedings of IEEE, vol. 85*, pages 349–365.

DeSouza, G. N. and Kak, A. C. (2002). Vision for mobile robot navigation: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:237–267.

Dias, M. and Lacerda, W. (2009). Hardware/software co-design using artificial neural network and evolutionaryy computing. In *Programmable Logic, 2009. SPL. 5th Southern Conference on*, pages 153–157.

DIAS, M. A., Sales, D. O., and OSORIO, F. S. (2010a). Hw/sw co-design architecture for evolutionary robotics. In *Proceedings of The 7th Latin American Robotics Simposium (LARS)*, pages 43 – 48.

DIAS, M. A., Sales, D. O., and OSORIO, F. S. (2010b). A profile-based method for hardware/software co-design applied in evolutionary robotics using reconfigurable computing. In *IEEE Electronics Robotics and Automotive Mechanics Conference (CERMA)*, pages 463–468.

Dudek, G. and Jenkin, M. (2000). *Computational principles of mobile robotics*. Cambridge University Press, New York, NY, USA.

Eles, P., Peng, Z., Kuchcinski, K., and Doboli, A. (1997). System level hardware/software partitioning based on simulated annealing and tabu search. In *Design and Automation for Embedded Systems*, volume 2, pages 5–32.

Ernst, R., Henkel, J., and Benner, T. (2002). *Hardware-software cosynthesis for microcontrollers*. Kluwer Academic Publishers, Norwell, MA, USA.

Estrin, G. (2002). Reconfigurable computer origins: the ucla fixed-plus-variable (f+v) structure computer. *Annals of the History of Computing, IEEE*, 24(4):3–9.

Facchinetti, L. and Osório, F. S. (2010). Navegação visual de robôs móveis autônomos baseada em métodos de correlação de imagens. In *Workshop on Computational Intelligence 2010 - Joint Conference SBIA-SBRN-LARS*, pages 518–523.

Fernandes, C. W., Bellar, M. D., and Werneck, M. M. (2010). Cross-correlation-based optical flowmeter. *IEEE T. Instrumentation and Measurement*, 59(4):840–846.

Gajski, D. D., Zhu, J., and Dömer, R. (1997). Essential issues in codesign. Technical report, Hardware/Software Co-Design: Principles and Practice.

Ghosh, S. K. (1999). *Hardware Description Languages: Concepts and Principles*. Wiley-IEEE Press.

Gonzalez, R. C. and Woods, R. E. (2006). *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Gupta, R. and Micheli, G. D. (1993). Hardware-software co-synthesis for digital systems. In *IEEE Design & Test of Computers*, pages 29–41.

Gupta, R. K. (1995). *Co-Synthesis of Hardware and Software for Digital Embedded Systems*. Kluwer Academic Publishers, Norwell, MA, USA. Foreword By-Micheli, Giovanni De.

Hauck, S. and Dehon, A. (2007). *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon)*. Morgan Kaufmann, 30 Cororate Drive, Suite 400, Burlington, MA.

Henkel, J. (2003). Closing the soc design gap. *Computer*, 36(9):119–121.

Hidalgo, J. I. and Lanchares, J. (1997). Functional paritioning for hardware-software codesign using genetic algorithms. In *Proceedings of the 23rd EUROMICRO conference*, pages 631–638.

Hongcheng, Y. and Liu, W. (2010). Design of time difference of arrival estimation system based on fast cross correlation. In *Proceedings of 2nd International Conference on Future Computer and Communication*, volume 2, pages 464–466. IEEE Computer Society.

Hubert, H. and Stabernack, B. (2009). Profiling-based hardware/software co-exploration for the design of video coding architectures. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1680–1691.

Ismail, T. B., Abid, M., and Jerraya, A. (1994). Cosmos: a codesign approach for communicating systems. In *CODES '94: Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 17–24, Los Alamitos, CA, USA. IEEE Computer Society Press.

Jones, S., Andresen, C., and Crowley, J. (1997). Appearance based process for visual navigation. In *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 551–557 vol.2.

Joost, R. and Salomon, R. (2006). Hardware-software co-design in practice: A case study in image processing. In *In Proceedings of the 32 nd Annual Conference of the IEEE Industrial Electronics Society (IECON)*.

Joulain, C., Gaussier, P., Revel, A., and Gas, B. (1997). Learning to build visual categories from perception-action associations. In *Intelligent Robots and Systems, 1997. IROS '97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 857–864 vol.2.

Kalomiros, J. and Lygouras, J. (2008). Hardware implementation of a stereo co-processor in a medium-scale field programmable gate array. *Computers & Digital Techniques, IET*, 2(5):336–346.

Kalomiros, J. and Lygouras, J. (2009). A reconfigurable architecture for stereo-assisted detection of point-features for robot mapping.

Kaplan, A. and Kastner, R. (2000). A survey of hardware/software system partitioning. In *UCLA Computer Science Department and UCSB Electrical and Computer Engeneering Department*.

Lee, G. G., Chen, Y.-K., Mattavelli, M., and Jang, E. (2009). Algorithm/architecture co-exploration of visual computing on emergent platforms: Overview and future prospects. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19(11):1576–1587.

Madsen, J., Grode, J., Knudsen, P., Petersen, M. E., and Haxthausen, A. (1997). Lycos: the lyngby co-synthesis system. In *Design and Automation for Embedded Systems*, volume 2, pages 195–236.

Martin, J. and Crowley, J. L. (1995). Experimental comparison of correlation techniques. In *IAS-4, International Conference on Intelligent Autonomous Systems*.

Matsumoto, Y., Inaba, M., and Inoue, H. (1996). Visual navigation using view-sequenced route representation. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 83–88.

Matsumoto, Y., Inaba, M., and Inoue, H. (1999). Visual navigation using omnidirectional view sequenced. *Proceedings IEEE International Conference on Robotics and Automation 1999*, 1:75–80.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8).

Pedroni, V. A. (2004). *Circuit Design with VHDL*. MIT Press, Cambridge, MA, USA.

Pei, L., Xie, Z., and Dai, J. (2010). Fast normalized cross-correlation image matching based on multiscale edge information. In *Proceedings of International Conference on Computer Application and System Modeling*, volume 2, pages 507–511. IEEE Computer Society.

Pellerin, D. and Taylor, D. (1997). *VHDL made easy!* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Russ, J. C. (2002). *Image Processing Handbook, Fourth Edition*. CRC Press, Inc., Boca Raton, FL, USA.

Se, S., Lowe, D., and Little, J. (2002). Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21:735–758.

Skliarova, I. and Ferrari, A. B. (2003). Introdução à computação reconfigurável. *Revista do DETUA*, 2(6).

Tao, Z., Feng-ping, Y., and Hao-jun, Q. (2010). An optimized high-speed high-accuracy image matching system based on fpga. In *Proceedings of the International Conference on Information and Automation*, pages 1107–1112. IEEE Computer Society.

Tong, J. G. and Khalid, M. A. S. (2008). Profiling tools for fpga-based embedded systems: Survey and quantitative comparison. *JCP*, 3(6):1–14.

Tsai, D.-M. and Lin, C.-T. (2003). Fast normalized cross correlation for defect detection. *Pattern Recogn. Lett.*, 24:2625–2631.

Vahid, F. and Gajski, D. D. (1995). Clustering for improved system-level functional partitioning. In *in International Symposium on System Synthesis*, pages 28–33.

Vahid, F. and Givargis, T. (2001). *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, Inc., New York, NY, USA.

Valderrama, C. A., de Lima, M. E., Cavalcante, S., and Barros, E. (2000). *Hardware/Software co-design: projetando hardware e software concorrentemente*. Escola de Computação, São Paulo, SP, Brasil.

Wang, X. and Wang, X. (2009). Fpga based parallel architectures for normalized cross-correlation. In *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering*, pages 225–229, Washington, DC, USA. IEEE Computer Society.

Wilmshurst, T. (2006). *Designing Embedded Systems with PIC Microcontrollers: Principles and Applications*. Newnes.

Wolf, W. (2006). *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Yalamanchili, S. (2001). *Introductory VHDL: From Simulation to Synthesis*. Prentice Hall Inc.

Yiannacouras, P., Rose, J., and Steffan, J. G. (2005). The microarchitecture of fpga-based soft processors. In *CASES '05: Proceedings of the 2005 int. conf. on Compilers, arch. and synthesis for emb. sys.*, pages 202–212, New York, NY, USA. ACM.

Yonghong, Z. (2010). An nprod algorithm ip design for real-time image matching application onto fpga. *Electrical and Control Engineering, International Conference on*, 0:404–409.

APÊNDICE

A

Artigos Publicados

Durante o desenvolvimento deste trabalho de pesquisa alguns artigos foram publicados. Relacionados diretamente com o trabalho de pesquisa:

- DIAS, M. A. ; OSORIO, F. S. . Hardware/Software Co-design for Image Cross-Correlation. In: First Internation Conference on Integrated Computer Technology, 2011, São Carlos - SP. Integrated Computer Technology - Communications in Computer and Information Science. Berlin Heildelberg : Springer-Verlag, 2011. v. 165. p. 161-175.

- DIAS, M. A. ; Sales, D. O. ; OSORIO, F. S. . A Profile-Based Method for Hardware/Software Co-Design Applied in Evolutionary Robotics Using Reconfigurable Computing. In: IEEE Electronics Robotics and Automotive Mechanics Conference (CERMA), 2010, Cuernavaca. Proceedings of the CERMA conference 2010. Los Alamitos, CA : Conference Publishing Services IEEE Computer Society, 2010. v. 1. p. 463-468.

- DIAS, M. A. ; Sales, D. O. ; OSORIO, F. S. . HW/SW Co-design Architecture for Evolutionary Robotics. In: Latin American Robotics Simposium - Joint Conference 2010 - SBIA - SBRN - JRI (LARS/ENRI), 2010, São Bernardo do Campo - SP. Proceedings of The 7th Latin American Robotics Simposium, 2010. v. 1. p. 1-6.

Trabalhos em conjunto com o Laboratório de Robótica Móvel:

- PESSIN, G. ; OSORIO, F. S. ; DIAS, M. A. ; Souza, J. R. ; Neto, D.F. . Avaliacao de Tecnicas de Otimizacao Aplicadas ‘a Formacao e Atuacao de Grupos Roboticos. In: III Workshop on Computational Intelligence - Joint

Conference 2010 - SBIA - SBRN - JRI (LARS/ENRI), 2010, São Bernardo do Campo. Joint Conference - Processdings of Workshops, 2010. v. 1. p. 495-499.

- Fernandes, L. C. ; DIAS, M. A. ; OSORIO, F. S. ; WOLF, D. F. . A Driving Assistance System for Navigation in Urban Environments. In: Ibero-American Conference on Artificial Intelligence – IBERAMIA 2010, 2010, Mar Del Plata. Advances in Artificial Intelligence - Lecture Notes in Computer Science - IBERAMIA 2010. Heidelberg : Springer, 2010. v. 6433. p. 542-551.

- PESSIN, G. ; OSORIO, F. S. ; WOLF, D. F. ; DIAS, M. A. . Genetic Algorithm Applied to Robotic Squad Coordination.. In: CERMA 2009 - Electronics, Robotics and Automotive Mechanics Conference, 2009, Cuernavaca. Proceedings of the CERMA Conference 2009. Los Alamitos, CA : IEEE Press / IEEE Computer Society, 2009. v. 1. p. 169-174.

Trabalhos em conjunto com o Laboratório de Computação Reconfigurável:

- Silva, B. A. ; DIAS, M. A. ; Silva, J. L. ; OSORIO, F. S. . Genetic Algorithms and Artificial Neural Networks to Combinational Circuit Generation on Reconfigurable Hardware. In: IEEE Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on, 2010, Cancun. Proceedings of IEEE ReConFig 2010, 2010. v. 1. p. 179-184.

Trabalhos anteriores relacionado com o tema:

- DIAS, M. A. ; Lacerda, W. S. . Hardware/Software Co-Design Using Artificial Neural Networks and Evolutionary Computing. In: V Southern Conference on Programmable Logic, 2009, São Carlos. Proceedings of V SPL. São Carlos : Rima Editora, 2009. v. 1. p. 153-157.