UNIVERSIDADE FEDERAL DE OURO PRETO

Uma Abordagem do Problema de Particionamento Hardware e Software para Design de Sistemas Computacionais Wearables

Rodolfo Labiapari Mansur Guimarães
Universidade Federal de Ouro Preto

Orientador: Dr. Ricardo Augusto Rabelo Oliveira

Dissertação submetida ao Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto para qualificação ao título de Mestre em Ciência da Computação

Ouro Preto, Setembro de 2017

Uma Abordagem do Problema de Particionamento Hardware e Software para Design de Sistemas Computacionais Wearables

Rodolfo Labiapari Mansur Guimarães
Universidade Federal de Ouro Preto

Orientador: Dr. Ricardo Augusto Rabelo Oliveira

Uma Abordagem do Problema de Particionamento Hardware e Software para Design de Sistemas Computacionais Wearables

Resumo

Este trabalho constitui-se de uma abordagem do problema de particionamento *hardware* e *software* para dispositivos vestíveis com foco em aumento de performance, visando o gasto de recursos em seu *trade-off*. A tecnologia digital de fatores como microeletrônica, sensores e comunicação móvel são constantemente melhoradas à medida que a informação torna-se cada vez mais sem-fio, tornando-a um grande estímulo para o desenvolvimento de dispositivos inteligentes e conectados como sistemas embarcados IoT ou vestíveis (do inglês *wearables*). Isso é visto no rápido desenvolvimento de dispositivos para comércio, entretanto, ainda com a dificuldade de satisfazer os requerimentos das várias aplicações modernas. Tais dispositivos utilizam de um leque enorme de sensores e necessitam de um serviço autônomo, o que implica numa grande demanda de performance somado com o baixo consumo de energia sem deixar design, segurança e confiabilidade a desejar. Para obter um produto de alta qualidade, atendendo aos requisitos solicitados, esta pesquisa consiste na análise de projetos para sistemas *wearables* utilizando plataformas FPGA como meio na realização de particionamentos *hardware* e *software* obtendo um bom *speedup* e eficiência em comparação a outros sistemas como *in-software*.

Palavras-chave: Particionamento *hardware* e *software*, Sistema *Wearable*, Sistemas Embarcados.

An Approach of Hardware and Software Partitioning Problem for the Design of Wearable Computing Systems

Abstract

This work consists of an approach of the hardware and software partitioning problem for wearable devices focused on performance enhancement, aiming at the expense of resources in their trade-off. The digital technology of factors such as microelectronics, sensors and mobile communication are constantly improved as information becomes increasingly wireless, making it a great stimulus for the development of intelligent and connected devices like embedded IoT or wearable systems. This is seen in the rapid development of devices for commerce, however, still with the difficulty of satisfying the requirements of the various modern applications. Such devices use a wide range of sensors and require autonomous service, which implies a great demand for performance coupled with low power consumption without leaving design, security and reliability to be desired. In order to obtain a high-quality product, meeting the requested requirements, this research consists of the analysis of designs for wearables systems using FPGA platforms as a means of realizing hardware and software partitions obtaining good speedup and efficiency compared to other systems such as in-software.

Keywords: Hardware and Software Partitioning, Wearable System, Embedded System.

Sumário

Lista de Figuras

Nomenclaturas

ASIC	*Application-Specific Integrated Circuit*
CI	*Circuito Integrado*
CUDA	*Compute Unified Device Architecture*
CPU	*Central Processing Unit*
CSoC	*Configurable System on a Chip*
DRES	*Dynamically Reconfigurable Embedded Systems*
FPGA	*Field-Programmable Gate Array*
FPLD	*Field-Programmable Logic Device*
GA	*Genetic Algorithm*
GC	*Grafo de Chamada*
GFC	*Grafo de Fluxo de Controle*
GPU	*Graphical Processing Unit*
HDL	*Hardware Description Language*
HLS	*High-Level Synthesis*
IoT	*Internet of Things*
LED	*Light-Emitting Diode*
LUT	*Look-Up Table*
NRE	*Nonrecurring Engineering*
PLD	*Programmable-Logic Device*
RAM	*Random-Access Memory*

SRAM	*Static Random-Access Memory*
VGA	*Video Graphics Array*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuits*

Capítulo 1

Introdução

O recente progresso na obtenção e manipulação de informações, em conjunto com a ascensão da tecnologia da microeletrônica, comunicação e sensores, proporcionou um enorme estímulo no desenvolvimento de sistemas computacionais inteligentes de comunicação e *wearables*, todos sendo dispositivos embarcados (Jóźwiak 2017). O projeto de sistemas computacionais está mais complexo que nunca. A demanda por curto tempo para disponibilidade ao mercado, somado ao fato dos produtos exigirem propriedades de corretude, rapidez, confiabilidade e preço acessível, representam um desafio para projetistas de sistemas embarcados em geral. Sistemas computacionais embarcados (também nomeados pela literatura como sistemas embarcados ou sistemas embutidos) possuem muitos componentes implementados tanto em *hardware* e *software* e esta decisão de escolha de local de implementação será o tema principal abordado neste documento com foco em dispositivos *wearable*.

Os sistemas computacionais embutidos são produtos que utilizam processadores ou controladores e podem estar desde fornos de microondas, até sistemas de controle de aeronaves. São computadores situados em dispositivos e que sua presença não é imediatamente óbvia. Requerem técnicas na qual difere das utilizadas para *design* de computadores de propósito geral ou aplicações de *software* dessas máquinas (Wolf 1994). Em 2015, foi previsto um total de 6,5 bilhões de dispositivos ativamente conectados para o ano de 2016, segundo notícias da empresa de pesquisa e consultoria Gartner (Rob van der Meulen 2015).

No âmbito de sistemas embutidos, existe um subconjunto na qual possui o propósito de integrar-se ao sistema corporal expandindo suas capacidades. Esses dispositivos

'vestíveis', chamados de sistemas *wearables*, envolvem grande volume de dados de múltiplos sensores ou outros sistemas e são requeridos para prover serviço autônomo, contínuo, em um longo período de tempo. Diferentemente de *smartphones* e *tablets*, os *wearables* são tecnologias eletrônicas ou computadores que são incorporados ao vestuário dos usuários criando uma integração cada vez mais intensa entre tecnologia e ser humano. Sua tendência é superar os dispositivos manuais como computadores e telefones pois são dispositivos mais sofisticados pela existência de recursos sensoreais e escaneamento. Cerca de 20% da população possui pelo menos um dispositivo sendo que 10% utiliza-o todos os dias (Lee et al. 2016). São dispositivos que permitem benefícios como um estilo de vida inspirado em dados *fitness* até realidade aumentada preenchida por objetos virtuais. Tais dispositivos demandam de uma alta performance e/ou baixo consumo de energia, sem apresentar *trade-off* de confiabilidade e segurança entre outros. Segundo o trabalho de Jóźwiak (2017), com computação de alta performance e estudos em gasto energético eficiente, foi possível a facilitação do rápido progresso da computação móvel e autônoma. Sistemas computacionais *wearables* podem ser definidos como 'embutidos que foram inseridos em ambiente *mobile* de seus usuários, não exercendo a mesma atividade' e seu propósito é criar dispositivos com acesso constante, conveniente, portátil e principalmente *hands-free*, aplicando não só na área de saúde e esportiva, mas também em entretenimento e assim por diante.

1.1 Problema

A redução do ciclo de comercialização de um produto e o aumento de sua eficiência de desenvolvimento de projeto tem se tornado uma preocupação na área de *design* de sistemas embarcados, incluindo também os *wearables*. E por este motivo, a técnica de particionamento *hardware* e *software* tem sido uma das principais tecnologias para o desenvolvimento de sistemas embarcados em geral, desde que, este afeta a performance do sistema como um todo. Para Hassine et al. (2017), uma das soluções mais elegantes na computação que provê otimizações sistêmicas sobre essas circunstâncias é por meio do particionamento *hardware* e *software*. Segundo Wolf (1994), sistemas embutidos gerais são considerados únicos pelo fato da necessidade de um *codesign* de *hardware* e *software*. As pesquisas em *codesign* de *hardware* e *software* têm como objetivo o *design* de sistemas heterogêneos, visando performance, custo e metas de confiabilidade Edwards and Forrest (1994).

Hidalgo and Lanchares (1997) dizem que o objetivo principal é o balanceamento de todas as tarefas de forma a otimizar alguns objetivos de sistema sobre determinadas restrições. Agrupar específicos conjuntos de instruções de uma aplicação e então mapear esses grupos tanto em *hardware* e *software*.

Um desafio de *design* é combinar a flexibilidade de demanda pelos vários ambientes e aplicações, e a alta performance exigida em tarefas com o baixo consumo de energia requerido para maximizar o tempo de uso da bateria.

A motivação é que, tal problema que envolve *design* colaborativo e multidisciplinar, é um passo-chave no *design* de produtos modernos (Trappey et al. 2016). Implementações que baseiam-se somente em módulos de *software* possuem mais flexibilidade e menos custosos, entretanto, seu custo eleva-se em termos de tempo de execução. Uma implementação de *hardware* customizada de um conjunto de computações proverá uma eficiência energética e *speedup* maior relativa à implementações em *software* (Canis et al. 2011, Hassine et al. 2017, Stone et al. 2010, Wolf 1994, Zhang, Luo, Zhang, Li and Wang 2008).

A tendência hoje para *design* é na combinação das funções do processador com os recursos dos arranjo de portas programáveis em campo (FPGAs, do inglês *Field-Programmable Gates Array*) formando um sistema computacional híbrido. Ao utilizar do FPGA para o problema de particionamento, é possível acelerar uma aplicação usando recursos de *hardware* melhorando no desempenho e eficiência energética em comparação com o *software* executado inteiramente em um processador (Cong and Zou 2009, Lo et al. 2009, Zhang, Betz and Rose 2008).

Este trabalho então, propõe o estudo e aplicação de técnicas de particionamento dentro do conjunto de aplicações e sistemas *wearable*, na qual, possui restrições significativas como bateria, recursos e custo.

1.2 Objetivos da Dissertação

Este trabalho tem como objetivo a abordagem do particionamento *hardware* e *software* bem como sua importância no mundo de sistemas computacionais embutidos, com foco em sistemas *wearables* apresentando algumas soluções utilizadas atualmente.

- Introdução de sistemas computacionais *wearables* e apresentação do problema de

particionamento *hardware* e *software* no âmbito de de sistemas computacionais embutidos, com foco em sistemas *wearables*;

- Apresentação das principais soluções apresentadas ao logo dos anos e as utilizadas atualmente, bem como as ferramentas HLS como LegUp e OpenCL para a geração de sistemas computacionais que usufruem de aceleradores em *hardware*.

- Exposição da abordagem metodológica a ser utilizada para a procura da solução do problema de particionamento *hardware* e *software* apresentado.

1.2.1 Justificativa

A justificativa para a realização deste é que, além do tema de particionamento ser proposto recentemente, com o desenvolvimento de *designs* mais complexos, esse problema de decisão torna-se cada vez mais desafiador para os *designsers* de projeto, no qual o grande requisito para eficiência necessariamente segue junto com a alta velocidade de processamento (Arato et al. 2005, Trindade and Cordeiro 2016, Yan et al. 2017).

Dessa forma, pesquisá-lo com foco em sistemas *wearable* torna-o ainda mais necessário pela pouca atenção dada pelo meio científico até o presente momento, como será demonstrado no decorrer do documento, principalmente pelo ampla utilização que este tipo de produto vem ganhando no espaço comercial.

1.2.2 Contribuição

O trabalho consiste numa busca sobre o aprimoramento de performance de um dispositivo computacional *wearable* visando os respectivos gastos relativos ao uso de aceleradores em *hardware* como recursos, gasto energético, área utilizada e outros itens que serão discutidos neste documento.

1.3 Organização da Dissertação

Os demais capítulos deste trabalho estão organizados da seguinte forma: No Capítulo 2 é apresentada uma revisão bibliográfica do problema, abordando alguns métodos de fundamental importância para este trabalho. No Capítulo 3 são descritos alguns trabalhos

relacionados ao tema abordado. Nos Capítulos 4 e 5 são apresentadas as metodologias, sendo a primeira a exibição do processo de *design* e a segunda a metodologia proposta. No Capítulo 6 são apresentadas as conclusões parciais e propostas de trabalhos futuros.

Capítulo 2

Referencial Teórico

Neste capítulo será descrito os principais conceitos necessários para o entendimento base dos fundamentos do trabalho, além de suas tecnologias e metodologias.

2.1 Unidade de Processamento Central para Wearables

Existem inúmeros tipos de processadores específicos para sistemas embutidos. A classe de processadores mais baratos e de baixo gasto energético usufruem de arquitetura geralmente de 8, 16 ou 32 bits na qual podem executar centenas de milhões de instruções por segundo. Outra classe é na utilização de processadores de ponta, na qual são integrados a videogames ou *switches* de rede. São processadores que possuem um custo elevado e executam uma taxa de bilhões de instruções por segundo, em comparação com o anterior.

Segundo Hennessy and Patterson (2011), por mais que exista um leque de processadores para sistemas embutidos, o preço é um dos fatores mais importantes para a *design* de um projeto, sendo tão importante quanto o requisito de desempenho pois necessita-se de um desempenho elevado a um preço reduzido.

Tais processadores possuem o fim de serem aptos a cumprir os requisitos de uma aplicação *wearable* como por exemplo o requisito de tempo real, na qual um segmento da aplicação possui um tempo de conclusão máximo absoluto. Outras situações também são permitidas, por exemplo um caso mais sutil na qual permite-se um tempo médio de restrição ao invés de um valor fixo, chamada de tempo real flexível.

Com o uso de sistemas processados, a aplicação é executada totalmente em nível de *software* por meio de instruções. Ou seja, ela é construída/compilada e executada diretamente pelo processador ou por um sistema operacional que intermedeia a aplicação com o seu fim, não usufruindo de nenhum aprimoramento oriundo de aceleradores em *hardware*. Além disso, todo este conjunto de sistema deve-se ser planejado junto com a quantidade de memória disponível para uso no *wearable* pois, sabe-se que, quanto maior os recursos de memória, maior o seu gasto energético.

2.2 Field-Programmable Logic Device (FPGA)

Até recentemente, os *hardwares* reconfiguráveis eram utilizados unicamente na protitipação de projetos de circuitos integrados de aplicação especifica (ASIC, do inglês *application-specific integrated circuit* e produção em baixo volume por causa de sua baixa velocidade e custo por unidade. Entretanto, com a variedade desses dispositivos disponibilizados hoje no mercado, em conjunto com a elevação do custo de engenharia não recorrente (NRE, do inglês *Nonrecurring Engineering*, que refere-se ao custo de pesquisa, *design*, desenvolvimento e teste de um novo produto e exigências de mercado), houve um crescente interesse na utilização de FPGAs para sistemas embutidos devido suas vantagens sobre ASICs em termos de flexibilidade de projeto e custo zero de engenharia não recorrente citada (Mei et al. 2000). Tais dispositivos, juntos com sua plataforma de interação, de forma geral, permitem ao *designer* de sistemas embutidos ter uma *lousa branca* em que possa implementar *hardwares* computacionais personalizados tão facilmente como o desenvolvimento de um *software*, como foi possível ilustrar na Figura 2.1.

Dessa forma, uma *plataforma FPGA* é um chip na qual, além de conter o componente FPGA, este está integrado à inúmeras interfaces e componentes, desde LEDs (do inglês *Light-Emitting Diode*) e *switchs* até porta Ethernet e interface vídeo VGA (do inglês *Video Graphics Array*) e seus respectivos circuitos e como possui recursos suficientes para circuitos complexos, é possível implementar funções de processamento de imagem, interfaces de rede, algoritmos criptográficos e processadores completo, cada projeto de acordo com os recursos por ela disponibilizados. Entretanto, enquanto configurar um *hardware* reconfigurável é uma tarefa fácil graças às ferramentas disponíveis hoje, criar um *design* de *hardware* inicial não é (Sass and Schmidt 2010).

A seguir, será descrito a tecnologia que consiste os *hardwares* reconfiguráveis, em

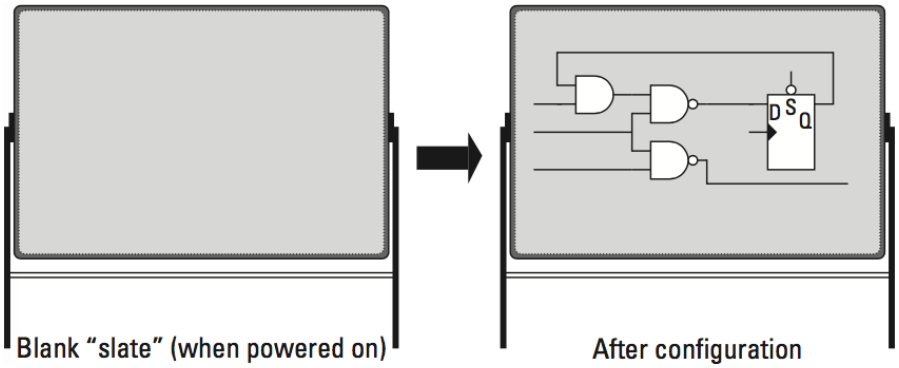

Figura 2.1: Ilustração em alto nível do funcionamento interno do FPGA. Fonte: Sass and Schmidt (2010).

especial o FPGA, e as respectivas linguagens de descrição de *hardware*.

O FPGA, item a ser utilizado nesta pesquisa, constitui de vários módulos lógicos programáveis relativamente pequenos e independentes, interconectados para criar funções maiores. Cada módulo lida, normalmente, com até quatro ou cinco variáveis de entrada. A maioria dos FPGAs utilizam uma *look-up table* (LUT) para criar as funções lógicas desejadas. Uma LUT funciona como uma tabela-verdade, no sentido que a saída é programada para criar a função combinacional armazenando valores verdadeiros e falsos adequado a cada combinação de entrada. Os recursos de roteamento de sinal programável dentro do chip tendem a ser bem variados, com extensões de caminhos diferentes disponíveis. Os atrasos de sinal em um projeto dependem do roteamento real de sinal selecionado pelo *software* de programação. Os módulos lógicos também contêm registradores programáveis. Eles não são associados a nenhum pino de entrada e saída (I/O, do inglês *Input and Output*). Em vez disso, cada pino de I/O é conectado ao bloco programável de entrada e saída que, por sua vez, é conectado aos módulos lógicos com linhas de roteamento selecionadas.

Uma arquitetura geral simplista de FPGA é exibido na Figura 2.2. Nela os quadrados menores situado nas laterais são blocos de I/O que podem ser configurados para fornecer recursos de entrada, saída ou bidirecionais. Os quadrados maiores situados no interior são as LUTs, usados para guardar dados que entram ou saem e realizar as operações lógicas. Os canais que interligam os blocos entre si são estabelecidas por meio de conexões que passam pelas linhas e colunas nos canais entre esses blocos e possuem a funcionalidade de serem interconexões programáveis (Tocci et al. 2011). Hoje, esses dispositivos possuem milhões de portas de lógica programável, bilhões de transistores, além de outros blocos de *hardware* dedicados dedicados como rápidas memórias embar-

cadas e multiplicadores de ponto-fixo tornando-o um dos circuitos integrados (CI) mais densos existente (Choi 2016).

Segundo Tocci et al., tais maravilhas de flexibilidade de projetos digital podem fornecer uma série de opções de projeto sendo voltados para indústria e até mesmo educação. Ao utilizar tecnologia CMOS, o consumo de energia do chip é relativamente baixo comparado com outras tecnologias podendo ser confeccionado em nível de tensão elétrica, frequências e cargas para os sinais de I/O. O mercado fornece diferentes graus de velocidade de FPGA a fim de que o projetista utilize o mais adequado ao projeto. Entretanto, um dispositivo FPGA pode ser configurado para um número infinito de projetos e isso implica na não possibilidade de afirmar o montante de dissipação de energia para um dispositivo FPGA. Dessa forma, FPGAs são chips que podem ser programados instantaneamente para funções de qualquer circuito digital (Choi 2016).

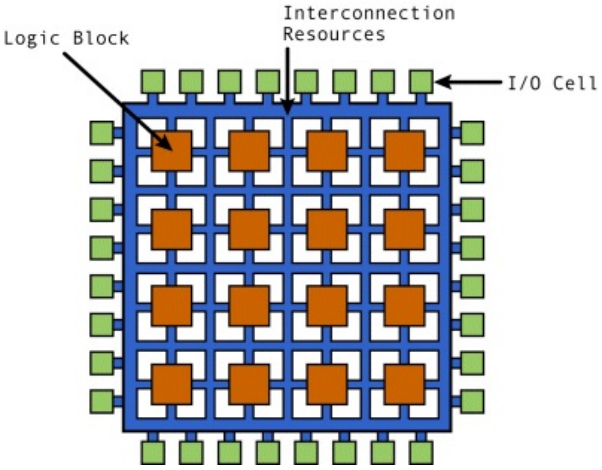

Figura 2.2: Exemplo da arquitetura internas de um FPGA. Fonte: `http://www.eetimes.com/document.asp?doc_id=1274496`. Acesso: 30/05/2017.

Plessl et al., Tocci et al. citam ainda que o motivo de PLDs estarem dominando o mercado é o fato de que, como são dispositivos programáveis, a mesma funcionalidade pode ser obtida com um circuito integrado (CI) ao invés de diversos circuitos individuais. Isso significa maior confiabilidade, menor espaço ocupado na placa, consumo de energia, complexidade de desenvolvimento e, geralmente, menor custo de fabricação.

A unidade de processamento central (CPU, do inglês *Central Processing Unit*) nesses tipos de sistema pode ser implementada em duas formas, sendo estas *hard* e *soft cores*. O primeiro é um *core* dedicado, ou seja, um pedaço de circuito integrado dentro (ou não) de um FPGA e o segundo é feito por meio da sintetização e mapeamento de um processador no FPGA com seus recursos lógicos, e assim, o processador é obtido por meio de *design* e sintetização na placa por meio das portas lógicas. Independente da forma de projeto da CPU, o FPGA será constituído da seguinte arquitetura exibida na Figura 2.3, chamada de SoC FPGA (do inglês, *System-on-Chip* FPGA). Nela, as setas representam os barramentos de comunicação entre os principais componentes.

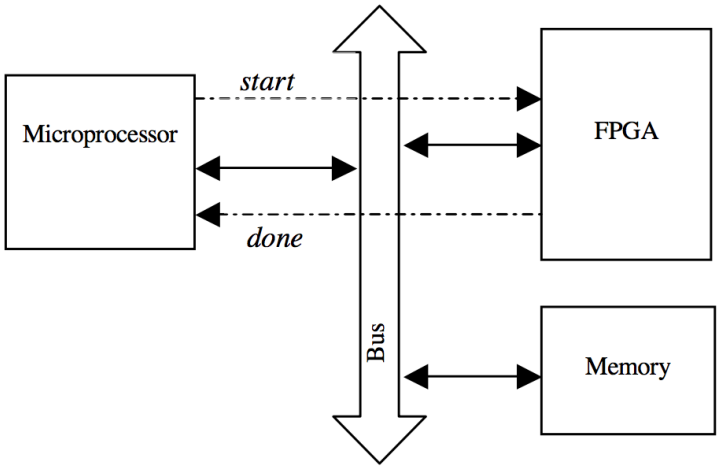

Figura 2.3: Visão geral de um SoC FPGA.

Cada um tipo de *core* possui suas vantagens. Ao utilizar um *hard core*, é possível utilizar todos seus recursos obtendo máxima performance nas atividades executadas, a utilização de um *soft core* permite a extensão da arquitetura (Plessl et al. 2003).

Um das maiores barreiras para o *design* de projetos em FPGA é a necessidade de uso de linguagens de descrição de *hardware*. Elas serão descritas a seguir.

2.2.1 Hardware Description Language (HDL)

Linguagem de Descrição de *Hardware* (do inglês *Hardware Description Language*) é uma das classes de linguagens de computação usados para descrever formalmente um circuito eletrônico. Uma expressão padrão baseada em texto HDL é capaz de descrever o comportamento temporal ou a estrutura de circuito (espacial) de um sistema eletrônico. Sua origem veio da necessidade de documentar o comportamento do *hardware*(Sass and Schmidt 2010).

HDL é amplamente utilizada em *design* de *hardware* especificando detalhes de *design* de chip tantos chips específicos quantos os próprios FPGAs. Para customizar algum tipo de chip digital, HDLs especifica um modelo para um comportamento específico de um circuito antes do circuito ser projetado e construído. Ferramentas lógicas de sínteses são invocadas em seguida para gerar informações geométricas que são utilizadas para produzir as máscaras *photolithographic*, necessárias para a fabricação do CI do projeto desenvolvido.

A utilização de uma HDL é o primeiro passo no processo de síntese em FPGA. O

código é entregue ao compilador lógico, chamado de ferramenta de síntese, e sua saída será carregada ao dispositivo reconfigurável. A propriedade única deste processo na qual fornece a lógica programável é que, com esse processo de síntese, é possível alterar o código HDL, compilá-lo e fazer sua síntese no mesmo dispositivo para testar, quantas vezes forem necessárias, sem custo adicional (Smith and By-Zamfirescu 1998).

Enquanto a maioria de engenheiros de *hardware* utilizam tanto o *VHDL* quanto o *Verilog*, na qual possuem um nível elevado de complexidade (Choi 2016), existe outras linguagens disponíveis para uso. Outras linguagens como *SystemC*, *HandelC* e *Impulse* fornecem a construção de sistemas *hardware* e *software* juntos, dando ao *designer* uma linguagem de alto nível para manuseio do projeto (Sass and Schmidt 2010).

2.2.2 High-Level Synthesis (HLS)

Sintetizador em Alto Nível (HLS, do inglês *High-Level Synthesis*) são procedimentos que sintetizam códigos em alto nível para HDLs.

Realizado uma especificação de *design* em *software*, um HLS pode reduzir os longos ciclos do processo de *design* de *hardware* e ainda trazer melhoria em performance e eficiência energética (Choi 2016).

As primeiras ferramentas sintetizadoras baseavam em linguagem *C*, mas não houveram um sucesso em seu uso pois os engenheiros de *hardware* acreditavam que existia uma lacuna entre o HLS e o *design* de *hardware* feito por humanos. A justificativa era que as ferramentas HLS não exploravam profundamente o recurso de paralelismo, além de que, para os engenheiros de *software* HLS continua sendo uma dificuldade já que muitas partes do projeto, como a integração do sistema, permanecem em grande parte como um processo manual.

Existem várias ferramentas para geração automática de código descritivo de *hardware*. Ferramentas como o *framework* LegUp e OpenCL (Trevett 2008) possuem o propósito de entregar um bom HLS além de tentar amenizar esses problemas de projetos.

LegUp High-Level Synthesis

LegUp é uma ferramenta que permite a utilização de técnicas de *software* para implementação de *design* de *hardware*. Com o *framework* LegUp, é possível utilizar um programa padrão C como entrada e automaticamente compila o programa para dispositivos FPGA.

É possível gerar procedimentos totalmente dispostos em *hardware* e outros híbridos na qual é utiliza-se de *soft* processadores, utilizando de um MIPS, e também gerando códigos para *hard* processador que utiliza circuitos ARM disponibilizados em FPGAs SoCs pela Altera sendo tais gerações usufruindo de aceleradores que comunicam por meio de interface de barramentos padronizados (Canis et al. 2011). Possui também um *framework* para *debug* e verificação de HLS (Fort et al. 2014).

OpenCL

OpenCL é um padrão que oferece uma API comum para execução de programas em sistemas compostos por diferentes tipos de dispositivos computacionais tal como processadores *multicores*, GPUs ou outros aceleradores.

Este padrão utiliza de coprocessadores como meio para operações aritméticas intensivas paralelas e com isso é possível realizar realizar uma computação heterogênea paralela em nível de tarefas e dados, nos vários dispositivos disponíveis hoje pelo mercado criando um novo nível de infraestrutura computacional paralela.

O OpenCL provê uma abstração fácil de uso e um amplo conjunto de APIs de programação baseada no sucesso obtido pelo CUDA (do inglês, *Compute Unified Device Architecture*) e outros ferramentas de programação. Ele define um *core* funcional que todos os dispositivos suportam incluindo uma mecanismo de extensão que permite os fabricantes exponham recursos de *hardware* exclusivos e interfaces de programação experimentais para benefícios dos desenvolvedores de aplicações.

De forma geral, um programa em OpenCL é executado num dispositivo computacional na qual os dispositivos podem conter um ou mais unidades computacionais (ou seja, *cores*) na qual são compostas de um ou mais elementos de processamento SIMD (do inglês, *single-instruction multiple-data*) (Stone et al. 2010). Como é possível visualizar na Figura 2.4, levando para o ambiente de FPGAs, existe um *host* que executa em um processador *soft* ou *hard* o *Runtime Environment* que fica responsável por controlar to-

dos seus dispositivos. Cada dispositivo conectado ao *host* pode conter vários *processing elements* que por sua vez pode conter vários *compute unit*.

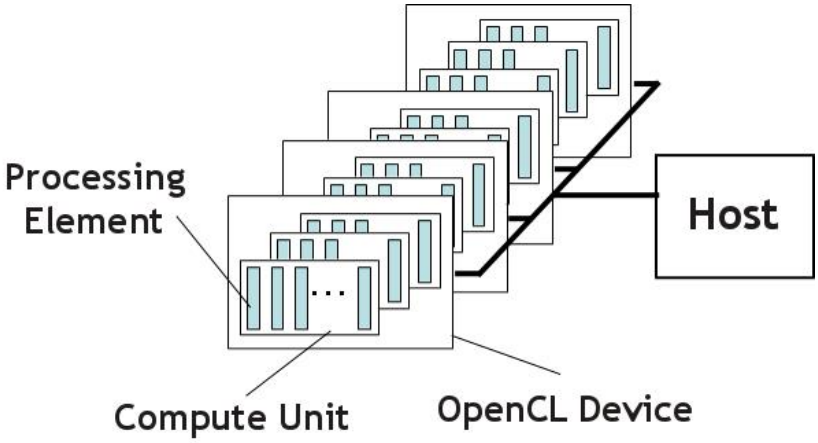

Figura 2.4: Sistema hierárquico de um projeto em OpenCL. Fonte: `https://handsonopencl.github.io/` Acessado em: 07/08/2017.

Todos os aceleradores descritos neste relatório, podem ser associados com a sintetização dos procedimentos de instruções no FPGA (Czajkowski et al. 2012, Shagrithaya et al. 2013).

Para prover um melhor suporte para o paralelismo em *hardware*, utilizaram de bibliotecas *multi-threads* como a *Pthread* (Barney 2009), *OpenMP* (*The OpenMP API specification for parallel programming* 2015) para criar aceleradores em *hardwares*paralelos.

É investigado otimizações em memória e arquitetura de sistema provendo melhorias em performance de circuito, área, utilizando uma abordagem do padrão produtor-consumidor para inferir circuito de transmissão em *hardware*.

2.3 Profile

Profile é uma procedimento para auxiliar o usuário a coletar informações do *software* em tempo de execução. Existem vários programas diferentes para adquirir essas informações e podem ser distinguidos em duas categorias. A primeira é aqueles que apresentam a quantidade de declarações e invocações de rotinas, e aqueles que exibem informações de tempo de declarações e rotinas (Nemeth et al. 2004). Um exemplo de saída é exibida a seguir.

```
Flat profile:
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 33.34      0.02     0.02     7208     0.00     0.00  open
 16.67      0.03     0.01      244     0.04     0.12  offtime
 16.67      0.04     0.01        8     1.25     1.25  memccpy
 16.67      0.05     0.01        7     1.43     1.43  write
 16.67      0.06     0.01                             mcount
  0.00      0.06     0.00      236     0.00     0.00  tzset
  0.00      0.06     0.00      192     0.00     0.00  tolower
  0.00      0.06     0.00       47     0.00     0.00  strlen
  0.00      0.06     0.00       45     0.00     0.00  strchr
```

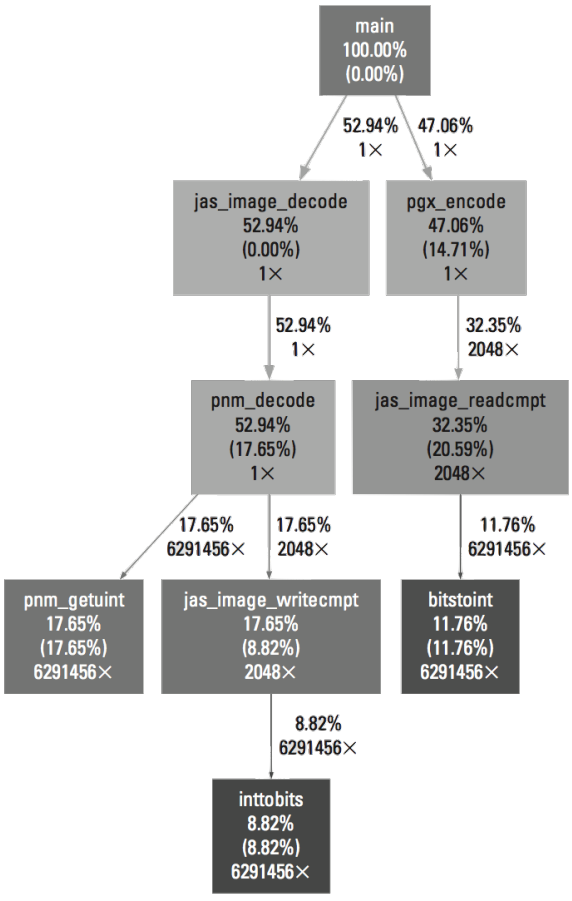

Figura 2.5: *Profile* da codificação de imagem em formato JPEG. Fonte: Sass and Schmidt (2010).

A Figura 2.5 exibe um exemplo gráfico de *profile* de *software* em um codificador de imagem em formato JPEG. O processo é feito ao colocar o *software* referencial (programa a ser coletado) como entrada representativa na ferramenta e a coleta é realizada em várias partes da aplicação ao longo de sua execução neste. Uma das técnicas do *profile* de mensurar uma aplicação é na realização de interrupções periódica no programa e amostrar o seu *program counter*. Dessa forma, é possível utilizar um histograma para contar quando um programa é interrompido em um endereço particular e a partir dessa informação, calcular a fração aproximada do tempo total de execução gasto em suas partes. Distribuições GNU/Linux possuem a ferramenta `gprof` na qual avalia procedimentos enviados por parâmetro, realizando o cálculo de tais informações de *software* (Graham et al. 1982).

2.4 Sistemas Computacionais Wearables

Sistemas computacionais *wearables* são sistemas que, com a possibilidade de ter um computador acoplado ao corpo, proporciona ao usuário um nível superior de informações contextualizadas dentro de um ambiente interativo (Amorim et al. 2017).

Com a tecnologia digital constantemente melhorando à medida que a informação se torna sem-fio, os avanços demandaram mais fatores móveis e *wearable* de produtos que possuem acesso à informação. Segundo Gemperle et al. (1998), um produto que é *wearable*, deveria ter sua '*wearability*' sendo este definido como a interação entre o corpo humano e o objeto *wearable* estendendo ao corpo em movimento.

Devido ao movimento de seus usuários, um computador *wearable* é embutido em um ambiente *mobile* e necessita-se da interação com o ambiente ao seu redor. Como é exibido na Figura 2.6, um sistema *wearable* pode ser composto por um conjunto de nós distribuídos e uma rede de comunicação centralizada num módulo principal, sendo possível determinar, por exemplo, a geração de eletricidade por meio de geradores *piezo-electric* integrados aos sapatos, energizando uma parte do sistema *wearable* com o caminhar do usuário (Kymissis et al. 1998) ou a ação do usuário por meio de acelerômetros (Kern et al. 2002, Van Laerhoven et al. 2002).

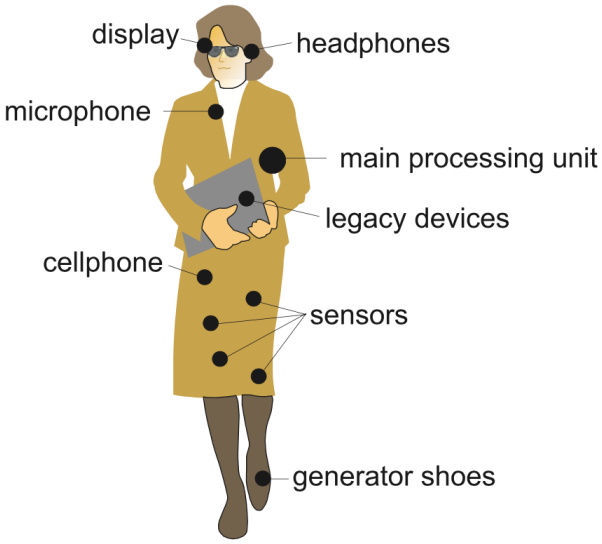

Figura 2.6: Exemplificação de alguns tipos de dispositivos *wearables*. Fonte: Plessl et al. (2003).

Com a distribuição espacial dos módulos pelo corpo, a comunicação torna-se um item importante em termos consumo de energia (Kymissis et al. 1998). A rede de comunicação é uma mistura de conexões cabeadas e sem-fios. Para dispositivos *wearables*, a comunicação sem-fio é a tecnologia predominante por causa da necessidade de mobilidade (Plessl et al. 2003).

A introdução desses sistemas no ambiente de pesquisa não é nova como é reportado por Sutherland (1968), Mann (1996) e Mann (1997). Entretanto, a aplicação desses dispositivos, depende diretamente da miniaturização dos componentes eletrônicos. Esse

fenômeno fica claro ao perceber o crescente espaço ganho pelos *smartwatches*, *fitness trackers*, óculos, equipamentos de realidade virtual e aumentada e muitos outros nas indústrias e nas atividades pessoais de usuários. Com esses novos dispositivos embarcados de propósito geral miniaturizados, aumenta-se a sua atração devido à fácil disponibilidade dos dispositivos, baixo preço e ferramentas de desenvolvimento disponíveis para desenvolvimento de aplicações específicas incluindo compiladores e sistemas operacionais para tal, mas não são otimizados para uso *wearable*. Isso pois, como Plessl et al. cita, muitos dos sistemas *wearables* construídos possuíam características diferentes de componentes que prestam auxílio à tarefas pessoais de usuários, citados anteriormente. Exemplos são o uso dados de teclados ou canetas para entrada de dados ou um *display* LCD, na qual contradiz com o paradigma de operações *hands-free* e a propriedade de computação *wearable* discreta. Sistemas de computação distribuídos no corpo construídos a partir desses equipamentos interativos são altamente ineficientes devido à falta de especialização de componentes individuais de propósito específico. Dessa forma, a computação *wearable*, seguindo esse conceitos, se define muito bem como um subconjunto de sistemas embutidos.

2.4.1 Característica de um Wearable

Entendido a localização de sistemas *wearable* sobre sistemas computacionais, a caracterização de um dispositivo *wearable* é feita acordando as classificações pré-estabelecidas em relação à suas funcionalidades e requisitos de *hardware*. O mercado possui um número considerável de dispositivos *wearables* que são utilizados em inúmeras áreas, e mesmo que cada equipamento separado tenha suas próprias características, muitas soluções em *hardware* compartilham uma arquitetura e organização interna de recursos implementados comum. Esses detalhes também podem ser expandidos às características relativas à recursos de sistemas operacionais, no qual dispositivos *wearables* podem ser classificados além de seus componentes de *hardware* internos como suas funções de performance (Amorim et al. 2017, Delabrida et al. 2016), como representado pela Figura 2.7.

De um lado desta gama de dispositivos existem os *dispositivos dependentes de interfaces de usuário* que possuem alta dependência com operações interfaces gráficas, provendo respostas sobre as interações do usuário. Esses dispositivos focam principalmente em tarefas para renderização em *displays* como por exemplo equipamentos de realidade virtual e aumentada para implementação de objetos tridimensionais. De outro lado, existem os *dispositivos dependentes de entrada e saída de sinais*. Esses dispositivos

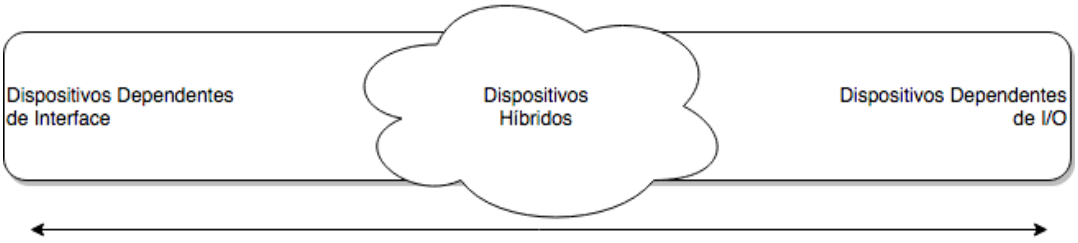

Figura 2.7: Classificação de sistemas *wearables*. De um extremo existe os dispositivos dependentes de interfaces de usuário e do outro os dependentes de entrada e saída de sinais enquanto entre eles, a gama de combinações possíveis. Fonte: Adaptado de (Amorim et al. 2017).

atuam principalmente com o estímulo por algum dado oriundo de outro dispositivo ou do ambiente assim entregue à nuvem respeitando restrições de tempo real. Isso se da pela utilização de sensores acoplados ao dispositivo que podem exigir uma boa vazão de dados e pequena latência como são os monitores de atividade remota, situação na qual cria-se o ambiente perfeito para o termo conhecido por IoT (do inglês *Internet of Things*).

Entre dispositivos dependentes de interfaces de usuário e de entrada e saída de sinais existe os híbridos. São dispositivos que utilizam recursos de ambos os extremos citados. Dispositivos como *smartwatches* e *fitness trackers* são exemplos de tais dispositivos híbridos no qual possuem restrições de equalização à prioridade dada pelo dispositivo para ambas as operações de interface e sinais.

A separação dos conceitos computação *wearable* e IoT ainda não estão claros segundo a bibliografia. Sistemas operacionais de propósito específico para ambientes *wearable* são comumente focado em um único tipo de seguimento de produto como os *smartwatches*, sendo que proporciona aos desenvolvedores um meio para sua aplicação final além de entregar um produto de alta qualidade. Também, atualmente, não existe nenhum sistema que satisfaça todos os requisitos apresentados (Amorim et al. 2017).

Já Jóźwiak (2017) caracteriza um sistema *wearable* como um sistema *cyber*-físico[1] móvel autônomo. São sistemas que podem ter mobilidade inerente ou poder ser transportado para outro sistema, industrial ou natural (incluindo humanos), sendo autônomos em termos de funcionalidade. Podem ser utilizados para aplicações de consumidores (computação móvel), extensões ou reposições de capacidades humanas, sistemas sociais (*health-care* inteligente), automotivo, industrial (monitoramento) e aplicações comerciais

[1]Sendo *cyber*- uma combinação dos termos 'computador', 'rede de computadores' ou 'realidade virtual' com um segundo termo, no caso o 'físico' oriundo de circuitos.

como realidade aumentada para informações turísticas. Eles representam uma grande parte da heterogeneidade de sistemas embarcados, cobrindo muitos campos e tipos de aplicações que vão desde um dispositivo inteligente integrado à roupa, focado no campo de computação *mobile* pessoal, até indústrias como dispositivos de segurança. Esses dispositivos podem também trabalhar de forma colaborativa com *smartphones*, redes e outros sistemas criando um sistema mais complexo.

Segundo Van Laerhoven et al. (2002), a distribuição de elementos computacionais, como sensores, em objetos mundanos em nosso cotidiano se adéqua à pesquisa denominada por computação ubíqua. Isso implica também que computação *wearable* também não foge deste arcabouço, uma vez que superfícies de roupas são uma plataforma de suporte ideal para uma grande quantidade de sensores (desde que sejam miniaturizados para que eles não obstruam o usuário). Essa restrição de tamanho geralmente significa que a própria qualidade do equipamento também está comprometida, o que leva ao conceito de muitos atuadores e sensores simplistas.

Sendo sistemas *wearables* uma subclasse de sistemas embutidos, estão sujeitos à várias restrições de *design*, sendo elas performance em multi-nós, gasto energético consciente e alta flexibilidade (Plessl et al. 2003):

Performance de multi-nós: Requer uma performance base fixa para tarefas que não mostram altas demandas computacionais nem restrições de tempo rigorosa. Sistemas *wearables* executam rajadas de tarefas de computação intensiva que consideram restrições de tempo-real. Não realizando as tarefas, o sistema torna-se inaplicável.

Gasto energético consciente: É essencial em sistema deve-se manter ativo e funcional num certo período de tempo. O *design* do gasto de energia conduz inúmeros desafios como gerenciamento computacional energético eficiente e energização dinâmica.

Diferenciando os termos de baixo custo de energia e eficiência energético, o consumo de energia é mensurado pela divisão da dissipação energética pelo tempo mensurado. A eficiência energética relaciona o total de energia necessário para computar uma tarefa específica sendo o desafio a construção de um *design* na qual possui-se alta eficiência energética para um dado conjunto de tarefas. O procedimento dinâmico de energização consiste na tarefa de associar determinada tarefa para o componente mais eficiente (energeticamente falando) disponível e forçar todos os outros não utilizados pelo sistema a serem postos em modo de economia de energia ou desligá-los quando apropriadamente.

Flexibilidade: Quando menciona-se sobre alta flexibilidade, é considerado o fato de que o dispositivo será utilizado em situações altamente dinâmicas. Isso fica claro na necessidade na qual os requisitos de aplicação variam de acordo com as escolhas do usuário, mas também com o contexto e local utilizado. Outro, é no fato de que o usuário troca de roupas constantemente e com isso os dispositivos devem ter a capacidade de serem acoplados e removidos, neste caso. Isso além de critérios como confiabilidade, disponibilidade e fatores dependentes de sua forma como volume e peso.

Dessa forma, pode-se estabelecer que os requisitos flexíveis em um sistema *wearable* demanda um sistema de computação programável de propósito geral, enquanto os requisitos de alta performance e consumo de energia consciente demandam um sistema computacional especializado. Assim, como meio para solução desses problemas, Plessl et al. utilizaram-se de um *hardware* reconfigurável para incorporar ao sistema. O trabalho exibe um sistema *wearables* compreendendo de um processador de até médio porte em termos de processamento e módulos reconfiguráveis. A utilização de *hardware* reconfigurável nos permite alcançar alto processamento com maior eficiência energética em relação à processadores para tarefas de computação intensiva em tempo real.

Capítulo 3

Trabalhos Relacionados

O particionamento de forma geral é um problema de otimização na qual vários autores utilizam métodos para auxiliar na decisão de cada componente do *design* referencial de *software*. Utiliza-se tanto processos manuais quanto algoritmos exatos ou heurísticos (Arato et al. 2005).

Os autores Edwards and Forrest (1994) utilizam do particionamento para aprimorar a performance do seu *software*. Em seu trabalho, realiza-se a tentativa de tratar regiões críticas da aplicação na qual uma solução em *software* não pode chegar à restrições de performance requeridas e uma solução em *hardware* deve ser encontrada, ou a performance total pode ser acelerada pela implementação de uma região crítica em *hardware*. Apresenta-se uma metodologia para desenvolvimento *codesign* no qual consiste na construção do código de uma aplicação em *C* e as regiões críticas são identificadas e particionadas. Feito isso realizar-se mensurações do projeto e uma nova verificação de particionamento é realizada, como é exibido na Figura 3.1. Utiliza-se também de um FPGA para as mensuras de performance e a propriedade de reconfiguração para novos testes em *hardware*.

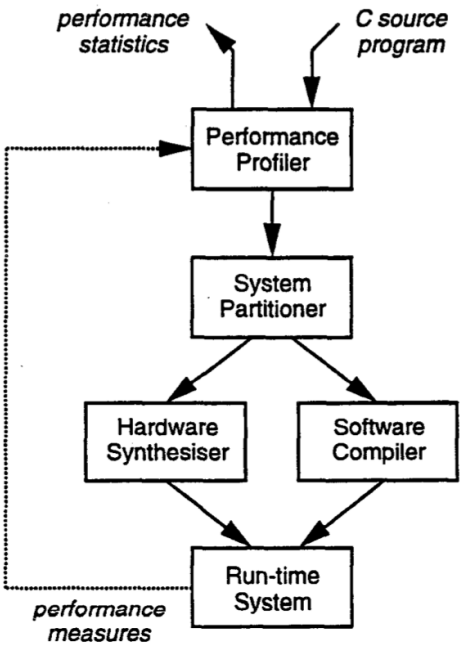

Figura 3.1: Metodologia de *codesign*. Fonte: Edwards and Forrest (1994).

Já o trabalho de Stitt et al. (2003) procura uma abordagem utilizando métodos de otimização de *software* dinâmicos, introduzindo a primeira abordagem para particionamento de *hardware* e *software* dinâmico. O processo consiste na detecção da região de *software* mais frequentemente executada e reimplementa-a em *hardware* de FPGA. Afirmam que a utilização de particionamento dinâmico trás uma série de vantagens comparado com abordagens tradicionais manuais.

Existem vários trabalhos que propuseram algoritmos exatos baseados em estratégias *branch-and-Bound* (Jigang and Thambipillai 2004, Mann et al. 2007, Strachacki 2008), programação dinâmica (Madsen et al. 1997, Wu and Srikanthan 2006) e linear inteira (ILP, do inglês *integer linear programming*) (Niemann and Marwedel 1997).

Nematbakhsh et al. (n.d.) parte para o exame da relação entre o *footprint* gerado para o FPGA e o *speedup* do *software* na situação na qual um FPGA é utilizado para a implementação de *loops* e sub-rotinas críticas. Como utilizam uma abordagem direta, utilizaram de ferramentas protótipos e comerciais como *Synopys' Nimble Compiler* e *Proceler*, para a facilitação do processo.

Abordagens mais recentes como a de Yan et al. (2017) parte da otimização *position disturbed particle swarm* com otimização invasiva de *weed* como o método de particionamento *hardware* e *software*.

Wang et al. (2016) citam que o particionamento depende da exploração de caracterização, estimação e *design* espacial das métricas de custo e performance sistêmica. É também mencionado que o *codesign* nos dias de hoje é tão complexo que a simplificação do particionamento só para duas partes não é suficiente para a representação do problema como um todo. Sobre essa crítica, dissertam sobre a inclusão de parâmetros chave de *design* e uso de recursos que deveriam ser incorporados à modelagem do sistema e dessa forma, o trabalho proposto visa considerar a modelagem de incerteza para particionamento de sistemas com um conjunto melhorado de parâmetros para compartilhamento de recursos *hardware* e *software*. Esse terceiro item a ser considerado ao problema de particionamento podem ser definidos em três tipos, sendo esses: *a)* o conjunto de recursos necessários para particionar uma dada tarefa (sendo esses RAM, ROM, DPS, blocos IP do inglês, *intelectual propriety*, entre outros); *b)* possuem vários parâmetros de configurações que provê diferentes *trade-off* entre recursos e performances como por exemplo circuitos de multiplicações/divisões que podem ser sintetizados sequencialmente (pouca área e lento) ou combinatório[1] (muita área e rápido); *c)* a dificuldade da mensura

[1]Por *unrolling*, técnica de otimização de código no qual tenta reduzir ou eliminar as iterações de

de desempenho e impacto no uso de recursos em várias configurações de partição com precisão precisando ter em mente a partição com a natureza incerta dessas estimativas não precisas. A teoria da incerteza é uma abordagem que é um sistema matemático que é designado para modelar a indeterminação ao invés do uso da teoria da probabilidade. Isso pois, quanto tem-se muitas amostras de uma quantidade indeterminada, seria significativo a utilização da teoria de probabilidade ou *fuzzy* pela propriedade de serem contínuas entre o intervalo 0 (zero) à 1 (um). Entretanto, se não possui-se amostras suficientes para a estimação de uma distribuição de probabilidade, utiliza-se do conhecimento do domínio para avaliação do grau de crença de que cada evento indeterminado acontecerá, no caso, aplicado pela teoria da incerteza. E a teoria da incerteza é um ramo da matemática axiomática para modelagem de graus de crença, de modo geral.

Como outro trabalho recente que aborda o particionamento, Choi (2016) descreve um *framework* chamado LegUp. Com essa ferramenta é possível compilar um *software* e suas *threads* gerando um sistema *hardware-only* ou também um sistema híbrido particionado paralelo utilizando aceleradores, gerando também todos os itens necessários para tal como memórias sintetizadas e sistemas de interconexões. Utilizam duas técnicas de descrição de paralelismo em *software* sendo elas a *Pthreads* e a *OpenMP* sendo a primeira permite a sintetização de funções operando concorrentemente em *hardware* com aceleradores, e a segunda usada para gerar os próprios aceleradores executados concorrentemente em um sistema de compartilhamento de memória. Afirmam também que a sua ferramenta produz HDL de alta performance que pode ser comparado com circuitos que são gerados por ferramentas HLS comerciais. Os resultados obtidos pelos cientistas Canis et al. (2011) mostraram que a ferramenta consegue gerar produtos tão bons quantos ferramentas HLS comerciais.

Entretanto, além de todo arcabouço de algoritmos para escolhas para sistemas de propósito geral tal como os citados, existe ainda uma linha de pesquisa específica do problema na qual trata-se do particionamento em sistemas embutidos.

O desenvolvimento para sistemas *hardware* e *software* participativo para sistemas embutidos ou microcontroladores já é pesquisado amplamente como os trabalhos de Bolsens et al., Ernst et al., Gajski et al., Gupta et al., Hardt, publicados na década de 90. Mei et al. (2000) em seu trabalho, descreve um particionamento de *hardware* e *software* além de uma abordagem de escalonamento para sistemas embutidos dinamicamente reconfiguráveis (DRESs, do inglês *dynamically reconfigurable embedded systems*) no qual

determinado *loop* do código.

possuem como projeto um processador de propósito geral junto com um FPGA sendo este reconfigurável em tempo de execução para reduzir custos. Dessa forma, seu trabalho consiste numa análise de tempo de configuração e, como contribuição, a análise do tempo de reconfiguração parcial do FPGA. Para o escalonamento, Mei et al. utiliza um método baseado na heurística do algoritmo genético (GA, do inglês *genetic algorithm*) e num algoritmo de escalonamento de lista com melhorias. O escalonador desenvolvido atua como uma sub-rotina do algoritmo de particionamento. Ele é invocado no passo *evolution* do GA. Além do escalonamento já conhecido em processador e barramento sequenciais determinando a ordem e tempo início de execução, o algoritmo também deve fazer o escalonamento no FPGA. Porém, não só determinando o tempo de início da tarefa, mas sim sua posição no FPGA respeitando as restrições de recursos e precedentes, tornando assim o problema uma abordagem ao problema de alocação restrita.

Já Arató et al. (2003) descreve algumas versões diferentes do problema de particionamento, correspondendo à sistemas de tempo-real e custo restringido respectivamente, fornecendo análise matemática formal para o problema, provando que são problemas $\mathcal{NP}$-difícil. Apresentaram uma abordagem baseada na programação linear inteira resolvendo o problema de forma otimizada, mesmo para sistemas de grande porte, além de outra abordagem utilizando a heurística GA na qual encontrar soluções próximas ao ótimo global para sistemas largos.

Mann et al., em 2007, descreveram uma primeira tentativa para um algoritmo exato, não heurístico, para o problema de particionamento. Utilizam um esquema na qual implementa-se a estratégia *branch-and-bound* como um *framework*, permitindo o incremento de outros algoritmos. Em sua implementação, realizaram várias investigações para incrementar a eficiência do algoritmo, incluindo várias técnicas sendo elas: *lower bounds based on LP-relaxation*, uma mecânica de inferência customizada, condições não-triviais necessárias baseadas num algoritmo *minimum-cut*, e diferentes heurísticas com passos pré-otimizados. O algoritmo também pode ser generalizado a fim de incluir mais de uma restrição, podendo também o *designer* prescreva quais nós devem estar em qual nível de projeto. Eles demonstram que o produto pode resolver problemas de particionamento altamente complexos em tempo razoável. Citam ao final que o resultado obtido é em entorno de dez minutos mais rápido que algoritmos exatos anteriores baseados em programação linear inteira para os testes realizados.

Pesquisas mais recentes como a de Hassine et al., em 2017, procuravam aplicar otimizações sobre o tempo de execução e gasto energético para *cores* baseados em sistemas embarcados por meio de algoritmos de particionamento. O algoritmo proposto destina-se

a alcançar um particionamento de grafos à procurar o melhor conjunto da relação energia e tempo de execução. Testado em comparação com outros algoritmos heurísticos como *Simulated Annealing*, Busca Tabu e Genético, o algoritmo mostra-se ser melhor adequado para aplicações em *cores* baseados em sistemas embarcados que necessitam do equilíbrio no *tradeoff*.

Trindade and Cordeiro (2016) por exemplo utiliza do GA para solucionar o problema de particionamento em sistemas embutidos. Em seu trabalho, é proposto novas abordagens para o problema usando técnicas de verificação baseadas nas teorias de módulo de satisfação (SMT, do inglês *satisfiability modulo theories*). Apresentam um exemplo de particionamento, modelam e solucionam-o usando três diferentes técnicas sendo a principal ideia é aplicar mo método de verificação SMT ao particionamento *hardware* e *software*, e por fim, comparar os resultados com técnicas de otimizações tradicionais como ILP e GA.

Jóźwiak em seu *survey* publicado em 2017, considera vários aspectos de uma aplicação embutida, bem como suas tecnologias de *design* com foco sistemas móveis modernos e *wearables*. É citado dois paradigmas de desenvolvimento para sistemas embutidos sobre sistema de multi-processadores heterogêneos sendo eles o paradigma de sistemas *life-inspired* e sistemas *quality-driven*. O paradigma de sistemas *life-inspired* especifica princípios básicos, características e organização funcional e estrutural de um sistema embutido por meio da analogia à vida de um organismo inteligente, além de básicas soluções de mecanismos e arquiteturas de sistemas para implementar tais princípios. Já o paradigma de sistemas *quality-driven* (ou seja, orientado pela qualidade) torna-se uma segunda solução para o *design* de dispositivos que necessitam satisfazer as exigências de *real-time*, baixo consumo de energia, entre outros. Dessa forma, especifica-se qual a nova qualidade do sistema a ser requeria e como esta meta é obtida. De forma a facilitar a compreensão, Jóźwiak define qualidade de uma solução sistêmica proposto como o total de sua eficácia e eficiência na resolução do problema real. Eficácia entende-se como o grau em que uma solução atinge seus objetivos e a eficiência o grau em que uma solução usa recursos para realizar seus objetivos e juntas determinam o grau de excelência. Elas são expressas em termos de parâmetros mensuráveis, o que é necessário para implementar o design *quality-driven*. Entretanto, é descrito ao final que, enquanto *designsers* aprenderam bastante na construção de plataformas de *hardware* heterogêneos altamente paralelos, os métodos e ferramentas automatizadas para a sua programação e o paralelismo do algoritmo, bem como o *codesign* coerente da arquitetura *hardware* e *software* ainda são atrasados perante à tecnologia.

Por fim, verificando no espectro de *wearables*, é possível ver vários trabalhos (Abdelhedi et al. 2016, Ahola et al. 2007, Lee 2015, Narumi 2016, Plessl et al. 2003) que relacionam FPGAs com aplicação *wearable*. Entretanto, nenhum trabalho menciona análise metodológica do problema de particionamento *hardware* e *software* para *design* de sistemas computacionais *wearables* e seus requisitos de funcionamento, tema tratado nesta pesquisa.

3.1 Comparativo entre os Trabalhos

Fazendo uma relação entre a pesquisa proposta com os trabalhos relacionados, é possível visualizar pela Tabela 3.1 que, a pesquisa tem como proposta uma metodologia para o particionamento *hardware* e *software* de dispositivos *wearables* utilizando recursos em *hardware* reconfigurável a fim de buscar otimizações em sua performance utilizando aceleradores em *hardware*.

A tabela exibe todos os trabalhos apresentados de forma a pontuar as ferramentas e os propósitos. Como existe muitos trabalhos que abordam sistemas embutidos, foi adicionado uma especificação na qual indica se além do foco em sistemas embarcados, o trabalho proposto também possui uma análise em sistemas *wearables*. Há trabalhos que trabalham com o particionamento, mas sem foco em *wearables*, outro que não utilizam de plataformas FPGA como meio para a obtenção de seus objetivo e outros que trabalham com *wearables* e FPGA mas o objetivo do estudo do particionamento. Dessa forma, a pesquisa tende a realizar uma análise utilizando todos estes conceitos, sendo eles, o particionamento para *wearables* utilizando plataformas FPGAs.

Tabela 3.1: Comparativo visual entre os principais itens tratado nos trabalhos relacionados e o trabalho aqui apresentado.

Trabalhos Relacionados	**Particio-namento**	**Embarcado/ *Wearable***	**FPGA**	**Observações Adicionais**
Edwards and Forrest (1994)	✓	✓ / ✗	✓	Proposta de uma metodologia nova
Stitt et al. (2003)	✓	✓ / ✗	✓	Particionamento Dinâmico
Jigang and Thambipillai (2004), Mann et al. (2007), Strachacki (2008)	✓	✗ / ✗	✗	*Branch-and-bound*
Madsen et al. (1997), Wu and Srikanthan (2006)	✓	✗ / ✗	✗	Prog. dinâmica
Niemann and Marwedel (1997)	✓	✗ / ✗	✗	Prog. linear inteira
Nematbakhsh et al. (n.d.)	✓	✗ / ✗	✓	*Footprint* FPGA vs. *speedup software*
Yan et al. (2017)	✓	✗ / ✗	✗	Otimização *position disturbed particle swarm*
Wang et al. (2016)	✓	✓ / ✗	✓	Modelagem de incerteza para o particionamento
Choi (2016)	✓	✗ / ✗	✓	*Framework* LegUp
Mei et al. (2000)	✓	✓ / ✗	✓	Processador (Particionamento e escalonamento para sistemas embutidos dinamicamente reconfiguráveis com GA)
Arató et al. (2003)	✓	✓ / ✗	✗	Particionamento para RTOS e custo restringido, utiliza-se de prog. linear inteira
Mann et al. (2007)	✓	✓ / ✗	✗	*Branch-and-bound*
Hassine et al. (2017)	✓	✓ / ✗	✗	Otimizações em *cores*. Particionamento que visa o tempo de execução vs. gasto energético
Trindade and Cordeiro (2016)	✓	✓ / ✗	✗	Utiliza algoritmo genético
Jóźwiak (2017)	✓	✓ / ✗	✗	*Survey* sobre particionamentos em sistemas embutidos. Classifica-os em dois paradigmas: *life-inspired* *quality-driven*
Abdelhedi et al. (2016), Ahola et al. (2007), Lee (2015), Narumi (2016), Plessl et al. (2003)	✗	✓ / ✓	✓	Não realizam análise metodológica sobre o problema de particionamento
Trabalho Proposto	✓	✓ / ✓	✓	**Metodologia para *wearable* com foco em aumento de *speedup* e controle energético**

Capítulo 4

Design de Sistemas Wearable

A seguir, serão descritos alguns tópicos seguindo conceitos estabelecidos pelo livro de Sass and Schmidt (2010) e utilizado por vários trabalhos como Arató et al. (2003), Arato et al. (2005), Hassine et al. (2017), Mann et al. (2007).

Os tópicos a seguir são a Definição de *Design* de Referência de *Software* (Seção 4.1) bem como o Ganho de Performance (Seção 4.2) em tais sistemas, o Particionamento *Hardware* e *Software* para Sistemas *Wearables* (Seção 4.3) e a Proposta de Procedimento Analítico (Seção 4.4).

4.1 Introdução ao Design Referencial de Software Utilizando Grafo de Controle de Fluxo

É possível descrever sistemas livre de especificações formais de *software* ou *hardware* por meio de descrições de protótipos simples. Estes, conhecidos como *design* referencial de *software*. Como será visualizado a seguir, sua vantagem mais notável é a generalização de uma especificação completa, eliminando quaisquer tipo de incertezas sobre o comportamento do sistema ao realizar uma análise sobre, sendo este podendo ser representado por diversas formas.

Como o algoritmo a ser analisado e particionado também pode ser representado em forma de grafo de tarefas/rotinas, pode-se então associar o *design* da sub-rotina também com o uso da teoria de grafos (Mann et al. 2007), em especial o Gráfico de Controle de

Fluxo (GCF). Ele é definido por

$$C = (B, F) \tag{4.1}$$

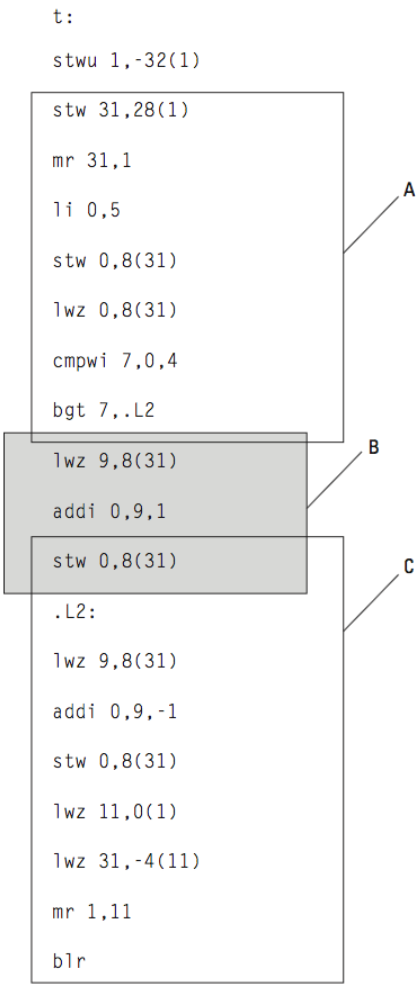

Figura 4.1: Identificação de blocos básicos em um código *assembly*.

onde B são vértices que representam os blocos básicos[1] e F são arestas que indicam a todas as possibilidades de caminhos entre os blocos.

Utilizando o exemplo da Figura 4.1, o primeiro grupo $\mathcal{A}$ é um bloco não básico porque não é maximal, ou seja, a primeira instrução `store word with update` deveria estar incluída ao grupo para conter o número máximo de instruções possuindo apenas um ponto de entrada e saída. Grupo $\mathcal{B}$ é um bloco básico e o grupo $\mathcal{C}$ não se define como bloco básico pois existe duas entradas para o bloco, sendo elas na instrução `store word` e também pelo `branching` direcionado para `L2`.

Dessa forma, fazendo uma relação entre o processo de gerar um grafo de controle de fluxo a partir de um código em alto nível, a Figura 4.2 exibe um pequeno código demonstrativo na qual o processo ocorrerá. A partir do código em alto nível (Figura 4.2 *a)*) é identificado os blocos básicos de acordo com o compilador[2] utilizado. Neste exemplo, utilizou-se de um compilador para `PowerPC`[3] onde os blocos básicos são identificados pela Figura 4.2 *b)*. Por fim o grafo de controle de fluxo resultante deste processo, representado pela Figura 4.2 *c)*.

A decomposição de um *design* referencial de *software* pode gerar dois componentes:

[1] De modo geral é um trecho de código sequencial maximal, onde só existe um ponto de entrada e um ponto de saída.

[2] Deve-se atentar que, só é possível identificar blocos básicos em um arquivo em linguagem de programação *C* desde que se saiba qual compilador foi utilizado para emitir o código *assembly*.

[3] Arquitetura que utiliza RISC como arquitetura do conjunto de instruções.

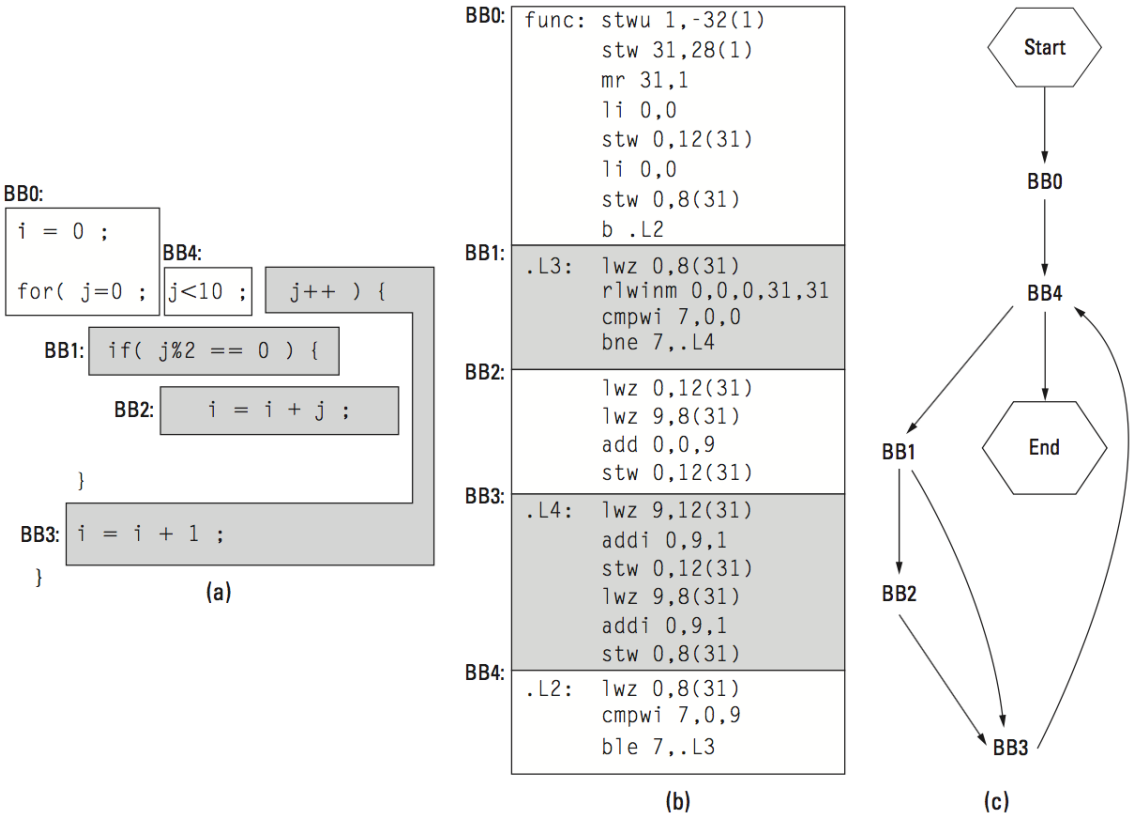

Figura 4.2: Identificação de blocos básicos e a representação por meio de um grafo não atrelado à uma especificação *hardware* e *software*.

uma porção a ser realizada em *hardware* e; outra executada em *software*. Essa decisão de divisão é chamada de Problema de Particionamento. Segundo Sass and Schmidt, para sistemas em FPGA, o particionamento é um sub-problema de um problema mais geral no âmbito de *codesign*, onde refere-se ao *design* cooperativo entre engenheiros de *hardware* e *software* para um desenvolvimento mais eficiente.

4.1.1 Grafo de Chamada

Modelado uma sub-rotina de um *design* referencial de *software* utilizando o grafo de controle de fluxo, definido na Seção 4.1, agora será descrito uma nova notação, chamada de Grafo de Chamada (GC) utilizado para o entendimento da partição. Consiste num conjunto de GCFs, um por sub-rotinas, ou seja,

$$\mathcal{C} = C_0, C_1, \ldots C_{n-1} \tag{4.2}$$

onde $C_i = (B_i, F_i)$ representa o grafo de controle de fluxo de uma sub-rotina i, como mostrado na Equação 4.1. Sendo assim, o grafo estático de chamada da aplicação é escrito por

$$\mathcal{A} = (\mathcal{C}, \mathcal{L}) \tag{4.3}$$

onde $\mathcal{A}$ representa uma aplicação específica e $\mathcal{L} \subseteq \mathcal{C} \times \mathcal{C}$ é um subconjunto do plano cartesiano dos GCF. Duas sub-rotinas são relacionadas se podem ser determinadas que, no tempo de compilação, a sub-rotina i tem potencial de invocar a sub-rotina j, ou seja, $(C_i, C_j) \in \mathcal{L}$.

É assumido que os blocos básicos de cada sub-rotina são disjuntos, ou seja, cada bloco básico em uma aplicação pertence a exatamente um GCF. Além do mais, é assumido também que um nó raiz para o GC é implícito, ou seja, uma sub-rotina é designada a iniciar a execução. Nem todos os executáveis podem ser expressados nesse modelo. Por exemplo, o manuseio de sinais e interrupções não são representadas e assim, não é possível determinar todos vértices F_i em uma dada sub-rotina C_i de um GCF antes da execução.

Um equívoco comum é de que uma definição formal de particionamento só aplica à separação de aplicação componentes de *hardware* e *software*, ou seja, a partição contém exatamente dois conjuntos. Todavia, para fazer o problema mais tratável, é comum agrupar primeiramente operandos em recursos, ou seja, uma partição com um grande número de subconjuntos, e então mapeia esses recursos tanto em *hardware* quanto *software*. Assumindo que esses recursos atuam razoavelmente bem *clustered*, então a decomposição de uma aplicação em componentes de *hardware* e *software* pode ser dirigida por comparações de ganho de performance desse recurso contra outro situado no outro conjunto.

Definidos os conceitos prévios, será definido agora, formalmente, o conceito de uma partição.

Uma partição $\mathcal{S} = \{S_0, S_1, \dots\}$ de um conjunto universal U é um conjunto de subconjuntos de U sendo que

$$\bigcup_{S \in \mathcal{S}} S = U \tag{4.4}$$

$$\forall S, S' \in \mathcal{S} | S \cap S' = \emptyset \tag{4.5}$$

e assim

$$\forall S \in \mathcal{S} \cdot S \neq \emptyset \tag{4.6}$$

Descrevendo textualmente:

Equação 4.4: cada elemento de U é um membro de, pelo menos, um subconjunto $S \in \mathcal{S}$;

Equações 4.5 e 4.6: os subconjuntos $S \in \mathcal{S}$ são emparelhados disjuntos e não vazio. Em outras palavras, cada elemento do nosso universo U termina exatamente em um dos subconjuntos de $\mathcal{S}$ e nenhum dos subconjuntos são vazios.

Por exemplo, considerando um sistema hipotético *wearable* na qual possui o universo $U = \{a, e, i, o, u, y\}$ de recursos de processamento. Dessa forma, uma partição desse problema $\mathcal{X}_a$ de U pode ser representado por

$$\mathcal{X}_a = \{\{a, e, i, o, u\}, \{y\}\} \tag{4.7}$$

e supondo que o conjunto possa ser representando por cada unidade, uma outra forma de representação pode ser

$$\mathcal{X}_b = \{\{a\}, \{e\}, \{i\}, \{o\}, \{u\}, \{y\}\} \tag{4.8}$$
$$\mathcal{X}_c = \{\{a\}, \{e\}, \{i\}, \{o\}\} \tag{4.9}$$
$$\mathcal{X}_d = \{\{a, e, i\}, \{i, o, u, y\}, \{\}\} \tag{4.10}$$

Sobre tais conceitos, a Partição 4.9 viola a Equação 4.4 e a Partição 4.10 viola as Equações 4.5 e 4.6, pois a primeira omite um item y e a segunda possui um conjunto vazio além de redundância. Assim, ao utilizar o particionamento para este exemplo, a Figura 4.3 ilustra o $\mathcal{X}_a$ (Equação 4.8) graficamente, segundo a Partição 4.7 como seria uma divisão entre *hardware* e *software* dos recursos listados.

Aprendido os conceitos, é possível aplicar o formalismo à $\mathcal{A}$. Se assumirmos que nosso universo é o conjunto de todos os blocos básicos B de todas as sub-rotinas de um

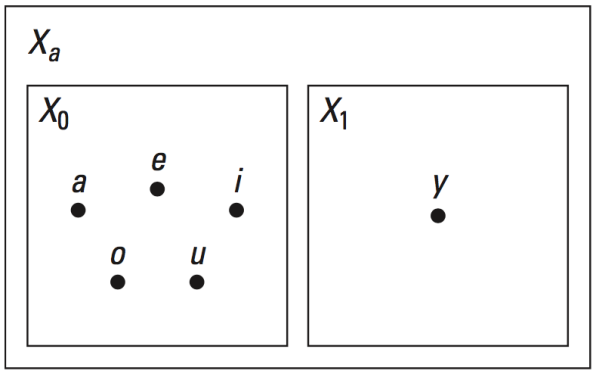

Figura 4.3: Figura representativa do mapeamento da aplicação.

dispositivo *wearable*, então U é as partições de sub-rotinas

$$U = \bigcup_{C \in \mathcal{C}} B(C) \tag{4.11}$$

e chamaremos de partição natural da aplicação, onde

$$\mathcal{S} = \left\{ \underbrace{\{b_0, b_1, \ldots, b_i\}}_{\text{sub-rotina } C_0}, \underbrace{\{b_i, b_{i+1}, \ldots, \}}_{\text{sub-rotina } C_1}, \ldots \underbrace{\{b_j, b_{j+1}, \ldots, \}}_{\text{sub-rotina } C_{n-1}} \right\} \tag{4.12}$$

O propósito é reorganizar a partição de blocos básicos e então mapear cada subconjunto de ambos os *hardware* e *software*. Dessa forma, estamos livres para criar e remover subconjuntos não vazios, e mover blocos básicos ao redor até termos uma nova partição e assim termos um novo resultado $\mathcal{A}' = (\mathcal{C}', \mathcal{L}')$, inferido a partir da reorganização da partição $\mathcal{X}'$. O segundo passo é mapear cada subconjunto de $\mathcal{X}$ para ambos *hardware* e *software* como é exibido abaixo

$$\mathcal{X}' = \left\{ \underbrace{\underbrace{\{b_j, b_{j+1}, \ldots, \}}_{\text{sub-rotina } C_k} \underbrace{\{b_k, b_{k+1}, \ldots, \}}_{\text{sub-rotina } C_l} \ldots}_{software} \underbrace{\underbrace{\{b_0, b_1, \ldots, b_i\}}_{\text{sub-rotina } C_r} \underbrace{\{b_i, b_{i+1}, \ldots, \}}_{\text{sub-rotina } C_s} \ldots}_{hardware} \right\} \tag{4.13}$$

4.2 Ganho de Performance

Para explicar como performance pode ser utilizada para guiar o particionamento, será descrita uma métrica simples chamada taxa de execução[4]. É parcialmente motivada pelo fato de que: *a)* o ganho de performance é relativamente fácil de ser mensurado e *b)* por causa de que, de todas as métricas comumente utilizadas, *speedup* é frequentemente a mais importante. Diferente do mundo *software* onde se tem análise de ordem de complexidade, em *hardware* não possui-se um guia geral para comparação. O ganho de performance para aplicações em geral pode estar na acumulação de pequenos ganhos que deveriam ser perdido numa aplicação direta na teoria de complexidade.

Assim, para o *software*, será usado a informação de *profile* (Seção 2.3) para coletar o tempo total de execução, bem com uma fração do tempo gasto em cada sub-rotina. O produto disso é a aproximação entre o tempo necessário para executar uma porção de aplicação em *software* e usar isso como o tempo que se espera que tomará em futuras execuções. É considerado uma aproximação pois é dependente dos conjuntos de dados de entrada para muitas aplicações além da existência de erros que podem impactar a performance. Será utilizado $s(i)$ para representar o tempo de execução esperado para uma invocação de uma sub-rotina i, ou seja, bloco básico.

Precisa-se também aproximar o tempo que uma implementação equivalente em *hardware* que iria tomar. No caso dos blocos básicos implementados, isso é frequentemente mais preciso. Como não possui-se um controle de fluxo, uma ferramenta auxiliar à síntese poderá dar uma aproximação de acurácia de propagação de tempo. Ou, se o recurso é *pipelined*, o número de estágios é mais precisamente conhecido. Caso o recurso inclua controle de fluxo mas não contenha nenhum *loop*, pode-se considerar o caminho mais longo como uma estimativa conservativa.

Recursos com um número variável de iterações através de um *loop* apresentam o maior obstáculo para encontrar um tempo de *hardware* aproximado. Nesse caso, implementação e *profiling* com recurso em *hardware* pode ser a única solução. Independente, assume-se que uma aproximação apropriada $h(i)$ para o existente tempo de execução em *hardware*.

Por fim, a 'interfaceação' entre *hardware* e *software* requer tempo e este custo também

[4]Taxa de execução é a velocidade na qual um sistema computacional completa uma aplicação, e em um sistema de plataforma FPGA olhamos também para o *hardware* para melhorar sua taxa de execução.

precisa ser contabilizado. Pode-se aproximar deste custo pela aproximação do montante total do estado que necessita ser transferido ou o custo de configuração e latência. Em ambos os caso, são representados por $m(i)$ para recursos $i \in \mathcal{H}$, sendo $\mathcal{H}$ o conjunto de recursos do *hardware*.

O ganho, ao comparar uma solução *hardware* e *software* contra uma solução puramente *software*, é tipicamente mensurado como *speedup*. Utilizamos γ para sua representação e isso nos permitirá comparar recursos diferentes contra outros para determinar melhores particionamentos. Dessa forma, qualquer subconjunto de blocos básicos que não produzem um ganho de performance, podem ser excluídos de consideração. Em outras palavras, somente subconjuntos de blocos básicos para qual $\gamma > 1.0$ são considerados recursos candidatos.

Então quando considerado se um conjunto particular de blocos básicos deveriam ser mapeados ao *hardware* ou *software*, estamos interessados em seu ganho em *speedup*, ou seja

$$\gamma = \frac{hardware\ speed}{software\ speed} = \frac{\frac{1}{hardware\ time}}{\frac{1}{software\ time}} = \frac{software\ time}{hardware\ time} \tag{4.14}$$

Mais especificamente, interessa-se no ganho de performance individual de cada recurso e assim, definindo $\gamma(i), i \in \mathcal{C}$

$$\gamma(i) = \frac{s(i)}{h(i) + m(i)} \tag{4.15}$$

onde $h(i)$ e $s(i)$ são o tempo de execução de uma implementação de um recurso i em *hardware* e *software* e a função $m(i)$ é o tempo que se leva para sincronização, ou seja, o tempo que leva para guiar um dado entre o processador e o item reconfigurável.

A velocidade da aplicação é dependente dos ganhos de performance do recurso e o quão frequentemente ele é utilizado no *design* referencial de *software*. Pode-se ter essa fração do tempo gerado de um recurso particular $p(i)$ a partir de informações de *profile* e dessa forma o *speedup* da aplicação no geral será

$$\Gamma = \left[(1 - p(i)) + \frac{p(i)}{\gamma(i)} \right]^{-1} \tag{4.16}$$

A inversão representa que estamos movendo entre taxa de execução e tempo de execução para manter o sentido de ganho de performance.

A partir dessa equação, podemos observar que aumentando a velocidade do *hardware* de um único recurso tem-se menos e menos impacto na performance da aplicação a medida que sua frequência decresce.

Assim, para aumentar a performance sistêmica de uma aplicação no geral, também deve-se aumentar o sistema com múltiplos recursos que aumentará a performance de componentes individualmente assim como aumentando a fração agregada de tempo gasto em *hardware*. Para computar o *speedup* de múltiplos recursos em *hardware*, ou seja, avaliar o ganho sistêmico de um conjunto de recursos $\mathbb{D}$ onde cada membro do conjunto contribui à performance do sistema baseado na fração do tempo gasto em cada característica. Para estimar a performance desta partição, podemos adicionar recursos e rearranjar os termos para ter um ganho de performance almejado no geral, assim para o cálculo de performance dos recursos, utiliza-se da Equação 4.17.

$$\Gamma(\mathbb{D}) = \left[1 + \sum_{i \in \mathbb{D}} \left(\frac{p(i)}{\gamma(i)} - p(i)\right)\right]^{-1} \tag{4.17}$$

4.2.1 A Considerações de Recursos

Seguindo a Equação 4.17, uma tentativa de consideração de recursos seria a adição de recursos na abordagem $\sum_i p_i$, ou seja, implementar tudo em *hardware* para maximizar a performance, ignorando todos os custos de desenvolvimento e recursos limitados. Num FPGA, há um número finito de recursos disponíveis para implementação de circuitos em *hardware*e como tais recursos são limitados, a maioria das aplicações realísticas iriam exceder esse limite disponível. Um meio de aproximação de recursos é contar o número de células lógicas requeridas para cada recurso. Um chip que terá um valor escalar r_{FPGA}, representará o total de números de células lógicas disponíveis. Então $r(i)$ pode ser usado para representar a quantidade de células lógicas requeridas por cada recurso i. Fazendo uma simples relação, tem-se que $\sum_{i \in \mathbb{D}} r(i) < r_{FPGA}$ restringe quão largo $\mathbb{D}$ pode crescer.

Sabendo que dispositivos modernos são heterogêneos, uma típica plataforma FPGA tem múltiplos tipos de recursos além de células lógicas como memória, blocos DSP, etc.,

podendo ser representados por um vetor de recursos

$$\vec{r}_{FPGA} = \begin{pmatrix} r_{Logic\ Cells} \\ r_{Memory} \\ r_{DSP} \\ \vdots \\ r_{n-1} \end{pmatrix} \tag{4.18}$$

e com isso,

$$\sum_{i \in \mathbb{D}} \vec{r}(i) < \vec{r}_{FPGA} \tag{4.19}$$

onde $\mathbb{D}$ é o conjunto de recurso incluídos no *design*.

4.3 O Particionamento de Hardware e Software para Sistemas Wearable

De início, será considerado como aplicação *wearable* um conjunto de instruções organizadas, e como visto na Seção 4.1.1, esta também representada por uma coleção de grafos de controle de fluxo, ou seja, grafo de chamada, especificando a sua ordem de execução. A partir deste, será feito análises a fim da procura de um particionamento que atenda aos requisitos de dispositivos *wearables*, como alto poder de processamento sem o *trade-off* de energia, além de miniaturização, confiabilidade e outros.

Após esclarecidos algumas definições prévias (Seção 4.3.1), será apresentado o problema na Seção 4.3.2.

4.3.1 Definições Prévias

Recurso: grupo conectado de instruções de uma aplicação de *design* referencial de *software* 'adequado' para uma implementação em *hardware*.

O recurso pode variar de um pequeno conjunto de instruções até um modulo de *software* completo consistente de múltiplas sub-rotinas. Como o tamanho dos recursos afetam na performance, a decisão de implementação em *hardware* depende

da sua melhoria no sistema por inteiro e mensura-se os recursos utilizados com relação a outros recursos candidatos.

Se determinado que o recurso é vantajoso, então os recursos de implementação em *hardware* aumentam a arquitetura de *hardware*;

Implementação em *hardware*: recurso adicional de uma específica aplicação;

Adequado: descrevendo de forma mais geral no âmbito de sistemas *wearable*, é a definição da situação na qual o projetista do sistema antecipa a percepção de vantagens na implementação em *hardware*.

4.3.2 Declaração do Problema

Nesta seção serão apresentadas as declarações matemáticas do problema de agrupamento de instruções em recursos e seus mapeamentos em *hardware* ou *software*, ou seja, o particionamento *hardware* e *software*.

No particionamento, muitos problemas práticos impactam diretamente na performance do sistema. Nem todos os problemas podem ser incorporados num modelo analítico Wang et al. (2016), e por isso, só podemos esperar que as soluções matemáticas produzam uma uma resposta aproximada ao problema de particionamento ao utilizar a declaração formal.

Muitas das entradas do modelo são estimadas ou aproximações no qual futuramente degrada a fidelidade de resultados. Dessa forma, é mais eficiente usar uma combinação de técnicas *ad hoc* e matemáticas para encontrar uma solução ótima ou aproximada do que simplesmente confiar numa intuição.

Já descrito as ferramentas matemáticas necessárias para descrever o problema fundamental do particionamento no Capítulo 2, pode-se então descrever formalmente o problema em termos de variáveis. A ideia básica consiste em encontrar um particionamento para todos os blocos básicos de uma aplicação e então separá-los em *hardware* e *software*. Formalmente, procura-se por uma partição $\mathcal{P}$ de todos os blocos básicos U de uma aplicação (Equação 4.11). Definida a partição e o universo, tem-se então um subconjunto $\mathbb{C} \mid \mathbb{C} \subseteq U$, onde $C \in \mathcal{C}$ é um vértice de um grafo de *design* referencial de *software* $\mathcal{A} = (\mathcal{C}, \mathcal{L})$ (oriundo da Equação 4.3). O conjunto $\mathbb{C}$, chamado conjunto de candidatos, contém todos os recursos arquiteturais potenciais, ou seja, o subconjunto de U que é esperado para melhorar a performance do sistema se implementado em um *hard-*

ware reconfigurável. Devido ao limite de recursos, deve-se refinar para o subconjunto $\mathbb{D} \subseteq \mathbb{C}$ que maximiza nosso métrica de performance. Assim

$$\begin{aligned} &\max && \Gamma(\mathbb{D}) \\ &\textit{subject to} && \textstyle\sum_{i \in \mathbb{D}} \vec{r}(i) \;<\; \vec{r}_{FPGA} \end{aligned} \tag{4.20}$$

Descrevendo algoritmicamente, uma abordagem seria encontrar todas as partições de U, sintetizando e *profiling* cada partição, e então, quantitativamente avaliar cada Γ. Entretanto, este é um problema linear inteiro devido à natureza de alocação e utilização dos recurso físicos do FPGA.

A seguir, será descrito como é feito uma abordagem para tal, seguindo estudos de Arató et al. (2003), Wang et al. (2016).

4.4 Proposta de Procedimento Analítico

Como conclusão deste capítulo, será formulada uma proposta de metodologia analítica baseada nos conceitos propostos pela literatura, com o foco em componentes procedurais integrados à sistemas *wearables*. Tal metodologia, apresentada pelo Algoritmo 4.1, será considerada apenas como uma orientação para os passos a serem realizados, sendo então, uma proposta representativa do processo a ser realizado para a análise e decisão de particionamento.

Apresentado o procedimento metodológico, a seguir será listada cada função pertencente à metodologia acima, descrevendo suas funções e seu propósito no trabalho para a obtenção dos objetivos. Para um melhor compreendimento, serão divididos em quatro seções, sendo elas: visualização do projeto, síntese, análise de síntese, particionamento.

1. Procedimentos para visualização geral do projeto. Por meio dos dados do *profile* e a construção dos grafos, é possível ter uma visão geral, verificando se o sistema possui possíveis seções de processamentos críticos para a realização de particionamento *hardware* e *software*. As funções presentes são:

    - `profile()`: procedimento referente à Seção 2.3. Realiza-se uma análise de tempo gasto em cada procedimento do projeto, apresentando-a ao final de sua execução. Com os resultados, é possível ver locais onde existe um tempo

Algoritmo 4.1: Metodologia para avaliação de *wearables*.

```
   Input: a project description.
   Output: a partition solved.
 1 begin
 2     profile();
       // extration project analyses
 3     makes_graph(Flow Control Graph);
 4     makes_graph(Call Graph);
 5     if exist partitions that can be synthesized then
 6         foreach partition project:it ∈ partition_list do
 7             synthesizes(it);
 8             resources_used(it);                    // analysis after synth
 9             die_used(it);
10             energy_spent(it);
11             performance_analysis(it);
12         end
13         integer_linear_solve(partition_list);     // analyse quantitatively
            each Γ
14     end
15 end
```

maior de processamento gasto ou de recursos utilizados indicando uma verificação detalhada sobre este a fim de candidatá-lo para uma implementação em *hardware*;

- `makes_graph(`*type*`):` constrói-se grafos relativos ao projeto, sendo estes de acordo com o tipo especificado no parâmetro. A construção do grafo segue como descrito na Seção 4.1 na qual procura-se por seções de códigos compondo-os em blocos, compreendendo o fluxo do algoritmo e consecutivamente os procedimentos de chamadas. O parâmetro simboliza a especificação de qual tipo de grafo será construído, sendo ele um grafo de controle de fluxo (Seção 4.1) ou grafo de chamada (Seção 4.1.1), que como já explicado, necessita do anterior para sua construção;

2. `synthesizes():` caso exista uma possibilidade de particionamento, análise obtida pelo *profile*, visualização do grafo e obtenção de uma lista de candidatos, realiza-se então o procedimento de geração de HDL para o desenvolvimento de aceleradores em *hardware* reconfigurável e sua análise de performance e gastos tanto de energia quanto de recursos;

3. Após a execução do processo de síntese realizado junto com a ferramenta sintetizadora assistida pelo computador, é possível obter vários dados analíticos de alocação de recursos como quantidade de elementos lógicos utilizados, pinos virtuais e físicos, quantidade de bits de memória, além de vários outros. E com a sua sintetização na plataforma, também é possível obter a avaliação de consumo energético, *profile* e performance. Estes são representados pelos métodos abaixo na qual, aplica-se a cada componente sintetizado separadamente.

    - `resources_used()`: quais e a quantidade de recursos alocados para utilização no projeto de geração de aceleradores. A alocação de muitos recursos implica num gasto maior de energia criando um *trade-off* a ser analisado;
    - `die_used()`: tamanho do *die* utilizado para o projeto do acelerador. Sistemas *wearable* possuem como requisito a sua miniaturização, impedindo que seu uso limite a capacidade de locomoção, usabilidade ou conforto do usuário, por exemplo;
    - `energy_spent()`: valores energéticos do uso do recurso implementado, incluindo os módulos pertencentes a ele como memórias, DSPs e quaisquer outros que estejam integrados à plataforma;
    - `performance_analysis()`: comparação das implementações em *hardware* sobre os recursos em nível de *software*, ou seja análise de performance.

4. `integer_linear_solver()`: após realizado todas as análises individuais acima, inicia-se o processo de procura de uma solução de particionamento que obtenha bons resultados nos requisitos de um sistema *wearable*. Dessa forma, é realizado uma avaliação por meio de um *solver* linear inteiro segundo os dados obtidos. Com o *solver*, é possível realizar uma busca a procura de um particionamento que atenda a quantidade máxima de restrições exigidas por um *wearable*.

Assim, o trabalho consiste numa metodologia na qual aborda o particionamento para dispositivos *wearables*, partindo de uma análise de execução do *software*, candidatando alguns procedimentos para sua sintetização e assim a avaliação destes segundo os recursos utilizados para a completude de suas tarefas o que chamamos de particionamento, consistindo da procura de um conjunto que traga melhor desempenho no seu uso. Tudo isso, utilizando recursos disponibilizados pela plataforma em *hardware*.

Capítulo 5

Metodologia Proposta

Apresentado o problema de particionamento, sua importância, será apresentado uma abordagem para o projeto de sistemas computacionais *wearables* e aplicações deste em sistemas hipotéticos.

Como foi possível perceber no Capítulo Trabalhos Relacionados (Capítulo 3), existem vários trabalhos em sistemas computacionais embarcados, entretanto, nenhum que abordasse o tema para *wearables*. Partindo deste pressuposto, esta pesquisa propõe o estudo de dispositivos *wearables* e a aplicação de técnicas de particionamento a fim de obter um desempenho melhor para estes.

A metodologia proposta baseia-se em procedimentos que utilizam as técnicas descritas neste documento. No Capítulo 4 foi descrito os principais métodos utilizados neste trabalho, e ao final foi apresentado um procedimento na qual a metodologia usará como base. O método consiste unicamente na organização dos conceitos discutidos a fim de obter uma análise sistemática do projeto e assim, realizar o particionamento a procura de melhorias. A diferença entre a análise metodológica de um exemplo hipotético e de um dispositivo real é que com o real, todas as principais funções do dispositivo são consideras e assim, a quantidade de procedimentos a serem considerados para o particionamento *hardware* e *software* seria maior, criando um leque maior de possibilidade para análise. Assim, para demonstração da metodologia, será realizada sua aplicação para a análise de dois dispositivos *wearables* hipotéticos descritos nas Seções 5.1 e 5.2 a seguir.

5.1 Wearable Hipotético 1: Mão Biônica

Apresentação Como já é de conhecimento, o ser humano, bem como todos os outros animais, usa de seus músculos corporais para as tarefas cotidianas como andar, comer, etc. Com o estímulo ordenado dos músculos, é possível realizar atuações até mais complexas como segurar um objeto pelas mãos ou mesmo, ter controle sobre um automóvel ou realizar uma performance artística/musical. Assim, uma situação de uso de *wearable* é, ao invés do usuário utilizar de suas próprias mãos para realizar alguma atividade como a de segurar um objeto, com a adição de um sistema *wearable*, seria possível intermediar a atuação final (tomar algo pelas mão), com a leitura da tensão elétrica muscular do usuário ao longo do tempo, a fim de substituir a atuação de sua mão por meio de atuadores mecânicos artificiais, como mostra a Figura 5.1.

Figura 5.1: Demonstração de uma mão biônica. Fonte: `https://goo.gl/96PURB` Acesso: 30/07/2017.

Descrevendo-o de uma forma mais técnica, o *wearable* deve ser capaz de realizar leituras eletromiográficas[1], interpretá-las de acordo com um pretexto e realizar a atuação. Um pseudo-algoritmo é apresentado em Algoritmo 5.1. Pode-se supor que o dispositivo foi inteiramente construído em uma solução em *software* com a justificativa de projeto de uma construção mais simples. Assim sendo, o particionamento *hardware* e *software* entra com o papel de analisar o projeto a fim de buscar por combinações de particionamento que apresentem melhorias em performance ou no gasto energético ao estado atual. A análise é importante mesmo para projetos que já possuam aceleradores em *hardware* em

[1]Técnica na qual realiza o monitoramento de atividades elétricas das células musculares.

seu projeto pois a verificação atua como uma forma de validação e confirmação de sua performance como uma solução aproximada do ótimo.

Algoritmo 5.1: *Wearable* 1 - Procedimentos realizados pelo dispositivo *wearable* denominado Mão Biônica.

Input: sensors reads.
Output: acting.

```
1  begin
       // statements
2      sensor_i ;                                  // i information's source
3      matrix_data ;                          // data's matrix before handling
4      matrix_data_preprocessed ;              // data's matrix after handling
5      buffer_state ;                        // last state applied and current
6      state_before ;                        // generated state to be compared
7      while wearable mode on do
8          matrix_data ← reads_sensors(sensor_i);
9          matrix_data_preprocessed ← pre_process(matrix_data);
           // process the datas
10         if buffer_state ← processes(matrix_data_preprocessed) ≠ state_before
            then
               // operates
11             state_before ← buffer_state;
12             operates(state_before);
13             backup();                          // saves the data persistently
14         end
15     end
16 end
```

Suponha então que deseja-se realizar uma avaliação de particionamento *hardware* e *software* de um dispositivo *wearable* que atua por meio de tais sinais eletromiográficos.

Análise Usando Profile O primeiro procedimento a ser realizado para a obtenção de informações de projeto e execução do programa é com a execução de um *software* que faça o seu *profile*. Ao realizar esta tarefa, obtêm-se resultados de tempo de execução dos principais procedimentos do projeto, listados de forma ordenada por tempo gasto, como já descrito na Seção 2.3.

Feito esta análise, é construído o grafo de chamada de acordo com a especificação de funcionamento do projeto.

Grafo A partir do pseudo-algoritmo dado e da análise *profile*, é possível gerar o grafo de chamada, exibido pela Figura 5.2. Como descrito na Seção 4.1, é realizado uma análise no projeto verificando os códigos e funções, gerando o seu respectivo grafo de chamada.

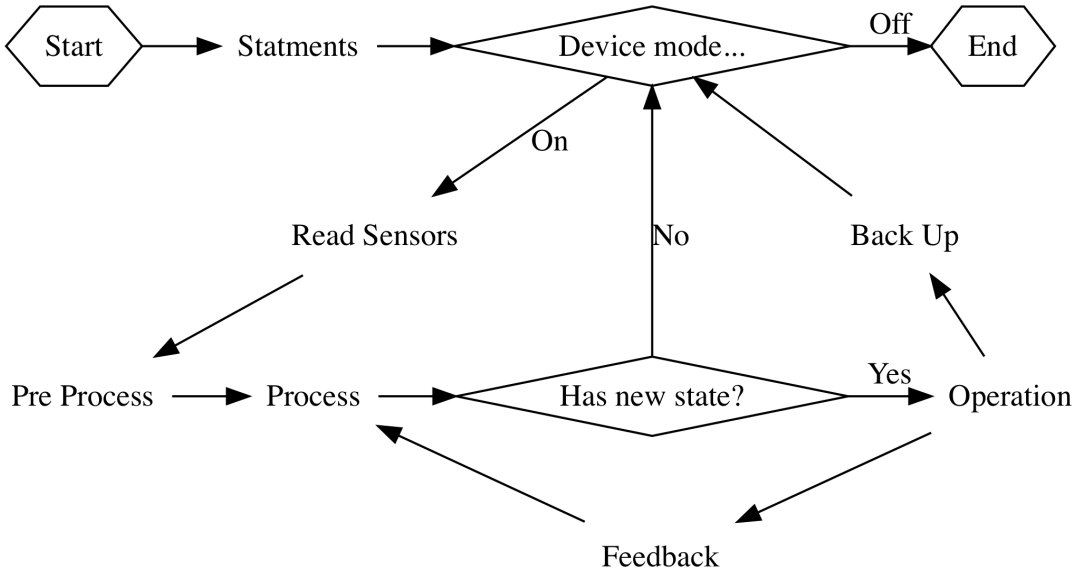

Figura 5.2: Exemplificação de um grafo de chamada para o exemplo.

O processo representado pelo grafo da Figura 5.2 inicia-se pelo estado de formato hexágono com etiqueta *start*. Ao iniciar o dispositivo, é realizado as inicializações de ambiente, definidas pelo estado seguinte *statments*, realizando todas as definições prévias para seu uso. Este processo é unicamente para inicialização do equipamento, redefinindo todos os estados da última operação ou de um estado específico de inicialização. Em seguida, tem-se o primeiro losango indicando uma verificação de estado. Neste, verifica-se se o dispositivo entrou-se em modo de desligamento, se sim, desligando-o de forma segura. Caso contrário, realiza leituras eletromiográficas, o pré-processamento delas e em seguida o processamento dos dados. Após processado, é verificado se existe um novo estado de acordo com os valores lidos. Se este novo estado é distinto do anterior, então o dispositivo deve realizar a operação de alteração da atuação para este novo, além do armazenamento deste em forma persistente para inicialização, caso estabelecido e o processo é reiniciado na verificação de de dispositivo ligado. Neste também é realizado a leitura de *feedback* da atuação retornando-o para o processamento para avaliar o erro e gerar uma resposta para tal.

O grafo gerado deste exemplo apresentado é uma solução simplista pelo fato de ser inteiramente hipotético. Com um exemplo real, seria possível ter um detalhamento maior no grafo, já que este teria informações suficientes sobre os procedimentos do dispositivo

para compreendimento.

Ao concluir a construção do grafo do sistema *wearable* e analisá-lo, consegue-se ter conhecimento de grupos procedurais importantes e assim, consequentemente separá-las em módulos. Foram denominados três grupo para a organização dos procedimentos, sendo eles Leitura, Processamento e Atuação, e serão descritos a seguir.

Módulo de Leitura: A leitura é realizada por meio de um sensor muscular na qual obtém sinais eletromiográficos, ou seja, um sensor de eletromiograma.

Não só como a leitura mas a interpretação dos sinais elétrico depende diretamente da natureza dos dados sensoriais obtidos pelo eletromiograma, um pré-processamento de tais dados é um fator está relacionado diretamente com a obtenção de informação do conjunto de dados. Então, supondo que o dispositivo *wearable* também necessite de um pré-processamento dos dados para filtragem e preparação para o processamento central de fato, este estaria incluso dentro do módulo de leitura, enviando-os em seguida para o processamento central.

O grupo será composto pelos procedimentos: `Statments`, `Read Sensors` e `Pre Process`.

Módulo de Processamento Central: Com os dados já trabalhados para serem utilizados de forma mais clara, o próximo passo a ser realizado é a interpretação destes de acordo com um pretexto pré-estabelecido pelo projetista e assim tomar uma decisão sobre este. Dessa forma, é realizado uma verificação de padrões do conjunto de dados recebido e a alteração de estado de atuação caso necessário.

Caso exista tal mudança, especificar qual e passar isso para o próximo módulo, a atuação. Caso contrário realiza novas leituras de dados, já que não obteve nenhuma alteração de estado de atuação.

Para o projeto ser mais responsivo, seria ideal obter também um *feedback* relativo à atuação, criando uma correlação mais precisa entre os dados lidos de sensores musculares com o estado do atuador e sua atuação. O *feedback* seria relacionado à completude da atuação, verificando se a atuação foi realmente executada como esperado e, caso negativo, encontrar o valor de erro para a sua tentativa de correção, processamento a ser realizado neste módulo.

Composto pelos procedimentos: `Process` e `feedback`.

Módulo de Atuação: Meio no qual realiza-se a atuação, segundo implicação oriunda do processamento central.

Neste, também seria realizado as leituras de *feedback*, enviando-os para o módulo de processamento central.

Composto pelas funções: `Operation` e `Back Up`.

Após feito o *profile* e o desenvolvimento do grafo, deve-se verificar quais procedimentos são candidatos a serem sintetizados para análise de performance. Procedimentos que realizam cálculos matemáticos repetitivos, ou que são executados ocasionalmente são excelentes candidatos já que pode-se utilizar os recursos de *hardware* para o processamento com maior vazão de dados ou o acionamento e desligamento do recursos quando não utilizado, respectivamente às situações apresentadas.

Análises de Síntese Construído o grafo e verificado os candidatos ao particionamento por meio do *profile*, realiza-se então a sintetização de todos os procedimentos candidatos bem como suas análises de uso de recursos e performance.

Todos os procedimentos do projeto apresentado são candidatos a serem sintetizados e analisados. São procedimentos que também necessitam de resposta imediata ao estímulo dado, aproximando ao máximo à uma atuação instantânea realizada pelos membros musculares naturais. Isso pois, todos os procedimentos podem ser executados sem a presença de um microcontrolador, usando recursos de uma máquina de estados para controle por exemplo.

Dessa forma, como são candidatos, serão sintetizados, analisados e comparados com os procedimentos realizados via *software* por meio de um microcontrolador.

Solver Obtida as análises de cada componente em nível de *software* e *hardware*, utiliza-se de *solvers* para encontrar uma solução boa.

No Capítulo 3, existem inúmeras aplicações que utilizam desde métodos exatos baseado em programação inteira linear até metaheurísticas. Dessa forma, após a aplicação do método *solver*, obteremos um particionamento solucionado pelo solver de acordo com as restrições de recursos impostas.

5.2 Wearable Hipotético 2: Capacete para Segurança de Ciclistas

Apresentação Outro exemplo de *wearable* e suas análises está em propor um capacete desenvolvido para a segurança de ciclistas. O hipotético propósito deste seria fornecer um auxílio para a segurança do usuário, junto com dados estatísticos de corrida com a bicicleta.

O equipamento seria utilizado para verificar situações na qual um veículo estaria se aproximando do usuário pela suas costas de forma a gerar uma colisão, local sem visibilidade constante pelo ciclista. Dessa forma, o capacete poderia avisar o ciclista para manter cuidado e tomar uma atitude segura enviando alguma informação para seu *smartphone* ou para algum dispositivo *wearable* acoplado ao seu corpo como uma pulseira. Ambos dispositivos poderiam passar a informação para o ciclista por meio de estímulos visuais ou físicos como luzes ou vibrações ou, pensando num equipamento mais tecnológico, um aviso em um óculos com realidade aumentada, por exemplo.

Como já dito, além da prevenção de acidentes, o equipamento poderia ainda suportar procedimentos que realizem a obtenção e envio de dados de atividades físicas do usuário como distância percorrida, tempo de atividade e na qual, após o processamento dos dados, a geração de informações como velocidade média, estimação de calorias queimadas, entre outras. Isso, enviando para um aplicativo no seu telefone móvel por meio de uma conexão sem fio.

O projeto, em termos técnicos, seria composto de sensores capazes de realizar a verificação de objetos volumosos e sua distância em relação ao ponto atual do ciclista. Ao avistar um objeto aproximando, por meio de uma conexão sem fio, o capacete enviaria um pacote de notificação para os dispositivos acoplados ao corpo do usuário avisando-o que está em uma situação de perigo. Além de tais dispositivos, o capacete poderia reagir também com sinais luminosos situados nele próprio, a fim de alertar o ciclista e/ou também o motorista mostrando que existe um algo em sua frente que na qual, necessita de sua atenção para não haver uma colisão.

Já os dados de atividade física podem ser obtidos por meio de acelerômetros, giroscópios, GPS, RTC, na qual permitem a interpretação dos dados por meio de cálculos, gerando informações de posicionamento, orientação, velocidade, tempo, além de gasto energético, velocidade média percorrida e até um sistemas de conquista de metas. É

possível desenvolver um sistema na qual relacione todos os sensores como o exemplo de que o sensor de risco só seja ativado quando o ciclista esteja se movimentado, situação na qual utiliza os sensores de distância e acelerômetro. Os dados de todos os sensores seriam gravados em uma memória não volátil para acesso e sincronização com um *smartphone* futuramente.

A seguir será exibido vários procedimentos responsáveis por demonstrar a execução do dispositivo. O primeiro, Algoritmo 5.2, exibe o procedimento para leitura de sensores e o armazenamento de tais informações em memória não-volátil. Caso verificado que existe uma situação de risco acontecendo por meio da leitura, também será realizado o envio de um sinal de interrupção para o controlador a fim de que os procedimentos de aviso sejam realizados.

Algoritmo 5.2: *Wearable* 2 - Procedimento que realiza leituras de sensores e armazena seus dados em memória não-volátil.

Input: sensors reads.
Output: backups and interruptions signals.

```
1  begin
       // statements
2      sensor_i ;                                  // i information's source
3      datas ;                                // data's matrix before handling
4      type_warning ;                         // type of warning to do to user
5      while wearable mode on do
6          datas ← reads_sensors(sensor_i);
           // process the datas
7          type_warning ← processes(datas);
8          if type_warning = dangerous then
9              does_interruption(type_warning);
10         end
11         backup(datas);
12     end
13 end
```

Dessa forma, o Algoritmo 5.2, na linha 6, realiza leitura dos sensores e em seguida verifica se existe alguma gravidade na situação. Se positivo, gera o sinal de interrupção. Como esse módulo é um módulo independente e gerador de interrupções, ao realizar uma, não necessita de criar-se um bloqueio em seus procedimentos. Isso significa que o módulo poderá continuar com suas instruções de ler, analisar e gravar os dados, independente

do controlador estiver ou não com interrupção.

Após o controlador receber a interrupção, inicia-se o processo do Algoritmo 5.3 para tratá-la. A função `does_interruption` é estimulada sempre que um sinal de interrupção é ativado ou mantido ativo. Após iniciada, tipo da interrupção é gravado numa variável auxiliar com nome `buffer_type_warning`. Isso é importante pois, caso a interrupção seja retirada no meio da função, não permita a possibilidade de um dispositivo realizar o alerta e outro não, criando uma confusão no compreendimento do usuário sobre sua situação. Como já dito, o processo de alerta ficará ativo enquanto a interrupção estiver ativada e assim, como exibido na linha 4, o procedimento continuará em repetição enquanto o valor não ser alterado, na linha 10. O primeiro passo a se fazer ao receber a interrupção é verificar se o capacete possui conexão sem-fio com os equipamentos que possuem suporte (como o *smartphone* e o *wearable*). Se sim, envia sinal do tipo de interrupção para os dispositivos para que eles façam a parte deles. Como é possível que nenhum dispositivo esteja conectado, foi idealizado também uma forma de aviso integrado ao capacete, a fim de não ficar à mercê desses para manter a segurança do ciclista. Dessa forma, o dispositivo também contaria com *leds* na qual seriam ativados quando estivesse em uma situação de risco, com é exibido na linha 9.

Ao final do Algoritmo 5.3, quando a interrupção é retirada, o `while` da linha 4 é tido como falso e assim, na linha 12, é realizado o término da interrupção em todos os dispositivos voltando para o estado normal.

Por fim, tem-se o Algoritmo 5.4 que fica responsável dos módulos que necessitam de controle para funcionamento. Isso é necessário em procedimentos de sincronização e controle, sendo que neste caso é a sincronização dos dados de atividades obtidas pelos sensores para o aplicativo no *smartphone*.

Tendo o projeto em mãos, realiza-se sua análise, usando primeiramente o *profile*.

Análise Usando Profile Tal como o exemplo hipotético anterior (Seção 5.1), aplica-se o processo de análise execução neste projeto a fim de identificar procedimentos que necessitam de muitos recursos ou de um tempo maior para processamento. Isso é obtido com o uso da ferramenta *profile*.

Grafo Em seguida, tendo o projeto em mãos, realiza-se a construção de seu grafo de chamada, no qual é exibido na Figura 5.3.

Algoritmo 5.3: *Wearable* 2 - Procedimento que realiza os procedimentos após um sinal de interrupção recebido pelo sensor de distância.

Input: interruption info.
Output: interruption warnings.

```
1  type_warning ;                                        // interruption signal
2  Function does_interruption do
3  |  buffer_type_warning ← type_warning ;      // gets the last interruption
   |   information
4  |  while is_there_interruption_yet(buffer_type_warning) do
5  |  |  if verifies_connection() then
6  |  |  |  tries_send_to_smartphone(buffer_type_warning);
7  |  |  |  tries_send_to_wearable(buffer_type_warning);
8  |  |  end
9  |  |  does_leds_warning(buffer_type_warning);
10 |  |  buffer_type_warning ← type_warning ;  // gets the last interruption
   |  |   information
11 |  end
12 |  finishes_interruption();
13 end
```

Algoritmo 5.4: *Wearable* 2 - Procedimentos para controle geral do *wearable*.

Input: sensors reads.
Output: acting.

```
1  begin
2  |  while device mode on do
3  |  |  while verifies_connection() do
4  |  |  |  if does app need fitness informations then
5  |  |  |  |  sends_to_smartphone(gets_datas());
6  |  |  |  end
7  |  |  end
8  |  end
9  end
```

O grafo apresentado possui vários módulos que trabalham tanto colaborativamente quanto independente, e assim, serão mencionados e descritos separadamente a seguir para seu compreendimento.

Antes de falar de cada módulo, será descrito alguns detalhes prévios. É possível

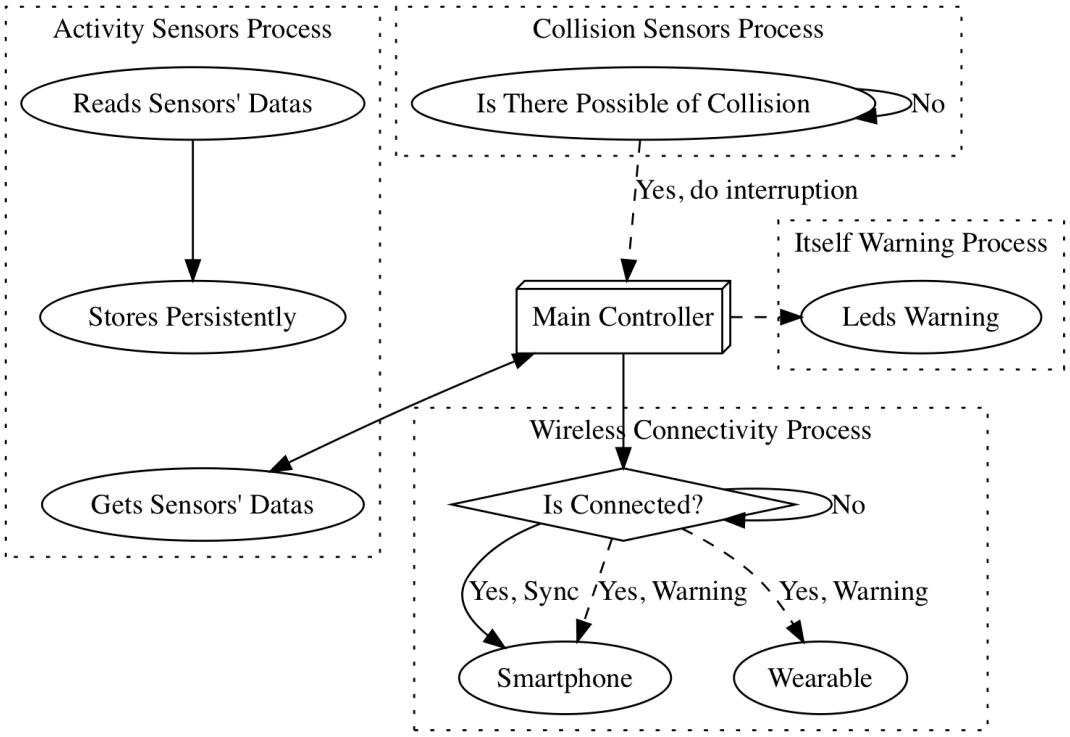

Figura 5.3: Exemplificação de um grafo de chamada para o exemplo hipotético do capacete de segurança.

visualizar que existem dois tipos de setas presente no grafo, isso foi feito como forma de demonstração de barramentos de comunicação na qual constituem de dados de baixa prioridade (troca de informações comuns) e dados com alta prioridade (interrupções). O primeiro, representado por uma seta não-tracejada ($\longrightarrow$) indica informações com baixa e o tracejado ($\dashrightarrow$), informações de alta prioridade, respectivamente.

Sensores de Atividade: Este bloco possui duas funções diferentes sendo a primeira apresenta neste tópico. O procedimento de leitura e armazenamento de dados sensoriais não é dependente do controlador, e com isso o processo (executado pelos procedimentos `lê sensores` e `Armazena Persistentemente`) é executado por si só. Dessa forma, neste processo, o capacete coleta dados e salva-os em memória constantemente a fim de criar um banco temporal de informações sem depender de um controlador para esta ação;

Leitura de Dados *Fitness* e Sincronização destes com o *Smartphone*: O outro processo, ainda relacionado ao modo anterior, parte da leitura dos dados já obtidos e envio desses do capacete para um aplicativo situado em um *smartphone*. Este processo é já necessita de ser controlado por uma controladora e assim, foi

associado com o controlador principal do projeto, para gerenciar com corretude a troca de informações;

Sensor de Colisão: Com a leitura de dados sensoriais, outro processo também independente e de grande importância para o sistema é na análise de situações de risco e a geração de interrupções no controlador, caso exista uma. Esse módulo de sensoriamento de colisão compõe-se de procedimentos que realizam análises constantes de situações de risco do ciclista e enviando mensagens com interrupções para o controlador quando necessário;

Alarmes: Ao receber uma interrupção de colisão, o controlador fará logo em seguida os procedimentos de avisos nos respectivos dispositivos associados ao capacete no momento ocorrido. Dessa, forma o projeto constitui-se de dois tipos de avisos diferentes para o usuário, descritos a seguir:

- ***Alertas para Dispositivos Sem-Fio:*** Como o sistema possui comunicação sem-fio, é possível que ele envie um aviso de alerta para outros sistemas próximos a ele como *smartphone* e *wearables*. Dessa forma, é possível que o *smartphone* faça ações como vibrações ou sinais sonoros, bem como o *wearable* e suas funções, a fim de alertar o usuário.
- ***Alertas Próprio:*** Caso a comunicação não funcione, o dispositivo também terá acoplado a ele um sistema próprio na qual realizará o papel de informar o usuário sobre sua situação de risco. Este funcionará independente dos sistemas externos como meio alternativo de concluir o objetivo de informar o usuário imediatamente.

Com as informações obtidas até o presente momento por meio do *profile* e o formato e fluxo do grafo, é possível identificar possíveis componentes a serem analisados em execução em *hardware*, ou seja candidatos.

Não é possível descrever os candidatos no exemplo hipotético pois necessita-se de informações de execução obtidas pelo *profile*. Entretanto, podemos inferir uma lista de acordo com o propósito final de projeto e esta indução será descrita a seguir. Para um projeto que tenha como foco sua performance e confiabilidade, procedimentos como Verificação de Colisão e Leitura de Sensores de Atividade poderiam ser taxados como candidatos pelo fato de talvez estes se beneficiarem com a utilização de recursos em *hardware* para a obtenção de seus objetivos. O procedimento de aviso integrado ao capacete também seria um candidato à análise, visto que desenvolvendo um circuito

digital para seu acionamento, tem-se então respostas imediatas às interrupções geradas. Estes formam os componentes candidatos ao particionamento, segundo a análise induzida realizada.

Por outro lado, o capacete também tem módulo de conexão sem-fio, requerendo um controlador para seu gerenciamento. Tal controlador deve ser capaz de certificar a conexão de dispositivos compatíveis de forma segura, realizar a transferência de informações de atividades entre o capacete e o aplicativo e tratar as interrupções com prioridade, sempre visando o consumo eficiente de energia. Estes, por sua vez, poderiam continuar em sua implementação em *software* (não tornando candidatos), por se tratarem de procedimentos que não necessitam de performance, nem de grandes recursos de *hardware* para a conclusão de suas tarefas. Os procedimentos compõe principalmente de leitura de dados em memória e seu envio via interface de comunicação sem-fio além do acionamento de tarefas a partir sinais interruptores.

Análises de Síntese Procedimento idêntico ao realizado com o *wearable* hipotético da Seção 5.1.

Solver Procedimento também idêntico ao realizado com o *wearable* hipotético da Seção 5.1.

5.3 Wearable Hipotético 3: Capacete Sensorial

Apresentação Suponha uma outra situação na qual uma empresa que realiza o plantio e colheita de uma determinada fruta, necessita de acompanhar o clima e vários outros dados do ambiente em vários pontos diferentes de seus hectares. Entretanto, a instalação de postes coletores de dados nos seus vastos terrenos seriam um projeto inviável, segundo a empresa.

Feita a respectiva análise de requisitos do projeto, verificou-se que a empresa necessita de informações de: temperatura, pressão, radiação ultravioleta, chuva, luminosidade, fogo, radiação infravermelha, monóxido de carbono, gás natural, gás combustível, além de um sensor de distância em relação ao ponto atual do usuário. Dessa forma, foi apresentado então uma solução que baseia-se em um capacete com realidade virtual na qual realizaria a coleta dos dados por meio de sensores e, além de exibi-los à tela integrada nele, também enviaria-os para uma nuvem na qual apresentaria os dados

para a central de processamento final. Diferentemente dos sensores passivos[2] como sensor de temperatura, chuva e outros, o cálculo da distância partiria de um processo mais complexo, envolvendo várias fases de processamento. O procedimento, de modo introdutório, baseia-se na utilização de dois raios lasers junto à uma câmera externa para avaliar a distância de objetos. O vão entre os dois pontos é relativo à superfície do objeto, na qual altera de acordo com a distância entre o usuário e o objeto. Dessa forma, A distância entre os dois pontos aumenta na imagem captada à medida que a distância entre o usuário e o objeto reduz, e vice-versa.

Assim, o Algoritmo 5.5 retrata, de forma geral, o processo de leitura dos sensores inclusive no cálculo de distância utilizando os lasers. Já o Algoritmo 5.6 gerencia os processos de leitura, exibição dos dados na tela do dispositivo e também o envio deste para a nuvem.

Algoritmo 5.5: *Wearable* 3 - Procedimentos de leitura do ambiente por meio de uma estação de sensores.

Output: data's sensors vector.

```
1  Function reads_sensors do
       // statements
2      vector_data ;                                   // data's vector
       // reads passive sensors
3      vector_data ← reads_temperature();
4      vector_data ← reads_pressure();
5      vector_data ← reads_ultraviolet();
6      vector_data ← reads_rain();
7      vector_data ← reads_luminosity();
8      vector_data ← reads_fire();
9      vector_data ← reads_infrared();
10     vector_data ← reads_carbon_monoxide();
11     vector_data ← reads_natural_gas();
12     vector_data ← reads_gas();
       // reads active sensors
13     vector_data ← reads_distance(image_process());
14     return vector_data ;          // return the datas to the controller
15 end
```

[2]Sensores na qual interpretam os respectivos dados do ambiente de forma passiva.

Algoritmo 5.6: *Wearable* 3 - Método principal para controle.

Output: acting.

```
1 begin
    // statements
2   vector_data ;                                   // data's vector
3   while wearable mode on do
4     vector_data ← reads_sensors();
5     show_informations(vector_data);        // shows the datas on the
        wearable's screen
6     cloud_backup(vector_data);          // saves the data persistently
7   end
8 end
```

Análise Usando Profile Com o processo de análise pela técnica de *profile* é possível identificar os principais procedimentos que requerem o maior tempo de processamento. Procedimento semelhante às análises de *wearables* anteriores (Seções 5.1 e 5.2).

O método de cálculo de distância utiliza de vários procedimentos complexos, possuindo talvez a maior quantidade de recursos de processamento e tempo para a conclusão de suas tarefas. Com o processo de geração de código HDL por meio de HSL, seria possível utilizar de recursos como aceleradores para o processamento de imagem a fim de agilizar o processamento ou a redução de gasto energético, por exemplo.

Grafo O grafo representante do Algoritmo 5.5 é exibido na Figura 5.4. É apresentado o processo de captura de dados pelos sensores passivos e também o método de cálculo de distância por meio dos dos lasers. Ao lado direito do grafo, existe um ponto na qual aponta para o processo de leitura dos sensores. Tal ponto representa a invocação do método, ou seja, os dados dos sensores só serão lidos quando forem solicitados e assim, serão retornados como é exibido pelo segundo ponto a baixo do grafo.

Realizado uma invocação do método de leitura de sensores, os dados de cada sensor passivo serão lidos e armazenados num vetor de informações. Por outro lado, também é realizado a captura de imagens para processamento da distância na qual o processo consiste, de forma geral, em três etapas, sendo elas a captura e o processamento da

imagem e o cálculo da distância em relação à distância dos pontos.

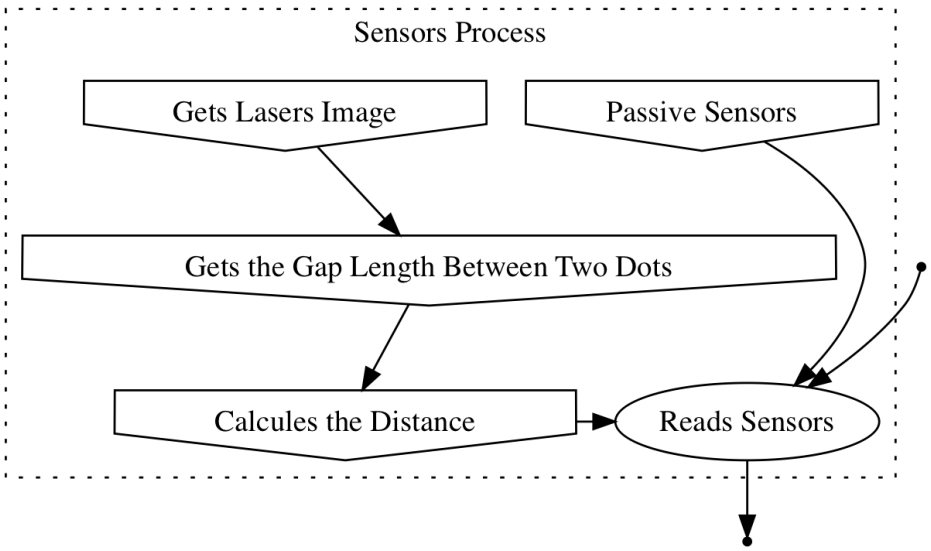

Figura 5.4: Exemplificação de um grafo de chamada para o exemplo.

Já o Algoritmo 5.6 é representado pelo grafo da Figura 5.5. Constitui-se do processo principal na qual invoca o procedimento de leitura de sensores. Como este é basicamente para controle, nele ordena-se os passos de leitura, exibição e envio dos dados para uma nuvem, caso o dispositivo permaneça ligado neste processo. O módulo de leitura de sensores exibido no canto esquerdo da Figura 5.5 simplifica o processo da Figura 5.4, já explicada.

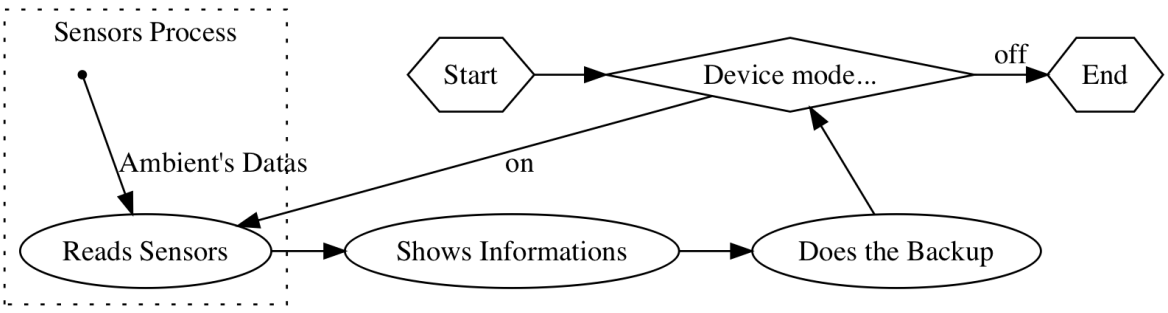

Figura 5.5: Exemplificação de um grafo de chamada para o exemplo.

Análises de Síntese Idem ao processo das seções anteriores, na qual realiza uma visão do projeto com os candidatos apresentados pelas ferramentas e sintetiza-os a fim de avaliar suas características de funcionamento.

Solver Idem às análises anteriores (Seções 5.1 e 5.2).

Capítulo 6

Conclusões Parciais e Trabalho Futuro

Neste capítulo serão descritas as conclusões parciais da pesquisa realizada até o momento e as próximas etapas a serem executadas.

6.1 Conclusões

Mesmo a tecnologia embarcada tenha tido um grande avanço nos últimos tempos sendo utilizada nos mais diversos fins, sistemas *wearables* tiveram seu primeiro aparecimento em pesquisas científicas em 1996 por Mann. Isso, por causa de empecilhos tecnológicos e ferramentais como a necessidade da miniaturização, mobilidade e eficiência energética da tecnologia dos equipamentos utilizados na qual, sem tais especificações, seria inviável a utilização do *wearable*.

Existe várias pesquisas relacionadas à área de sistemas embarcados no âmbito de particionamento de *hardware* e *software*, inclusive utilizando FPGAs como meio. Entretanto, até o momento deste, não foi encontrado nenhum relato de experiências científicas sobre o particionamento para sistemas *wearables* com utilização de recursos de *hardwares* reconfiguráveis, tema tratado neste documento.

Objetivos realizados até o presente momento:

- Introdução de sistemas computacionais *wearables* e apresentação do problema de particionamento *hardware* e *software* no âmbito de de sistemas computacionais embutidos, com foco em sistemas *wearables*;

- Apresentação de:
  - Principais soluções apresentadas ao logo dos anos e as utilizadas atualmente;
  - Ferramentas HLS como LegUp e OpenCL para a geração de sistemas computacionais que usufruem de aceleradores em *hardware*.
- Apresentação da abordagem metodológica a ser utilizada para a procura da solução do problema de particionamento *hardware* e *software* apresentado.

6.2 Trabalho Futuro

A seguir será apresentado os tópicos relativos às próximas etapas a serem realizadas neste trabalho.

- Geração de HDL, utilizando as ferramentas automatizadas quando aplicável:
  - LegUp;
  - OpenCL.
- Realizar-se-á análises de algoritmos situados em sistemas diferentes, sendo esses:
  - **Totalmente em nível de *software*:** sem auxílio de qualquer acelerador como ao utilizar a plataforma Beagle Bone como sendo o sistema embarcado completo para execução em aplicações *wearables*; e em
  - **Híbridos:** formados de processadores e aceleradores utilizando dos recursos de dispositivos reconfiguráveis a fim de prover maior *speedup* utilizando recursos de *hardware*.
- A análise de desempenho será feita com os dados obtidos dos dois sistemas citados acima. Executando o projeto nos dois sistemas, em nível de *software* e híbrido, é possível relacionar o ganho de desempenho realizando as operações apresentadas na Seção 4.2.
- Redução do tempo de seu desenvolvimento até a disposição do produto ao mercado utilizando os recursos que o desenvolvimento com *hardware* reconfigurável proporciona.

Referências Bibliográficas

Abdelhedi, S., Bourguiba, R., Mouine, J. and Youssef, A.: 2016, Fall Detection FPGA-Based Systems : A Survey, *International Journal of Automation and Smart Technology* **6**(4), 191–202.
URL: *http://www.ausmt.org/index.php/AUSMT/article/view/1105*

Ahola, T., Korpinen, P., Rakkola, J. and Teemu, R.: 2007, Wearable FPGA Based Wireless Sensor Platform, *Proceedings of the 29th Annual International Conference of the IEEE EMBS* pp. 2288–2291.
URL: *http://ieeexplore.ieee.org/document/4352782/*

Amorim, V. J. P., Delabrida, S. and Oliveira, R. A. R.: 2017, A constraint-driven assessment of operating systems for wearable devices, *Brazilian Symposium on Computing System Engineering, SBESC*, pp. 150–155.

Arató, P., Juhász, S., Mann, Z. Á., Orbán, A. and Papp, D.: 2003, Hardware-software partitioning in embedded system design, *2003 IEEE International Symposium on Intelligent Signal Processing: From Classical Measurement to Computing with Perceptions, WISP 2003 - Proceedings*, pp. 197–202.

Arato, P., Mann, Z. A. and Orban, A.: 2005, Algorithmic aspects of hardware/-software partitioning, *Acm Transactions on Design Automation of Electronic Systems* **10**(1), 136–156.

Barney, B.: 2009, POSIX threads programming, *Laboratory. Disponível em:< https://computing. llnl. . . . .*
URL: *http://delta.cs.cinvestav.mx/ fraga/Cursos/SistemasOperativos/2015/POSIXthreads.pdf*

Bolsens, I., De Man, H. J., Lin, B., Van Rompaey, K., Vercauteren, S. and Verkest, D.:

1997, Hardware/software co-design of digital telecommunication systems, *Proceedings of the IEEE* **85**(3), 391–417.

Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., Brown, S. and Czajkowski, T.: 2011, LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems, *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '11* p. 33.
URL: *http://portal.acm.org/citation.cfm?doid=1950413.1950423*

Choi, J.: 2016, From Software Threads to Parallel Hardware with LegUp High-Level Synthesis.
URL: *http://search.proquest.com/openview/f3708284b8a4583de0daf686f891b0ae/1?pq-origsite=gscholar&cbl=18750&diss=y*

Cong, J. and Zou, Y.: 2009, FPGA-Based Hardware Acceleration of Lithographic Aerial Image Simulation, *ACM Transactions on Reconfigurable Technology and Systems* **2**(3), 1–29.
URL: *http://portal.acm.org/citation.cfm?doid=1575774.1575776 http://dl.acm.org/citation.cfm?id=1575774.1575776*

Czajkowski, T. S., Aydonat, U., Denisenko, D., Freeman, J., Kinsner, M., Neto, D., Wong, J., Yiannacouras, P. and Singh, D. P.: 2012, From OpenCL to high-performance hardware on FPGAs, *Proceedings - 22nd International Conference on Field Programmable Logic and Applications, FPL 2012*, pp. 531–534.

Delabrida, S., D'Angelo, T., Oliveira, R. and Loureiro, A.: 2016, Building wearables for geology.

Edwards, M. and Forrest, J.: 1994, A development environment for the cosynthesis of embedded software/hardware systems, *and Test Conference, 1994. EDAC, The ...* .
URL: *http://ieeexplore.ieee.org/abstract/document/326834/*

Ernst, R., Henkel, J. and Benner, T.: 1993, Hardware-Software Cosynthesis for Microcontrollers, *IEEE Design and Test of Computers* **10**(4), 64–75.

Fort, B., Canis, A., Choi, J., Calagar, N., Lian, R., Hadjis, S., Chen, Y. T., Hall, M., Syrowik, B., Czajkowski, T., Brown, S. and Anderson, J.: 2014, Automating the Design of Processor/Accelerator Embedded Systems with LegUp High-Level Synthesis, *Proceedings - 2014 International Conference on Embedded and Ubiquitous Computing, EUC 2014*, pp. 120–129.

Gajski, D., Vahid, F., Narayan, S. and Gong, J.: 1994, Specification and design of embedded systems.
URL: *http://tocs.ulb.tu-darmstadt.de/4948270X.pdf*

Gemperle, F., Kasabach, C., Stivoric, J., Bauer, M. and Martin, R.: 1998, Design for wearability, *Digest of Papers. Second International Symposium on Wearable Computers (Cat. No.98EX215)*, IEEE Comput. Soc, pp. 116–122.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=729537 http://ieeexplore.ieee.org/document/729537/*

Graham, S. L., Kessler, P. B. and Mckusick, M. K.: 1982, Gprof: A call graph execution profiler, *Proceedings of the 1982 SIGPLAN symposium on Compiler construction - SIGPLAN '82*, Vol. 17, ACM Press, New York, New York, USA, pp. 120–126.
URL: *http://portal.acm.org/citation.cfm?doid=800230.806987*

Gupta, R. K., By-Micheli, R. K. and De, G.: 1995, *Co-synthesis of hardware and software for digital embedded systems*, Kluwer Academic Publishers.
URL: *http://dl.acm.org/citation.cfm?id=546156*

Hardt, W.: 1995, An automated approach to HW/SW-codesign, *IEE Colloquium on Partitioning in Hardware-Software Codesigns*, number figure 1, p. 4.
URL: *http://link.aip.org/link/IEESEM/v1995/i32/p4/s1&Agg=doi*

Hassine, S. B. H., Jemai, M. and Ouni, B.: 2017, Power and Execution Time Optimization through Hardware Software Partitioning Algorithm for Core Based Embedded System, *Journal of Optimization* .
URL: *https://www.hindawi.com/journals/jopti/2017/8624021/abs/*

Hennessy, J. L. and Patterson, D. A.: 2011, *Computer architecture: a quantitative approach*, Elsevier.
URL: *https://books.google.com.br/books?hl=pt-BR&lr=&id=gQ-fSqbLfFoC&oi=fnd&pg=PP1&dq=quantitative+archtecture&ots=mXAuKJX3qn&sig=E7_UQzqEFm*

Hidalgo, J. I. and Lanchares, J.: 1997, Functional partitioning for hardware-software codesign using genetic algorithms, *Conference Proceedings of the EUROMICRO*, pp. 631–638.

Jigang, W. and Thambipillai, S.: 2004, A branch-and-bound algorithm for hardware/-software partitioning, *Signal Processing and Information* .
URL: *http://ieeexplore.ieee.org/abstract/document/1434407/*

Jóźwiak, L.: 2017, Advanced mobile and wearable systems, *Microprocessors and Microsystems* .
URL: *http://www.sciencedirect.com/science/article/pii/S0141933117300741*

Kern, N., Schiele, B., Junker, H., Lukowicz, P. and Troster, G.: 2002, Wearable sensing to annotate meeting recordings, *Proceedings - International Symposium on Wearable Computers, ISWC*, Vol. 2002-January, pp. 186–193.

Kymissis, J., Kendall, C. and Paradiso, J.: 1998, Parasitic power harvesting in shoes, , *1998. Digest of . . .* .
URL: *http://ieeexplore.ieee.org/abstract/document/729539/*

Lee, B.: 2015, Design and Implementation of a Face Recognition System-on-a-Chip for Wearable / Mobile Applications, *Journal of Korea Multimedia Society* **18**(2), 244–252.
URL: *http://www.kpubs.org/article/articleDownload.kpubs?downType=pdf&articleANo=MTMDCW*

Lee, L., Lee, J., Egelman, S. and Wagner, D.: 2016, Information disclosure concerns in the age of wearable computing, *NDSS Workshop on Usable Security (USEC)*, Vol. 1.

Lo, W. C. Y., Redmond, K., Luu, J., Chow, P., Rose, J. and Lilge, L.: 2009, Hardware acceleration of a Monte Carlo simulation for photodynamic treatment planning, *Journal of Biomedical Optics* **14**(1), 014019.
URL: *http://biomedicaloptics.spiedigitallibrary.org/article.aspx?doi=10.1117/1.3080134*

Madsen, J., Grode, J., Knudsen, P. V., Petersen, M. E. and Haxthausen, A.: 1997, LYCOS: the Lyngby Co-Synthesis System, *Design Automation for Embedded Systems* **2**(2), 195–235.
URL: *http://dx.doi.org/10.1023/A:1008884219274*

Mann, S.: 1996, Smart clothing: the shift to wearable computing, *Communications of the ACM* **39**(8), 23–24.
URL: *http://portal.acm.org/citation.cfm?doid=232014.232021*

Mann, S.: 1997, Wearable computing: A first step toward personal imaging, *Computer* .
URL: *http://ieeexplore.ieee.org/abstract/document/566147/*

Mann, Z., Orbán, A. and Arató, P.: 2007, Finding optimal hardware/software partitions, *Formal Methods in System Design* .
URL: *http://www.springerlink.com/index/C73P1G152L286187.pdf*

Mei, B., Schaumont, P. and Vernalde, S.: 2000, A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems, *Proceedings of ProRISC* .
URL: *http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.8984&rep=rep1&type=pdf*

Narumi, T.: 2016, An FPGA-Based Tiled Display System for a Wearable Display, *The Fifth International Conference on Informatics and* .
URL: *https://www.researchgate.net/profile/Natalie_Walker4/publication/310753834_Proceedings_of_Japan_2016/links/5835e29608ae74bb3aa25cff/Proceedings-of-the-Fifth-International-Conference-on-Informatics-and-Applications-Takamatsu-Japan-2016.pdf#page=14*

Nematbakhsh, S., Stitt, G., Vahid, F., Nematbakhsh, S., Stitt, G. and Vahid, F.: n.d., The effect of fpga size on software speedup from hardware/software partitioning.

Nemeth, E., Hein, T. R. and Snyder, G.: 2004, *Manual completo do Linux: guia do administrador*.

Niemann, R. and Marwedel, P.: 1997, An algorithm for hardware/software partitioning using mixed integer linear programming, *Design Automation for Embedded Systems* .
URL: *http://www.springerlink.com/index/K45Q58P1U31458W7.pdf*

Plessl, C., Enzler, R., Walder, H., Beutel, J., Platzner, M., Thiele, L. and Tröster, G.: 2003, The case for reconfigurable hardware in wearable computing, *Personal and Ubiquitous Computing* **7**(5), 299–308.

Rob van der Meulen: 2015, Gartner Says 6.4 Billion Connected "Things"Will Be in Use in 2016, Up 30 Percent From 2015.
URL: *http://www.gartner.com/newsroom/id/3165317*

Sass, R. and Schmidt, A.: 2010, *Embedded Systems Design with Platform FPGAs Principles and Practices*, 1 edn, Morgan Kaufmann.

Shagrithaya, K., Kepa, K. and Athanas, P.: 2013, Enabling development of OpenCL applications on FPGA platforms, *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pp. 26–30.

Smith, D. and By-Zamfirescu, A. F.: 1998, HDL Chip Design: A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog.
URL: *http://dl.acm.org/citation.cfm?id=551302*

Stitt, G., Lysecky, R. and Vahid, F.: 2003, Dynamic hardware/software partitioning: a first approach, *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)* pp. 250–255.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1219003*

Stone, J. E., Gohara, D. and Shi, G.: 2010, OpenCL: A parallel programming standard for heterogeneous computing systems, *Computing in Science and Engineering* **12**(3), 66–72.

Strachacki, M.: 2008, Speedup of branch and bound method for hardware/software partitioning, *Information Technology, 2008. IT 2008. 1st* .
URL: *http://ieeexplore.ieee.org/abstract/document/4621608/*

Sutherland, I.: 1968, A head-mounted three dimensional display, *Proceedings of the December 9-11, 1968, fall joint* .
URL: *http://dl.acm.org/citation.cfm?id=1476686*

The OpenMP API specification for parallel programming: 2015.
URL: *http://www.openmp.org/ http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf*

Tocci, R. J., Widmer, N. S. and Moss, G. L.: 2011, *Sistemas digitais: princípios e aplicações*, Vol. 11, Prentice Hall.

Trappey, A., Shen, W. and Cha, J.: 2016, Special issue editorial on advances in collaborative systems engineering for product design, production and service network, *Journal of Systems Science and Systems* .
URL: *http://link.springer.com/article/10.1007/s11518-016-5313-5*

Trevett, N.: 2008, OpenCL: The open standard for heterogeneous parallel programming, *Group* (November).
URL: *https://www.khronos.org/opencl/*

Trindade, A. B. and Cordeiro, L. C.: 2016, Applying SMT-based verification to hardware/software partitioning in embedded systems, *Design Automation for Embedded Systems* **20**(1), 1–19.
URL: *http://link.springer.com/10.1007/s10617-015-9163-z*

Van Laerhoven, K., Schmidt, A. and Gellersen, H. W.: 2002, Multi-sensor context aware clothing, *Proceedings - International Symposium on Wearable Computers, ISWC*, Vol. 2002-January, pp. 49–56.

Wang, R., Hung, W. N. N., Yang, G. and Song, X.: 2016, Uncertainty model for configurable hardware/software and resource partitioning, *IEEE Transactions on Computers* **65**(10), 3217–3223.
URL: *http://ieeexplore.ieee.org/document/7387705/*

Wolf, W. H. W.: 1994, Hardware-software co-design of embedded systems, *Proceedings of the IEEE* **82**(7), 967–989.

Wu, J. and Srikanthan, T.: 2006, Low-complex dynamic programming algorithm for hardware/software partitioning, *Information Processing Letters* **98**(2), 41–46.

Yan, X.-H., He, F.-Z. and Chen, Y.-L.: 2017, A Novel Hardware/Software Partitioning Method Based on Position Disturbed Particle Swarm Optimization with Invasive Weed Optimization, *Journal of Computer Science and Technology* **32**(2), 340–355.
URL: *http://link.springer.com/10.1007/s11390-017-1714-2*

Zhang, W., Betz, V. and Rose, J.: 2008, Portable and scalable FPGA-based acceleration of a direct linear system solver, *Proceedings of the 2008 International Conference on Field-Programmable Technology, ICFPT 2008*, Vol. 5, ACM, pp. 17–24.
URL: *http://dl.acm.org/citation.cfm?doid=2133352.2133358*

Zhang, Y., Luo, W., Zhang, Z., Li, B. and Wang, X.: 2008, A hardware/software partitioning algorithm based on artificial immune principles, *Applied Soft Computing* **8**(1), 383–391.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S1568494607000257*