UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

FILIPE BACHINI LOPES

Exploração de Espaço de Projeto de Síntese de Alto Nível Orientada a Eficiência Energética

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
2022

‘O começo de todas as ciências
é o espanto de as coisas serem o que são”
— ARISTÓTELES

RESUMO

FPGAs têm ganhado força na área de sistemas embarcados, com sua união de flexibilidade, desempenho e custo operacional. Novas técnicas e heurísticas vêm surgindo para facilitar o desenvolvimento de aplicações para sistemas embarcados utilizando esse dispositivo, diversificando ainda mais as possibilidades de uso do FPGA. Com o aumento da complexidade das aplicações, restrições como desempenho, uso de recursos e potência, começam a ser atreladas ao desenvolvimento. Para endereçar essa questão, surgem heurísticas de exploração de espaço de projeto, buscando encontrar implementações que otimizem essas múltiplas métricas. No entanto, ainda são poucas as propostas que têm foco em combinar síntese de alto nível e exploração de espaço de projeto orientados às métricas de potência e consumo de energia. Métricas essas que tem grande relevância dentro do escopo de sistemas embarcados. Aliando HLS, DSE e otimização de desempenho e potência, apresentamos aqui uma plataforma de exploração de espaço de projeto orientada a eficiência energética para FPGAs.

Palavras-chave: FPGA. HLS. DSL. Eficiência energética. DSE.

Design Space Exploration with High-Level Synthesis oriented for Energy Efficiency

ABSTRACT

FPGAs are gaining attention in the embedded systems area, with the union of flexibility, performance, and operational costs. New methods and heuristics are advancing to facilitate the development of applications using these devices, increasing the utilization of FPGAs. With the increase in application complexity, restrictions in terms of performance, resource usage, and power consumption affect the development process. Addressing this matter, we have the design space exploration heuristics, using different techniques to find the best implementation of a design that optimizes these metrics. However, few proposals aim to combine high-level synthesis and design space exploration oriented to power consumption and energy efficiency. These two metrics have major relevance in the embedded systems area of study. Combining HLS, DSE, and latency and power consumption optimization, we propose a design space exploration platform focused on energy efficiency for FPGAs.

Keywords: FPGA, HLS, DSL, Power Consumption, DSE.

LISTA DE ABREVIATURAS E SIGLAS

ASIC	Application Specific Integrated Circuits
FPGA	Field Programmable Gate Array
DSL	Domain Specific Languages
DSE	Design Space Exploration
HLS	High Level Synthesis
HDL	Hardware Description Language
CLB	Configurable Logic Block
LUT	LookUp Table
RTL	Register Transfer Level
JSON	JavaScript Object Notation

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Dentro da área de sistemas embarcados, diversos dispositivos dedicados têm seu uso cada vez mais comum, uma vez que com o avanço da tecnologia e a alta demanda de produtos eletrônicos pela indústria, aumenta-se a necessidade de dispositivos específicos e de alto desempenho. Geralmente estes dispositivos têm seu uso restrito às suas áreas características, uma vez que o poder de processamento tende a ser otimizado para uma determinada tarefa. No entanto, dispositivos lógicos programáveis têm potencial na utilização em sistemas eletrônicos sofisticados e em diversas áreas de atuação. O principal destes é o Field-Programmable Gate Array (FPGA), um dispositivo reprogramável baseado em circuitos lógicos multiplexados. O grande diferencial do FPGA está na possibilidade de reconfiguração, o que permite mudanças de projeto sem a necessidade de alterações no hardware. Essa flexibilidade expande as possibilidades de uso, mesmo que sua performance ainda não atinja os níveis obtidos por Application-Specific Integrated Circuits (ASICs) (AMARA; AMIEL; EA, 2006). Por outro lado, alterações mais rápidas e menos custosas são um fator importante em qualquer projeto.

A aceleração em hardware vem ganhando notoriedade, uma vez que diversas áreas podem se beneficiar do desempenho computacional destes dispositivos, no entanto, o desenvolvimento de designs complexos e eficientes ainda pode ser desafiador. No inicio do desenvolvimento de aplicações para FPGA, era necessária a descrição do projeto em HDL, uma linguagem especifica para o desenvolvimento de aplicações para hardware, focada na descrição da estrutura e o comportamento de circuitos lógicos, com pouca abstração. Para geração de projetos maiores, é necessário que o desenvolvedor tenha um conhecimento ostensivo da linguagem e do hardware alvo. Para diminuir essa complexidade, foi desenvolvido o conceito de síntese de alto nível, que engloba técnicas de compilação e tradução de linguagens de mais alto nível (e.g. C, Java, Domain-Specific Languages (DSLs)) para descrição de hardware (NANE et al., 2016). Com o avanço das linguagens e das técnicas envolvidas, o desenvolvimento de aplicações diversas se torna mais prático, e o desempenho do FPGA mais eficiente.

Esse progresso leva à viabilidade de desenvolvimento de designs mais complexos para FPGAs. Sob outra perspectiva, o avanço da complexidade traz consigo restrições específicas de projeto, como área, recursos, desempenho, entre outros, afetando o desenvolvimento dessas aplicações e a integração com um FPGA alvo. Mesmo com os avanços e a abstração trazida pela HLS, o programador ainda fica a cargo da correta parametrização

das otimizações e do dimensionamento dos recursos para atender as diferentes restrições de projeto. Em um cenário de múltiplos pontos de otimização e diferentes combinações de implementação é necessário um método de escolha de designs. As heurísticas relacionadas a análise e identificação de boas implementações orientadas por parâmetros tipicamente realizam Design Space Exploration (DSE), ou Exploração de Espaço de Projeto. DSE tem um escopo geral, não sendo limitada a sistemas com restrições, uma vez que tem relação com técnicas de poda e escolha eficiente de soluções a partir de parâmetros pré-definidos. Nessa linha, diferentes técnicas são utilizadas para seleção e implementação de designs para FPGAs (O'NEAL; BRISK, 2018).

Nessa linha de restrições, sistemas embarcados têm um foco especial em baixo uso de potência, uma vez que o consumo energético pode ser um fator limitante em diversos casos. O FPGA, dentre os sistemas embarcados, tem um consumo energético elevado, uma vez que a flexibilidade exige uma lógica mais complexa. Sendo assim a busca por designs mais eficientes em relação a potência para FPGAs é continuo (SUN et al., 2010).

1.1 Motivação

Concomitante ao avanço da tecnologia dos FPGAs, temos o avanço das tecnologias de HLS, com o desenvolvimento de novas linguagens, técnicas de compilação e geração de RTL mais eficientes (LAHTI et al., 2019). Em conjunto com HLS, diversas técnicas de DSE têm emergido para guiar essa geração, buscando as melhores implementações de forma ágil e efetiva (SCHAFER; WANG, 2020).

Um grande impulso para a utilização de HLS é a difusão de Linguagens de Domínio Especifico (DSLs) voltadas para FPGA. DSLs têm como base linguagens de maior abstração e múltiplos conceitos de programação de alto nível, como orientação a objetos e linguagens funcionais. Por baixo dessa abstração, diversas camadas de compilação e geração de RTL removem do programador a necessidade de descrever fielmente como o hardware deve se comportar, assim, focando seu tempo no desenvolvimento da aplicação em si (KAPRE; BAYLISS, 2016). Além do aumento da produtividade, ferramentas de DSL podem conter otimizações dentro de sua compilação, não só no nível de código, como também em espaço de projeto. Como é o caso do Spatial (KOEPLINGER et al., 2018), uma linguagem baseada em Scala, que durante a compilação de um design, efetua a exploração do espaço de projeto em termos de área e desempenho utilizando o Hypermapper (NARDI; KOEPLINGER; OLUKOTUN, 2019), uma ferramenta independente de

DSE.

No entanto, grande parte das implementações de DSE em conjunto com HLS fazem a busca de espaço de projeto priorizando apenas as métricas de área, desempenho e/ou latência (LIU; SCHAFER, 2016; LIU; LAU; SCHAFER, 2019; SIRACUSA et al., 2019). Mesmo que estes sejam parâmetros de grande importância, no momento em que endereçamos sistemas embarcados, é necessário avaliar também a potência, buscando a melhor eficiência energética.

Com o avanço da tecnologia de semicondutores utilizada pelos FPGAs, aliado à busca de maiores performances, o consumo de potência se torna uma preocupação central (SYLVESTER; KAUL, 2001). O aumento no uso de potência está atrelado também aos custos operacionais relacionados ao FPGA, como por exemplo, gastos de energia com o funcionamento do chip e dissipação de calor. Sendo assim, se faz necessário que a potência seja uma das métricas envolvidas quando trabalhamos com exploração de espaço de projeto em conjunto com HLS.

Uma vez que o consumo de potência ganha relevância, avaliar a implementação de aplicações sob a utilização desta métrica se torna indispensável. No entanto, a obtenção dos valores de potência é complexo, uma vez que envolve informações específicas do FPGA alvo, como também dados de tempo de execução, como chaveamento de portas e roteamento de sinais (SUN et al., 2010). Mesmo que esse tempo seja mitigado por ferramentas de simulação, em termos de DSE, com múltiplos pontos de exploração, o tempo computacional para obtenção dos valores de potência pode ser inaceitável. Sendo assim, técnicas de estimação de potência tendem a ganhar espaço, agilizando o processo de exploração, embora com precisão reduzida.

1.2 Contribuições

Neste documento será apresentado o desenvolvimento de uma plataforma de exploração de espaço de projeto orientada à métrica de eficiência energética. Diretamente ligada a ferramentas de HLS e de DSE, nossa plataforma tem como objetivos: analisar a geração de designs a partir de síntese de alto nível; verificar o espaço e as combinações de pontos de projeto; analisar e gerar valores de potência para o design alvo; buscando assim a melhor implementação em termos de uso de potência.

Usando diferentes aplicações para teste de nossa implementação e comparando com o método tradicional de área x desempenho, pretende-se apresentar um ganho em

termos de eficiência energética sem acrescentar grande complexidade ou ônus em tempo de projeto.

As contribuições deste trabalho incluem:

- Investigar as ferramentas de HLS, DSL e DSE, buscando uma integração para nossa plataforma.
- Desenvolver um estimador de potência para a exploração de espaço de projeto ágil desta métrica.
- Desenvolver uma plataforma completa de análise, estimação e geração de designs voltados ao uso de potência.
- Realizar experimentos para avaliar e comparar os ganhos de eficiência energética em comparação a plataformas de HLS e DSE tradicionais.

As contribuições acima elencadas constituem a plataforma PowerMapper. Utilizando a estimativa de potência dentro do laço de DSE, conseguimos reduzir o consumo de energia dos designs avaliados em 9.75%, em média, enquanto o tempo de processamento foi mantido dentro de patamares razoáveis.

1.3 Organização do Trabalho

Este trabalho é organizado da seguinte maneira: o Capitulo 2 apresenta uma revisão bibliográfica dos conceitos englobados neste trabalho, bem como os trabalhos relacionados. Na sequência, o Capitulo 3 descreve o desenvolvimento da nossa plataforma, o Powermapper, caracterizando os módulos e a sequência de passos desenvolvidas durante a exploração do espaço de projeto. Após, no Capitulo 4, são apresentados as aplicações que foram utilizadas nos nossos experimentos e os resultados da comparação com Spatial e Hypermapper original. Por fim no Capitulo 5, são apresentadas a discussão final e a conclusão deste trabalho.

2 REVISÃO BIBLIOGRÁFICA

Apresentam-se neste capitulo os fundamentos teóricos e as áreas relacionadas a este trabalho, iniciando com um background sobre o FPGA e seus recursos. Após, são analisados estudos sobre limitações de potência e seu consumo em FPGAs, além de formas de estimação. Conceituam-se, ainda, as Linguagens de Domínio Específico e as heurísticas de Exploração de Espaço de Projeto.

2.1 FPGA

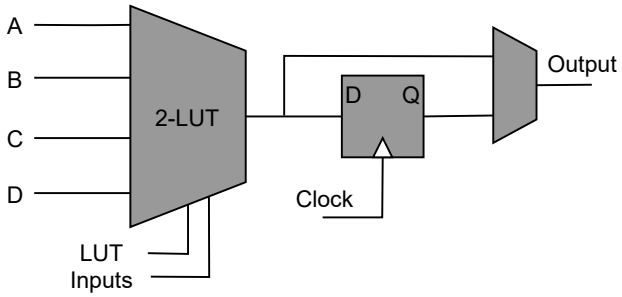
Por trás da reconfiguração e eficiência do FPGA existem diversos recursos e blocos lógicos interligados. A lógica básica é operada nas Configurable Logic Blocks (CLBs), representadas na Fig. 2.1. Esse bloco funciona com uma lógica de multiplexação operada pela LUT, que passa para a saída apenas uma das entradas a partir dos bits de seleção e costuma incluir também flip-flops. Um FPGA tem milhares de blocos como esse, que, em conjunto, podem performar a lógica programável definida pelo projetista. Além deste bloco básico, um FPGA também pode dispor de blocos de memória SRAM, conhecidos como BRAM, blocos específicos para operações matemáticas, visando diminuir tempos de execução, e blocos de entrada e saída para comunicação externa, como pode ser observado na Fig. 2.2. O grande diferencial do FPGA é o seu paralelismo intrínseco, uma vez que esses blocos podem processar dados simultaneamente e em paralelo.

A sequência de passos para o desenvolvimento de uma aplicação para FPGA se inicia no desenvolvimento do design lógico em uma linguagem de descrição de hardware, que será processada por ferramentas apropriadas. Essa ferramenta sintetiza o código RTL para descrever o circuito lógico no nível dos componentes (e.g., LUTs, Flip-Flops) referente ao design implementado e a partir dele gera um arquivo de programação do FPGA, denominado bitstream. Esse arquivo é carregado no FPGA, configurando, então, o hardware para o design desejado.

2.2 Consumo de Potência em FPGAs

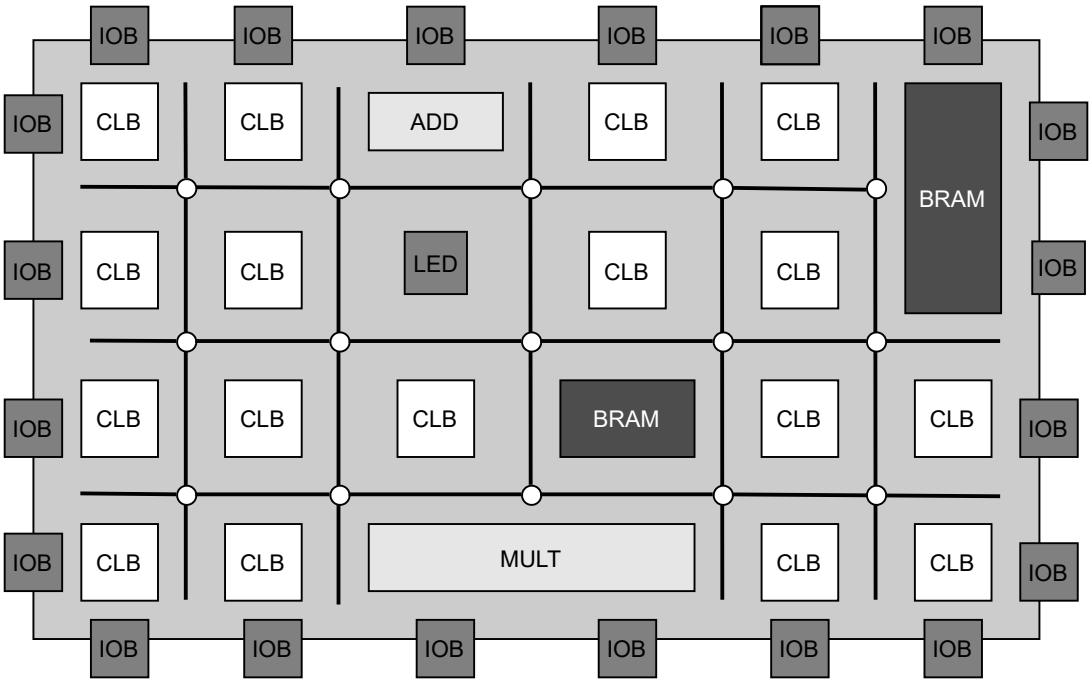
O consumo de potência é um dos fatores mais importantes em qualquer projeto eletrônico. Com o avanço da tecnologia de semicondutores, sistemas cada vez mais sofis-

Figura 2.1 – Representação de um CLB



Fonte: O Autor

Figura 2.2 – Representação dos recursos de um FPGA



Fonte: O Autor

ticados e de alto desempenho têm surgido. Ao mesmo tempo que uma maior densidade de transistores visa aumentar a velocidade de processamento, a potência envolvida ganha cada vez mais atenção.

O consumo de potência no FPGA pode ser divido em consumo de potência estática, referente à corrente de fuga perdida pelos transistores, e potência dinâmica, relativa às trocas de estados dos transistores do FPGA (SUN et al., 2010). Observando a importância de um sistema de baixa potência, diferentes trabalhos buscam a diminuição desse consumo através de diferentes metodologias.

Quanto a potência estática, a busca é diminuir a utilização de circuitos lógicos, por consequência diminuindo a corrente de fuga. Os principais métodos estão em áreas de suspensão, buscando a maior compactação possível da lógica em termos de área e mantendo os recursos não utilizados em modo de suspensão (GAYASEN et al., 2004;

CALHOUN; HONORE; CHANDRAKASAN, 2003). Importante mencionar que esses métodos têm um melhor desempenho em cenários onde o FPGA pode utilizar técnicas de chaveamento de potência (Power Gating) (FLYNN et al., 2007), desligando a alimentação das áreas não utilizadas.

Por outro lado, múltiplas técnicas podem ser utilizadas para diminuir a potência dinâmica (WANG et al., 2006). Uma vez que essa potência está diretamente relacionada ao chaveamento e funcionamento dos recursos, os principais métodos para diminuição são:

- Diminuição da lógica, diminuindo os recursos e circuitos lógicos utilizados.
- Chavemanto de Clock (Clock Gating), podando a árvore de clock de circuitos que não estão sendo utilizados. Note-se que é necessário uma lógica extra para este método.
- Escalonamento Dinâmico de Tensão, uma vez que esta potência está relacionada a tensão e capacitância dos recursos, a diminuição da tensão em certas áreas do FPGA quando possível, diminui a potência dissipada.

Como observado, diferentes métodos podem ser utilizados para diminuir a potência de utilização de um FPGA. Por outro lado, em termos de desenvolvimento e simulação de designs, é necessária uma estimação eficiente dessa métrica, diminuindo o tempo de projeto e a eficiência da implementação.

2.2.1 Estimadores de Potência

Como visto acima, a potência em um FPGA pode ser dividida em estática e dinâmica, cada uma com sua respectiva caracterização. Quando falamos em Estimadores de Potência estamos endereçando os trabalhos de modelagem e predição de potência a partir de diversos atributos e parâmetros. Para uma predição eficiente, potência dinâmica e estática são modeladas e equacionadas para previsão da potência final mesmo durante a implementação do design.

Alguns trabalhos modelam a atividade de transição e a capacitância de interconexão (ANDERSON; NAJM, 2004; DEGALAHAL; TUAN, 2005), analisando o funcionamento e conexão das células básicas do FPGA, modelando seu chaveamento. Outros trabalhos utilizam um treinamento a partir de valores existentes de potência para determinado design, como em (SUNWOO et al., 2010; LAKSHMINARAYANA; AHUJA;

SHUKLA, 2011), onde atributos como recursos, roteamento de sinais, operações e IPs, são modelados para predição de novos valores de potência.

É importante observar que no momento em que trabalhamos com modelagem e estimação de valores de potência, existe uma perda da precisão dos valores encontrados. Isso é esperado quando se trata de estimação, e se engloba no trade-off entre precisão e tempo computacional. Uma vez que o ciclo completo de implementação e simulação de um design para caracterização da potência é demorado, e tende a aumentar com a complexidade do design, essa troca se torna relevante. Outro caso importante dessa troca está nas heurísticas de DSE, já que a caracterização de muitos designs distintos é necessária. Alguns trabalhos, observando a complexidade da obtenção dessa métrica, utilizam de estimadores para auxiliar no processo de DSE, como é o caso do HL-Pow (LIN et al., 2020).

2.3 Linguagens de Domínio Específico

O recente crescimento na popularidade do FPGA tem grande relação com os avanços nas tecnologias de HLS. O desenvolvimento de aplicações para FPGA teve por muito tempo uma lógica restrita, com ferramentas complexas, dificultando um desenvolvimento eficiente de aplicações de alto desempenho. As linguagens referência para FPGAs são HDLs, como Verilog e VHDL, com uma proposta de descrição de hardware, com conceitos pouco abstratos, manipulando diretamente os sinais. Além disso, essas linguagens não acompanham o progresso de recursos, como pode ser visto em diversas linguagens modernas, com macros, classes, objetos, testes automáticos, entre outros (CHEN; CONG; PAN, 2006).

Para diminuir os efeitos negativos no desenvolvimento, surge o conceito de síntese de alto nível, onde linguagens de mais alto nível de abstração (e.g., C, Scala, P, Clojure) são utilizadas para desenvolver soluções para FPGAs. Ferramentas para a compilação e tradução dessas linguagens têm se tornado cada vez mais comuns e eficientes, sendo implementadas até nos programas das empresas referência na área, como no Vivado (XILINX, 2021) da Xilinx e no Quartus (INTEL, 2021) da Intel.

Uma das subáreas relacionadas a HLS são as linguagens de domínio específico (DSL). Essa área aborda linguagens concebidas diretamente para áreas específicas, não somente para sistemas embarcados, dedicadas exclusivamente na solução de um domínio em particular. No caso de FPGAs, essas linguagens observam os fluxos completos do

design, desde a implementação da solução até a descrição de hardware (KULKARNI; BREBNER; SCHELLE, 2004).

Diversas linguagens, baseadas em diferentes linguagens de alto nível, surgiram para utilização em FPGAs (KAPRE; BAYLISS, 2016). Essa variedade de propostas engloba diferentes áreas computacionais, auxiliando no desenvolvimento de múltiplas aplicações.

Como exemplo, temos a linguagem G (BREBNER, 2009), baseada em Click, que busca o melhor uso do paralelismo do FPGA para aplicações de redes. Por outro lado, temos o Chisel (BACHRACH et al., 2012), baseado em Scala, buscando a melhor tradução entre a abstração de orientação a objetos para descrição de hardware.

Um trabalho que deu sequência a linguagem Chisel, foi o Spatial (KOEPLINGER et al., 2018), também baseado em Scala. O Spatial tem foco na melhor caracterização do paralelismo, interfaces e memórias do FPGA, facilitando o uso desses recursos, a partir de funções e tipos auxiliares próprias para este uso. Além disso, essa linguagem tem utilização de otimizações de código e exploração de espaço de projeto. Dessa forma, foi a linguagem escolhida para utilização junto de nossa plataforma.

2.4 Exploração de Espaço de Projeto

O termo Exploração de Espaço de Projeto pode ser definido como a análise e caracterização de pontos de designs favoráveis a partir de uma definição de parâmetros de interesse. Quando relacionamos DSE com FPGA, diversos parâmetros podem ser de interesse para certa aplicação, dentre eles, performance, recursos, área utilizada, consumo de potência, uso de memória, vazão de dados de entrada e saída. Cada cenário pode exigir a otimização de um dos parâmetros, ou uma combinação deles, o que caracteriza uma DSE multiobjetivo. O objetivo do DSE é buscar, utilizando diferentes métodos, o melhor design (ou conjunto de designs) que satisfaça os parâmetros de interesse (PIMENTEL, 2017). Com a combinação de parâmetros, o espaço de busca se torna cada vez mais complexo, uma vez que um amplo conjunto de soluções possíveis começa a aparecer.

Com o aumento da complexidade e restrições das aplicações modernas, tanto para CPUs como em FPGAs, a maioria dos projetos necessita de um DSE multiobjetivo (O'NEAL; BRISK, 2018). Diferentes métodos buscam aumentar a eficiência ao passo que diminuem o tempo necessário para busca de soluções, uma vez que projetos mais complexos podem levar dias de processamento (LIU; LAU; SCHAFER, 2019). A prin-

cipal chave para o processamento eficiente é a utilização de técnicas de poda agressiva, visando eliminar soluções indesejadas rapidamente, antes mesmo da aplicação chegar a estágios de síntese lógica (LIU; SCHAFER, 2016).

Com a difusão do uso de HLS para FPGAs, a utilização de DSE vem ganhando destaque, uma vez que o processo de compilação pode englobar não só a tradução entre linguagem e RTL, como também otimizações de diversos parâmetros. As ferramentas primárias de HLS, baseadas em C, oferecem a possibilidade de otimizações derivadas de pragmas (desenrolamento de loop, pipelines, paralelismo de funções) definidos pelo programador. Todavia, essas ferramentas não têm foco em otimização ostensiva e automática de parâmetros específicos, apenas uma otimização manual definida pelo projetista (KOEPLINGER et al., 2016), o que dificulta a busca de implementações mais eficientes.

Sendo assim, na busca pelo melhor desempenho de HLS, algumas propostas utilizam dentro do seu processo de compilação, métodos de DSE, como por exemplo o MPSeeker (ZHONG et al., 2017), uma ferramenta de DSE para aplicações em C/C++, que busca a melhor utilização do paralelismo do FPGA, utilizando como parâmetros, desempenho e uso de recursos. Por outro caminho, temos o HLscope+ (CHOI et al., 2017), descrevendo uma estimativa de performance baseada em ciclos a partir do código em HLS, aumentando a eficiência na predição de soluções. Outra ferramenta de DSE importante é o Hypermapper (NARDI; KOEPLINGER; OLUKOTUN, 2019), um explorador de espaço de projeto desenvolvido sobre o algoritmo de florestas randômicas de decisão. O Hypermapper em si é uma ferramenta independente e pode ser utilizado em diversas aplicações a partir de uma configuração prévia. Por outro lado, um dos casos de uso dessa ferramenta, é o Spatial (KOEPLINGER et al., 2018), que utiliza o Hypermapper para exploração das métricas de área e desempenho. Por essa comunicação já desenvolvida entre ambas as ferramentas, o Hypermapper foi utilizada como ferramenta de DSE junto de nossa plataforma. Outra questão importante, são as metodologias voltadas para a melhor busca da solução, como por exemplo, utilizando métodos de aprendizado de máquina (LIU; CARLONI, 2013).

2.5 Trabalhos Relacionados

Nosso trabalho engloba estimação de potência, a partir da utilização de HLS, em especifico DSL, junto de ferramentas de DSE na busca da eficiência energética em designs para FPGAs.

Na área de estimação de potência aliada a DSE podemos citar dois trabalhos relacionados. Começando com o *xPlore-Power* (CHEN et al., 2007), uma plataforma de estimação de potência com heurísticas de exploração de espaço de projeto integradas. Essa plataforma utiliza a modelagem de chaveamento e recursos para predição de potência, além disso, técnicas de poda e validação são feitas durante a predição para diminuir os pontos analisados. No entanto essa plataforma tem como entrada diagramas de transição de estados, um modelo de descrição do circuito, enquanto nossa plataforma tem como entrada aplicações em código fonte de DSLs, com mais flexibilidade ao desenvolvedor.

Mais recentemente foi proposto o *HL-Pow* (LIN et al., 2020), uma plataforma de modelagem e estimação de potência aliada a HLS. Este trabalho utiliza uma modelagem de recursos, atividade e operações para estimação de potência, valendo ressaltar que nosso estimador usa como base os estudos apresentados por este trabalho. Além do estimador, é proposto um algoritmo de exploração de espaço de projeto buscando melhorar as métricas, latência e potência, operando em designs descritos na linguagem C.

Na linha de trabalhos que utilizam a exploração de espaço de projeto para minimizar a energia em FPGA, podemos citar o *MultiVers* (LIGNATI et al., 2021), uma plataforma de exploração do multi-versionamento de kernels gerados automaticamente em HLS, buscando otimizar métricas como uso de recursos, performance e consumo de energia em aplicações utilizando o compartilhamento CPU-FPGA em serviços de Cloud. Uma vez que a ênfase de *MultiVers* está na seleção de kernels a partir de uma biblioteca, este trabalho é complementar ao aqui apresentado, já que nossa plataforma objetiva produzir agilmente designs com melhor eficiência energética.

Na linha de HLS aliada a exploração de espaço de projeto, podemos observar o HLscope+ (CHOI et al., 2017), o MPSeeker (ZHONG et al., 2017), citados acima, como também outras plataformas de DSE aplicadas juntas de síntese de alto nível (SHAHSHAHANI et al., 2020). A grande diferença dessas propostas para a nossa, é a priorização e avaliação da métrica de potência dentro do ciclo de exploração de espaço de projeto.

Diferentemente dos trabalhos relacionados, nossa proposta tem como contribuições a adaptação de uma ferramenta de DSE baseada em DSL para otimização de eficiência energética, com suporte de uma ferramenta de estimação de potência baseada em aprendizado de máquina, reimplementada e adaptada para esse propósito específico.

3 IMPLEMENTAÇÃO: POWERMAPPER

Nesta seção será exposto o desenvolvimento de nossa plataforma de exploração de espaço de projeto, bem como as ferramentas utilizadas e a arquitetura final.

3.1 Avaliação e Escolha das Ferramentas

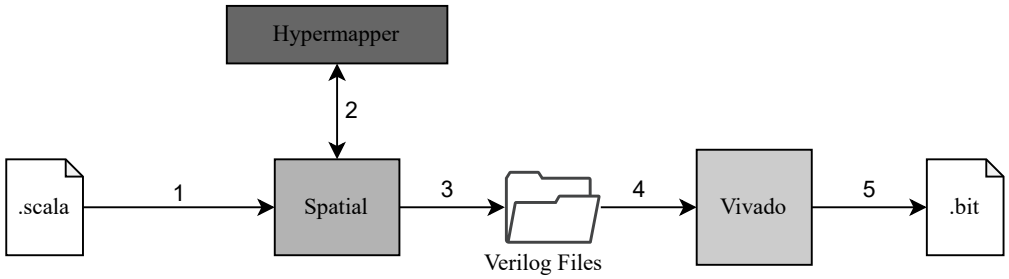
Uma das bases para o melhor desenvolvimento das aplicações, é uma boa escolha de linguagem de aplicação específica para FPGAs. Nossa busca teve três pilares principais. Seria vantajosa a escolha de uma linguagem: que possibilite o desenvolvimento ágil e eficiente de aceleradores para FPGA; que tenha ligação direta com ferramental de DSE; que tenha ferramentas disponíveis em código aberto.

Existem diversas linguagens voltadas para uso em FPGAs (KAPRE; BAYLISS, 2016), com diferentes focos de atuação, no entanto, seguindo os objetivos de busca citados, selecionamos o projeto Spatial (KOEPLINGER et al., 2018). A linguagem Spatial é baseada na linguagem de programação Scala, utilizando as vantagens e abstrações de uma linguagem funcional de alto nível, junto de funcionalidades específicas para o desenvolvimento direcionado para FPGAs. O projeto do Spatial faz a união entre a abstração de linguagens de alto nível, estruturas de otimização de código (e.g., Desenrolamento de Loops, Pipelining, Reduce) e controle sobre memórias e IOs. Além dessas funcionalidades, essa linguagem é integrada com a ferramenta de DSE denominada Hypermapper (NARDI; KOEPLINGER; OLUKOTUN, 2019). Essa ferramenta utiliza o método de florestas randômicas aleatórias para a otimização multiobjetivos, gerando uma fronteira de Pareto entre os diferentes parâmetros buscando as melhores soluções. A combinação entre Spatial e Hypermapper forma a base de HLS da nossa plataforma.

Para observar a implementação de uma aplicação utilizando a linguagem Spatial, foi adicionado no apêndice C, o código da implementação do Quicksort. É possível observar a anotação da variável parallel que busca otimizar a cópia de matriz. Essa anotação cria a possibilidade da utilização de qualquer valor presente na anotação para aquela variável, sendo um parâmetro de DSE. Outro ponto é a utilização do Stream.Foreach, uma função nativa do Spatial, que busca formas de otimizar e paralelizar o for em questão.

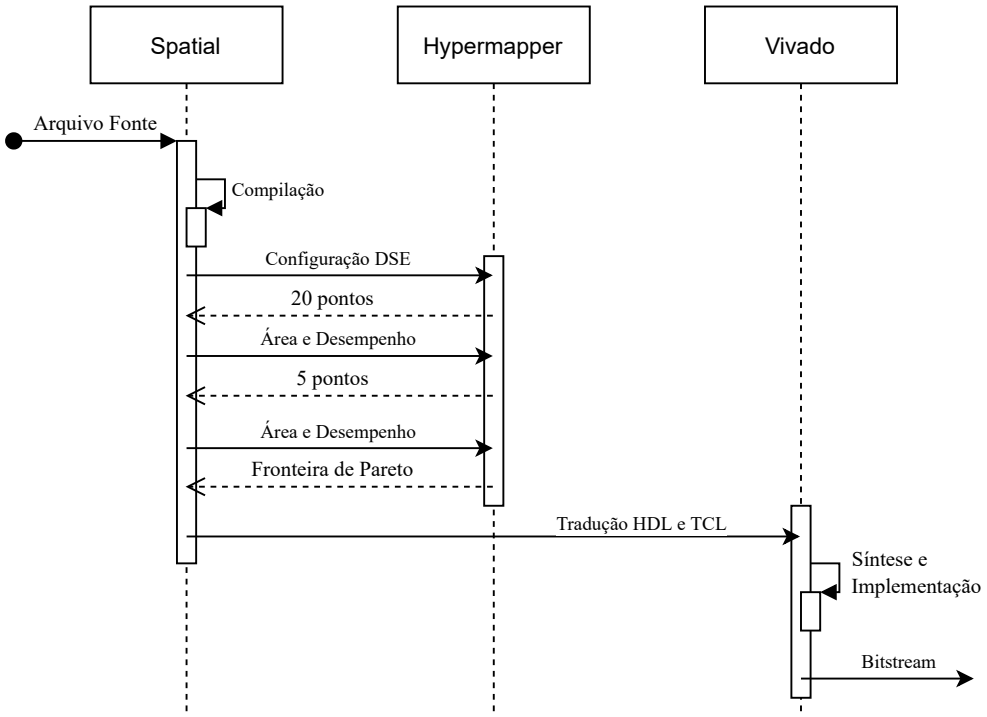
A sequência de funcionamento e a interligação de ambas as ferramentas pode ser observado nas Figuras 3.1 e 3.2. O primeiro passo para a elaboração de uma aplicação com a ferramenta está no desenvolvimento da funcionalidade diretamente utilizando a

Figura 3.1 – Funcionamento Spatial e Hypermapper



Fonte: O Autor

Figura 3.2 – Diagrama de Sequência Spatial e Hypermapper



Fonte: O Autor

linguagem. Spatial oferece toda a abstração de uma linguagem de alto nível com elementos chaves para melhor desenvolvimento de aplicações em FPGA. É possível especificar os trechos de código que serão acelerados em hardware, como também criar macros de paralelismo e redução, facilitando a melhor utilização dos recursos do FPGA alvo. Além disto, anotações de DSE podem ser inseridas facilmente em qualquer variável do projeto, especificando a extensão de valores que podem ser utilizados na exploração da melhor implementação.

Com a aplicação desenvolvida, o processo se inicia na comunicação rotulada com 1 na Fig. 3.1, com o Spatial efetuando a compilação do código utilizando a JVM do Scala. Além de todas as verificações e otimizações feitas em cima da aplicação, o Spatial se comunica com o Hypermapper, indicado pela conexão 2, para desenvolver a exploração

de espaço de projeto, utilizando os parâmetros anotados na aplicação.

O Hypermapper é uma ferramenta de DSE independente, sendo assim, o Spatial utiliza o modo cliente-servidor do Hypermapper, na nomenclatura utilizada pelos autores (NARDI; KOEPLINGER; OLUKOTUN, 2019), para fazer a exploração de espaço de projeto. O Hypermapper é agnóstico aos parâmetros de entrada e a métrica de otimização, ficando a cargo do Spatial tanto a modelagem das informações, como também as decisões a nível de projeto. O Spatial tem, inserido em sua implementação, modelos de recursos e tempo de execução dos FPGAs alvo, que são utilizados para estimar essas métricas e utilizá-las junto do processo de DSE.

A comunicação se inicia com o Spatial enviando ao Hypermapper um JSON contendo as informações básicas sobre o design a ser explorado. Informações como: o número de parâmetros de exploração, o tipo e os valores possíveis para esses parâmetros, as informações sobre o FPGA que serão enviadas como referência (uso de recursos, área e ciclos) e por último quais destas métricas devem ser otimizadas. O Spatial tem, como objetivo, sempre buscar os designs na fronteira de Pareto entre área e ciclos. Esses parâmetros de otimização são fixos, não sendo possível a escolha por parte do desenvolvedor.

Com o Hypermapper inicializado com as informações do design, este solicita uma sequência de pontos para início da exploração de espaço de projeto, para que o Spatial envie as informações de recursos e desempenho de volta. O número de amostras padrão é definido em 20 pontos, mas pode ser alterado pelo programador. O Spatial então, utilizando seus modelos, devolve ao Hypermapper os atributos desses 20 pontos iniciais. Essas informações são então utilizadas pelo Hypermapper para treinamento de seu algoritmo. Após esse treinamento, o Hypermapper solicita 5 pontos adicionais do espaço para o Spatial, para que este envie os valores de recursos, área e ciclos em resposta, assim verificando o treinamento e aperfeiçoando a busca. Esses pontos adicionais servem para análise do treinamento do Hypermapper, independente do número de amostras definido no configuração, o Hypermapper sempre solicita 5 pontos adicionais para validação. Após a troca de informações destes pontos adicionais com o Spatial, o Hypermapper tem a fronteira de Pareto completa e devolve estes valores.

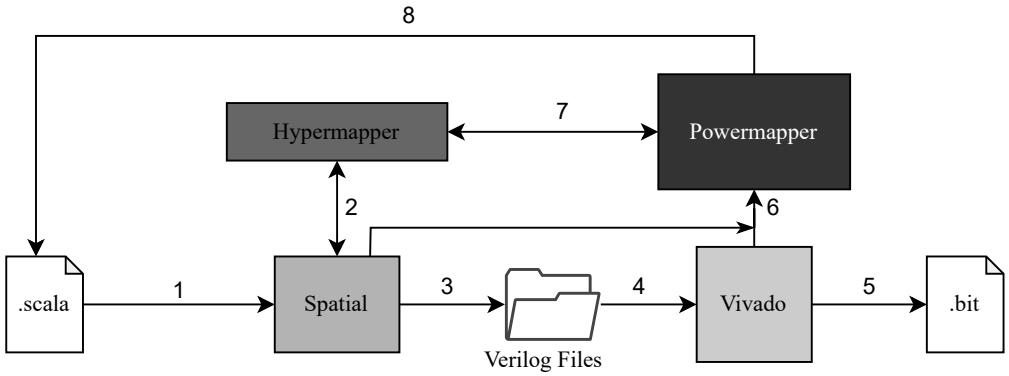
Sendo assim, após esta comunicação, o Spatial tem os designs que otimizam as métricas de seu interesse. Partindo de qualquer um destes designs, então, o Spatial pode finalizar a compilação e implementar a aplicação completamente, criando os arquivos em HDL, pelo caminho indicado na seta de rótulo 3. Com estes arquivos gerados, é possível sintetizar e implementar a aplicação junto ao Vivado indicado pelo passo 4. O Vivado

então, após o fluxo completo de síntese e implementação, gera um arquivo bitstream para programação do FPGA alvo, que é saída deste processo, rotulado com 5.

3.2 PowerMapper

A nossa plataforma tem como objetivo adicionar a métrica de potência no processo de implementação do Spatial, expandindo o processo de exploração de projeto, além de dar suporte a minimização do gasto de energia em FPGAs.

Figura 3.3 – Plataforma inserida no ciclo completo



Fonte: O Autor

O processo desenvolvido pode ser observado na Fig. 3.3, com a inserção do Powermapper no ciclo do Spatial + Hypermapper. Pode ser observado no apêndice na Fig. B.1, o diagrama de sequência do funcionamento do Powermapper, mostrando a troca de informações após o primeiro ciclo do Spatial.

Recebendo os arquivos HDL gerados, os relatórios de implementação do Vivado, junto dos arquivos de configuração da aplicação gerada pelo Spatial, nossa ferramenta tem todos os recursos necessários para adicionar a métrica de potência ao ciclo de DSE.

O ciclo completo de geração do Hypermapper original de uma aplicação com o Spatial termina na implementação e geração do bitstream pelo Vivado, apontado no caminho 5 da Fig. 3.3. Nesse momento, temos os arquivos gerados pelo Spatial para a aplicação, os arquivos de exploração de espaço de projeto gerados pelo Hypermapper, e os relatórios de uso de recursos e potência do FPGA gerados pelo Vivado.

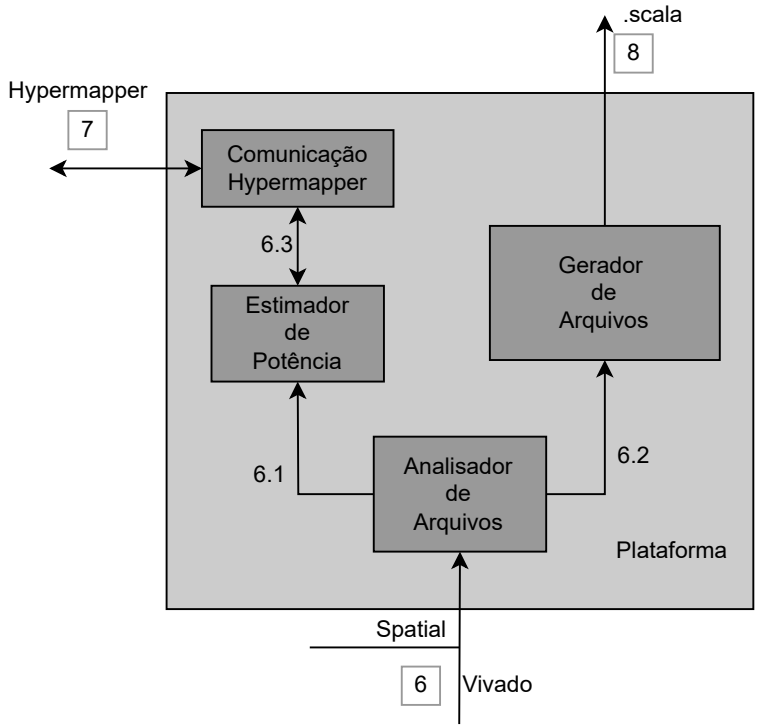
Neste ponto, nosso objetivo é fazer a exploração de espaço de projeto a nível de potência. Para isso, existem dois caminhos possíveis: obter a informações de potência sobre todos os pontos de exploração analisados na exploração de espaço de projeto ou modelar e treinar um estimador com base em diferentes parâmetros para avaliar a potência

para qualquer ponto necessário.

Esses dois caminhos apresentam um trade-off entre tempo computacional e qualidade na exploração. Por um lado, gerar, sintetizar e implementar todos os pontos do design pode levar dezenas de horas, no entanto, tem como resultado um valor muito mais preciso para o melhor design em potência. Por outro lado, utilizar um estimador de potência tem um processo de aprendizagem que pode levar apenas minutos, porém com uma precisão mais baixa. Uma vez que para designs pequenos (e.g., aproximadamente 1000 pontos de exploração), podemos levar 2 dias para avaliar todos os pontos, como será apresentado na Seção 4, o método de avaliação de força bruta não é escalável e impraticável para muitos designs.

Em contrapartida, o estimador necessita apenas que alguns pontos do design sejam sintetizados, para que os relatórios de uso de recursos e potência, gerados no processo de implementação do Vivado, sirvam de base para o treinamento do mesmo.

Figura 3.4 – PowerMapper



Fonte: O Autor

A plataforma em si pode ser divida em 4 módulos principais, como pode ser observado na Fig. 3.4. Todos os blocos a seguir, além da sua integração com o ferramental original do Spatial e Hypermapper, foram desenvolvidos no contexto deste trabalho. A seguir serão analisados cada um dos blocos separadamente, como também observadas as três novas comunicações adicionadas no ciclo original observado na Fig. 3.1.

3.2.1 Analisador de Arquivos

A porta de entrada dos dados na plataforma é o analisador de arquivos. Uma vez que muitas informações, de fontes diferentes, são utilizadas, é necessário um parsing dos arquivos, buscando, selecionando e categorizando os valores e informações relevantes para a plataforma. A comunicação de entrada é caracterizada na Fig. 3.3 pelo número 6, tendo como partida o Vivado. Ao fim do processo do Vivado, temos os arquivos e informações necessários para início do trabalho com potência.

O comando de início do Powermapper exige apenas uma informação, o nome da aplicação alvo. O analisador de arquivos então busca os seguintes arquivos:

- Arquivo fonte original (.scala) da aplicação, com os parâmetros de DSE anotados
- Logs e Resultado Final da comunicação entre Spatial e Hypermapper
- Relatórios de Recursos e Potência pós implementação da aplicação pelo Vivado

O arquivo fonte original é enviado pelo caminho rotulado com 1 na Fig. 3.4, para que o Gerador de Arquivos tenha a descrição da aplicação que será necessária para sua operação. Os Logs e Resultado Final são utilizados para obter os pontos analisados pelo Hypermapper e a saída da fronteira de Pareto gerada. Essas informações seguem ambos os caminhos 6.1 e 6.2, uma vez que os pontos na fronteira serão utilizados na geração de arquivos, como também no treino do estimador. Por último temos os relatórios vindos do Vivado, as últimas informações a serem recebidas, uma vez que dependem do trabalho inicial do gerador de arquivos que será explicado na sequência. Essas informações seguem o caminho 6.2, sendo utilizadas para treino do Estimador.

3.2.2 Gerador de Arquivos

Com o arquivo fonte e os pontos analisados pelo Hypermapper recebidos do Analisador, se inicia o processo de Geração de Arquivos. Esses arquivos serão utilizados pelo Spatial para a geração de designs para o treinamento do Estimador de Potência. Assim, é necessário que alguns pontos sejam gerados e implementados pelo Vivado, para termos uma base de amostragem de treino. Para o nosso caso, foi definido um número de 10 pontos. Assim, são gerados 10 arquivos fonte (.scala) para o fluxo completo do Spatial. Esse valor pode ser elevado para tornar a estimação mais precisa, embora também alongue o processo total. Para a geração destes arquivos, são removidas as anotações de DSE da

aplicação original, colocando em cada arquivo um ponto do espaço de projeto analisado pelo Hypermapper.

Esses arquivos são enviados diretamente para compilação do Spatial, sem a flag do Hypermapper ligada, uma vez que esses arquivos não tem exploração de espaço de projeto. Dessa forma o ciclo ocorre sem a comunicação indicada com 2 na Fig. 3.3. Após a compilação e geração dos arquivos em HDL, o Vivado faz a síntese e implementação, obtendo assim os relatórios mencionados acima.

Após o procedimento de treino, estimação e comunicação com o Hypermapper que será explicado em sequência, temos a escolha de um novo ponto de design, que será aquele de maior eficiência energética estimada dentre os designs avaliados. Esse ponto é recebido pelo Gerador de Arquivos, que efetua o mesmo procedimento de criação de arquivo a partir do original, envio para o Spatial, passando pelo ciclo completo de geração, e assim gerando o bitstream final de saída.

3.2.3 Estimador de Potência

Para a avaliação e exploração das possibilidades de implementação em relação a potência foi necessário o desenvolvimento de um estimador de potência. Quanto a DSE, o Spatial tem seu foco em área e número de ciclos, buscando minimizar a área enquanto mantém um bom desempenho. Dessa forma, os modelos internos da ferramentas estimam apenas essas informações no momento da exploração de espaço de projeto. Uma vez que a informação relevante para este trabalho é eficiência energética, foi desenvolvido um estimador voltado para estimar potência. Utilizando a estimativa de potência, juntamente com a estimativa de desempenho, pode-se estimar a eficiência energética de cada design.

Nosso estimador tem como base os estudos apresentados em (LIN et al., 2020), utilizando os conceitos de coleção de dados e as características arquiteturais de potência apresentados. Estes conceitos avaliam os melhores parâmetros para estimação de potência, como também a melhor forma de observar as mudanças utilizando um fator de escala. Vale mencionar que nossa implementação não tem relação direta com a implementação do estimador deste trabalho, apenas utilizando como base estes conceitos.

A partir dos estudos, foram observadas quais seriam as entradas necessárias para o treinamento eficiente do estimador. Foram selecionados os seguintes parâmetros dos pontos analisados:

- O valor de latência em ciclos
- Os valores de diferentes recursos como:
    - Look Up Tables (LUTs)
    - Flip Flops (FFs)
    - Digital Signal Processors (DSPs)
    - Block Random Access Memory (BRAMs)

Essa combinação de atributos é relacionada com os parâmetros de DSE equivalentes de cada ponto no espaço de projeto, para que o estimador faça a correlação destes parâmetros com um resultado de potência.

Para este estimador, foi utilizado o algoritmo de classificação e regressão florestas randômicas, implementado com a utilização da biblioteca de IA SciKit-Learn (PEDREGOSA et al., 2011), para a predição dos valores de potência.

Após a geração e implementação dos arquivos, os valores obtidos dos relatórios são recebidos pelo estimador que faz o treinamento da floresta randômica. Com todos os pontos analisados e treinados pela nossa ferramenta, pode-se iniciar a comunicação com o Hypermapper, que será descrita na sequência. O estimador tem como objetivo, nessa próxima etapa, enviar os resultados em potência dos pontos que o Hypermapper selecionou para exploração.

3.2.4 Comunicação Hypermapper

Com o estimador treinado, inicia-se uma comunicação no modo cliente-servidor com o Hypermapper, nos mesmos moldes do Spatial. Todavia, neste caso, a métrica enviada para otimização na configuração do Hypermapper é potência, dessa forma, é feita a exploração de espaço de projeto buscando minimizar o seu uso.

O Hypermapper faz a requisição de 20 pontos para o nosso estimador, que devolve a informação de potência apenas. O Hypermapper inicia seu treinamento, e como já observado anteriormente, solicita 5 novos pontos, que são estimados em tempo de execução pelo nosso estimador, devolvendo os valores. Após algumas trocas de informações, o Hypermapper devolve o ponto com o melhor valor de potência associado, de acordo com seu algoritmo. Esse ponto é então repassado para o Gerador de Arquivos que pode implementar o design final do nosso projeto.

4 EXPERIMENTOS

Nessa seção serão discutidos as aplicações utilizadas e seus métodos de otimização, os experimentos realizados, as etapas seguidas e os resultados obtidos.

Para fins de comparação e avaliação, foram realizados dois tipos de experimentos. O primeiro visa observar o impacto da síntese de todos os pontos de exploração (ou seja, sem utilizar a estimação de potência) tanto no resultado em energia, como também em tempo de execução, comparado ao método original utilizado pelo Spatial e ao caminho do PowerMapper. O segundo experimento foca na comparação direta entre os resultados obtidos através da utilização do Hypermapper, voltado na otimização de área e desempenho, e os da nossa plataforma, voltada para potência e consequentemente energia.

Para todos os testes e experimentos foi utilizado um computador equipado com uma CPU Intel Core i7-6700 CPU @ 3.40GHz, memória DDR3 de 32 GB @ 1600MHz rodando Ubuntu 20.04. Para a síntese, implementação e geração de relatórios dos designs foi utilizado o Vivado 2020.1.

Foi utilizado nas simulações o FPGA Zedboard Zynq-7000 utilizando a frequência de 125MHz. Importante mencionar que esta é a frequência de operação padrão utilizada pelo Spatial. O consumo de energia também está atrelado a frequência do FPGA, mas está fora do escopo deste trabalho a analise do impacto desta métrica nos resultados de energia.

4.1 Desenvolvimento e Análise de Aplicações

Para observar os impactos da potência com os diferentes métodos de exploração de projeto definidos acima, foram desenvolvidas as aplicações de transferência de memória e o QuickSort. Visando aumentar os benchmarks as aplicações de Produto de Matrizes e do Kmeans foram adaptadas de exemplos do próprio Spatial. Todos os benchmarks foram desenvolvidos diretamente utilizando a linguagem Spatial, buscando uma variedade tanto de funcionalidade, como também de complexidade entre elas.

4.1.1 Transferência de Memória

Uma vez que sistemas baseados em FPGAs utilizam diferentes tipos de memória (e.g., SRAM, DRAM) com diferentes focos de utilização, é comum a transferência de dados entre esses componentes tanto para processamento, como para armazenamento. Foi desenvolvida uma aplicação simples de transferência entre DRAMs (mais lentas, maior volume) e SRAMs (mais rápidas, menor volume), utilizando buffers de passagem. As variáveis utilizadas para a exploração de projeto foram: o tamanho de cada bloco do buffer, a quantidade de blocos transferidos por ciclo entre componentes, e os parâmetros de redução. A operação de redução se trata da soma dos resultados de operações sobre uma única coleção de dados, onde a saída se trata do valor combinado final. As operações utilizadas dentro da redução podem ser paralelizadas quando independentes, ou otimizadas através de técnicas de pre-fetching. Em ambos os casos, o tamanho dos blocos de operações a serem feitas em conjunto pode ser definido, e no caso desta aplicação, otimizado através de DSE. A combinação de pontos gerados para esse design é de 768 pontos.

4.1.2 QuickSort

Aplicações de ordenação, com diversas operações de controle, geralmente não são interessantes para aceleração em hardware como FPGAs. No entanto, algoritmos mais simples como o QuickSort podem se beneficiar do paralelismo oferecido. Foi desenvolvido o algoritmo padrão do Quicksort com 1000 entradas, com a variável de exploração de projeto sendo o nível de paralelismo do loop de comparação de elementos. Para este design foram observados 1980 pontos de exploração.

4.1.3 Produto de Matrizes

Uma aplicação comum para utilização em FPGAs é a multiplicação de matrizes, uma vez que as operações independentes podem ser paralelizadas, utilizando todo o potencial da aceleração em hardware. Foi adaptada do repositório de testes do Spatial, uma aplicação de multiplicação de matrizes 100x100, onde as variáveis de exploração foram: a extensão da redução, a quantidade de operações em paralelo, e a utilização de pipelines. Foram gerados 3600 pontos de exploração de espaço de projeto para este design.

4.1.4 Kmeans

Algoritmos de aprendizado de máquina estão em crescente uso, com novas tecnologias e áreas de estudo. Um dos principais algoritmos de aprendizado não supervisionado é o Kmeans, com a proposta de agrupamento de dados a partir de características em comum. Foi adaptada do repositório de testes do Spatial, o algoritmo Kmeans, com 960.000 dados como entrada, buscando o agrupamento de 16 clusters em 64 épocas de teste. Todas as variáveis de ajuste dos centroides, como o nível do paralelismo das operações foram utilizas para exploração de projeto. Para este design foram gerados 32,768 pontos de exploração de espaço de projeto.

4.2 Propostas de análise do Espaço de Projeto

Uma aplicação que tem a utilização da exploração de espaço de projeto está buscando a melhor implementação voltada para alguma métrica desejada. Esse processo gera inúmeros pontos a serem analisados, aumentando junto com a complexidade da aplicação e o número de métricas de interesse. Ferramentas de DSE têm como objetivo acelerar esse procedimento de análise, uma vez que analisar todos os pontos pode ser custoso computacionalmente.

No entanto, para observar a eficácia energética do nosso estimador com o design mais eficiente possível, foram feitos dois experimentos distintos.

O primeiro faz a utilização da nossa plataforma, o Powermapper, sem o treinamento e predição do estimador, com todos os pontos analisados pelo Hypermapper sendo gerados e sintetizados sob demanda para definição do valor de potência. A comparação com a proposta do estimador visa observar a precisão do estimador, como também a diferença em tempo computacional.

O segundo experimento é a utilização da força bruta, gerando, sintetizando e implementando todos os pontos do espaço de projeto, buscando assim o melhor ponto em termos de potência. Com este experimento, podemos comparar a precisão de todas as outras propostas, como também avaliar os prós e contras de uma exploração completa do espaço de projeto.

Para este experimento foi utilizada a aplicação do QuickSort, com 1980 pontos para exploração. Primeiro foi sintetizada a aplicação pela versão original do Spatial, tendo os pontos analisados pelo Hypermapper, gerando na saída uma fronteira com 10

pontos. Todos esses designs foram sintetizados e implementados pelo Vivado, gerando os relatórios de utilização de recursos e potência. Para a versão Hypermapper (HM), foi selecionado, dentre esses, o design de menor consumo energético.

Para a versão Powermapper (PM) do experimento, foram utilizados os dados dos relatórios para o treinamento do estimador, sendo este utilizado para busca do design com menor potência junto ao Hypermapper como explicado na Seção 3. Este processo gerou o design final a ser sintetizado, passando pelo processo do Spatial, seguido da implementação no Vivado, retornando os relatórios para este experimento.

Para a versão do Powermapper sem estimador (PMSE) do experimento, nossa plataforma foi utilizada diretamente com o Hypermapper, mas com a lógica de geração de arquivos vindo diretamente da comunicação 7 da Fig. 3.3. Sem o treinamento e utilização do estimador, nossa plataforma se comunica diretamente com o Hypermapper, utilizando a mesma configuração de otimização de potência. O Hypermapper então, envia os primeiros 20 pontos que serão analisados, solicitando os valores de potência para nossa plataforma. Sem o estimador, nossa plataforma deve fazer o ciclo completo do Spatial para geração e implementação dos pontos de design requisitados. O gerador de arquivos elabora os arquivos fonte necessários, enviando-os para o Spatial, seguindo o caminho completo até a síntese e implementação pelo Vivado. Com o ciclo terminado para todos os pontos requeridos pelo Hypermapper, nossa plataforma pode obter dos relatórios do Vivado a informação de potência para os pontos, devolvendo o valor real para a exploração de espaço de projeto. Após a treinamento do Hypermapper, 5 novos pontos são requisitados para a avaliação, onde nossa plataforma reproduz o mesmo caminho já citado, devolvendo os valores de potência. Ao final desse processo, o Hypermapper tem como saída o ponto do design com melhor potência.

Para a versão Força Bruta (FB), foram discriminados todos os pontos de exploração de espaço de projeto, passando pela ferramenta de criação de arquivos da nossa plataforma, gerando o código fonte para todos os pontos. A partir dessa geração, foi aplicado o ciclo básico do Spatial, sem a utilização do Hypermapper, gerando os arquivos para o Vivado, que por fim sintetizou e implementou todos as possibilidades.

Na tabela 4.1 são mostrados os dados de potência e ciclos para os quatro tipos: Hypermapper (HM), Powermapper (PM), Powermapper sem Estimador (PMSE) e Força Bruta (FB).

Pode-se observar a diminuição da potência entre as quatro propostas. Como esperado, a versão de força bruta tem uma potência muito menor que as outras, sendo 30% que

Experimento	Potência	Ciclos
HM	4.962 W	245194
PM	4.629 W	251187
PMSE	4.524 W	251269
FB	3.463 W	278985

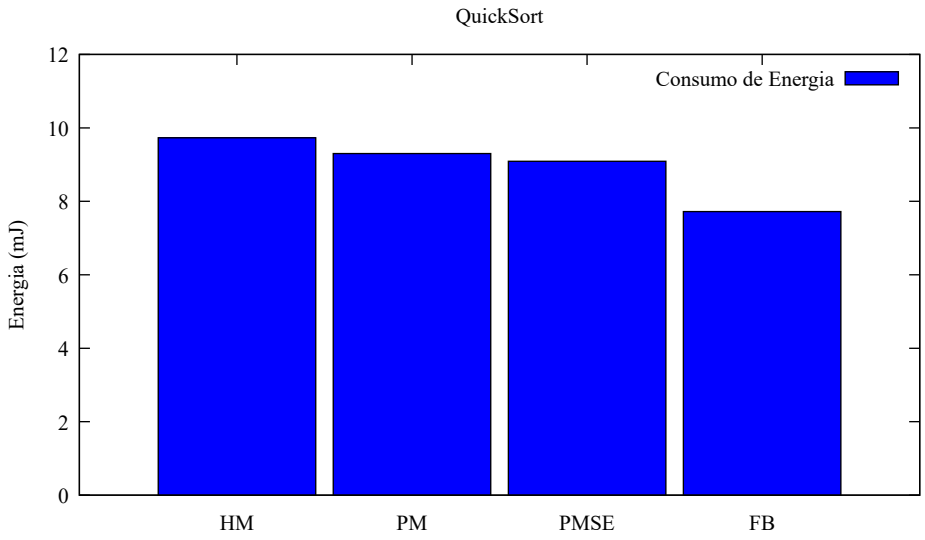
Tabela 4.1 – Potência e Ciclos - QuickSort

o valor original. O estimador focado em potência tem uma diminuição de 7% comparada com a original. Na mesma linha, o experimento da nossa plataforma sem o estimador teve uma diminuição de 8%.

Por outro lado pode ser observar um aumento no número de ciclos, o que implica em um desempenho pior, uma latência maior no processo. Tanto o design utilizado pelo estimador, como o do PMSE tem um aumento de 2% no número de ciclos, enquanto temos um aumento de 12% na proposta de força bruta.

Os valores de energia podem ser obtido a partir da potência, dos ciclos e da frequência utilizada para as analises, que nesse caso foi de 125MHz. Os resultados da eficiência energética podem ser observados no gráfico da Fig. 4.1.

Figura 4.1 – Eficiência Energética - QuickSort



Fonte: O Autor

A partir do gráfico, pode-se observar a diferença entre o consumo de energia entre as propostas, com o valor do Powermapper tendo 5% de diminuição, o Powermapper sem Estimador com 6% de diminuição e o design força bruta com 20%.

Foram obtidos também os valores de recursos utilizados para todos os quatro casos, buscando observar onde se encontram as diferenças de utilização atreladas ao consumo de energia. Para este experimento os recursos relevantes mostrados serão lookup

tables (LUTs), LUTRAMs, que englobam LUTs utilizadas como RAM distribuídas ou registradores de deslocamento, flip flops (FF) e blocos de RAM.

	Recursos				
Tipo	Total LUTs	Logic LUTs	LUTRAMs	FF	RAMB36
HM	1085(2.04%)	1081(2.03%)	4(0.02%)	887(0.83%)	1(0.71%)
PM	1022(1.92%)	1014(1.91%)	8(0.05%)	818(0.77%)	1(0.71%)
PMSE	1021(1.92%)	1017(1.91%)	4(0.02%)	808(0.76%)	1(0.71%)
FB	857(1.61%)	857(1.61%)	0(0.00%)	706(0.66%)	1(0.71%)

Tabela 4.2 – Recursos - QuickSort

A partir desses resultados, é possível observar uma diminuição do uso de todos os recursos com a troca de abordagem, o que afeta diretamente o consumo de potência no FPGA. Outro ponto importante levantado por esses resultados está no uso de recursos específicos, como pode ser visto na diferença entre o experimento HM, PM e PMSE.

No caso do PM em comparação com o HM, existe uma diminuição pequena entre os valores de LUTs e FFs, no entanto, existe um aumento de LUTRAMs. No caso do PMSE, existe uma diminuição de todos os recursos analisados, com a mesma utilização de LUTRAMs quando comparado ao HM. Ambos os experimento utilizando o Powermapper tem um menor consumo de potência, evidenciando que não apenas a diminuição do uso de recursos, como também a diferente utilização de recursos específicos pode contribuir na diminuição da potência.

Em suma, pode se analisar que os experimentos PM, PMSE e FB tem uma melhor eficiência energética, a partir de uma diminuição dos recursos, mas por outro lado, com um aumento no número de ciclos. É esperado que haja esse tipo de trade-off ao passo que a potência está ligada diretamente ao desempenho e o tipo dos recursos utilizados, a alteração em uma dessas métricas é sempre acompanhada pelas outras.

Observando esses resultados, e em especial o consumo de energia apresentado no gráfico da Fig. 4.1, o analisador força bruta pode parecer uma boa opção na busca de um design de baixo consumo. Uma vez que temos a análise completa dos pontos, é evidente que teremos o melhor design em relação a potência. Todavia, essa proposta demanda um tempo de computação muito maior, uma vez que analisa todos os designs.

Tomando como exemplo a aplicação do Quick Sort, o tempo para geração, síntese e implementação de um único design leva, em média 2min54s. Assim, podemos calcular o tempo computacional para as quatro propostas. Para o método Hypermapper, temos o ciclo de apenas um design, sendo assim o tempo computacional é de 2min 54s. Para o método Powermapper, é necessária a síntese de 10 designs para obter os valores de recursos

e potência para treino, sendo assim, leva o tempo de 29 min. Para o método Powermapper sem Estimador, é necessário a geração tanto dos 20 primeiros designs utilizados, como também os 5 adicionais requisitados pelo Hypermapper, chegando assim em 1h 12min 30s. Agora com a proposta de exploração força bruta, temos o ciclo de implementação para os 1980 pontos, resultando em 3 dias, 23h e 42min.

Dessa forma, o analisador completo está longe de ser escalável, ao passo que o ciclo com o Powermapper, mesmo que com uma diminuição menor em termos de potência, tem um tempo computacional aceitável e pode ser utilizado em design maiores.

Quanto à comparação do Powermapper com estimador (PM) e sem o estimador (PMSE), podemos observar uma grande proximidade. Isso indica que os resultados estimados estão adequados e levando a exploração do espaço de projeto no caminho correto da otimização do uso de potência. A diferença está no tempo computacional entre os dois experimentos.

Neste caso, com a utilização do número padrão de amostras do Hypermapper, que são 20, o experimento PMSE tem um pouco mais que o dobro do tempo computacional, uma vez que o Powermapper utiliza 10 experimentos para o treinamento, enquanto este experimento sintetiza 25 designs para obtenção da potência. Ainda assim, ambos os tempos são computacionalmente aceitáveis.

No entanto, uma vez que o número de amostras pode ser definido pelo programador, a relação entre ambos os experimentos pode mudar drasticamente. Para aumentar a precisão da exploração de espaço de projeto, ainda mais em projetos complexos com muitas combinações, podem ser utilizados mais pontos como base de treinamento do Hypermapper. Esse aumento não tem nenhum efeito no Powermapper utilizando o Estimador, uma vez que independente do número de pontos requeridos, são utilizados 10 designs para o treinamento. Por outro lado, o experimento PMSE necessitaria gerar e implementar o número de designs definidos, o que afeta a escalabilidade desta proposta.

4.3 Comparação PowerMapper e HyperMapper

Após comparar os resultados de uma múltiplas análises do espaço de projeto em relação a potência, seguimos para uma comparação direta entre o ciclo original do Spatial utilizando somente o Hypermapper e o nosso ciclo proposto, utilizando o Powermapper. As aplicações utilizadas serão explicadas na sequência, com uma variedade tanto de complexidade como de foco de uso, buscando assim observar as duas propostas em diferentes

casos. Essa comparação tem como objetivo validar nossa proposta com diferentes aplicações, além de analisar as diferenças de uso de recursos e desempenho em relação ao ciclo original. A seguir serão apresentados os resultados dos experimentos, seguindo a ordem de complexidade da aplicação: Transferência de memória, Multiplicação de Matrizes e Kmeans.

4.4 Transferência de Memória

Iniciando os resultados, temos a aplicação Transferência de Memória. A base deste experimento é a transferência de dados, com toda a lógica voltada para os buffers e operações de manipulação de dados.

Para obter os resultados foi seguida a metologia descrita na Seção 3, os valores atribuídos ao Hypermapper, foram obtidos através do ciclo comum do Spatial. A partir dos resultados deste ciclo, 10 pontos de design foram sintetizados e implementados pelo Vivado, buscando os valores de recursos e potência para treino do Powermapper. Após o treinamento, o espaço de projeto foi explorado observando a métrica de potência, o resultado nos experimentos é caracterizado como Powermapper.

O tempo de síntese e geração de um design desta aplicação é, em média, 3min 13s. Este é o tempo para a geração dos resultados no método Hypermapper original, para o Powermapper temos 32min 10s. Utilizando o Powermapper sem estimador, o tempo seria de 1h 20min 25s. Caso fosse utilizado o método de força bruta, o tempo computacional seria de 1d 17h 10min.

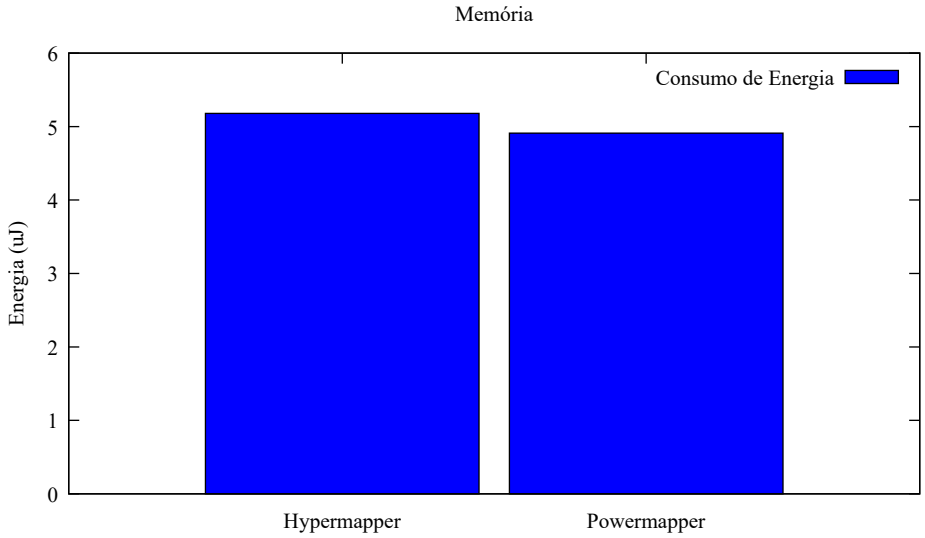
Os resultados quanto a potência e número de ciclos são apresentados na Tabela 4.3. Como pode ser observado, a potência obtida através do Powermapper tem uma diminuição de 9% em relação ao Hypermapper. Por outro lado temos um aumento de 5% no número de ciclos de operações.

Experimento	Tipo	Potência	Ciclos
Memória	Hypermapper	3.581 W	181
	Powermapper	3.236 W	190

Tabela 4.3 – Potência e Ciclos - Memória

A partir dos valores obtidos de potência e ciclos, utilizando a mesma frequência de 125MHz para os experimento obtemos o consumo de energia. Os resultados são apresentados na Fig. 4.2, mostrando uma diminuição de 5% no consumo de energia com a nossa proposta.

Figura 4.2 – Eficiência Energética - Transferência de Memória



Fonte: O Autor

Para observar a diferença completa entre as métricas, são reportados também os valores de uso de recursos, apresentados na Tabela 4.4. Para este experimento, as métricas mais relevantes são LUTs, LUTRAMs, Flip Flops e blocos de RAM. Como já observado no experimento do Quick Sort, temos uma redução no uso de recursos, e, neste caso, em todas as categorias apresentadas.

	Recursos					
Aplicação	Tipo	Total LUTs	Logic LUTs	LUTRAMs	FF	RAMB36
Memória	Hypermapper	949	949	0	833	1
	Powermapper	852	852	0	746	1

Tabela 4.4 – Recursos - Memória

4.4.1 Multiplicação de Matrizes

Aumentando a complexidade da lógica e das operações, aliada a maiores possibilidades de paralelismo, temos a aplicação Multiplicação de Matrizes.

O tempo de síntese e geração de um design desta aplicação é, em média, 4min 8s. Este é o tempo para a geração dos resultados no método Hypermapper original. Para o Powermapper temos 41m 20s. Utilizando o Powermapper sem estimador, o tempo seria de 1h 43min 20s. Caso fosse utilizado o método de Força Bruta, o tempo computacional seria de 10d 8h.

Os resultados quanto a potência e número de ciclos são apresentados na Tabela

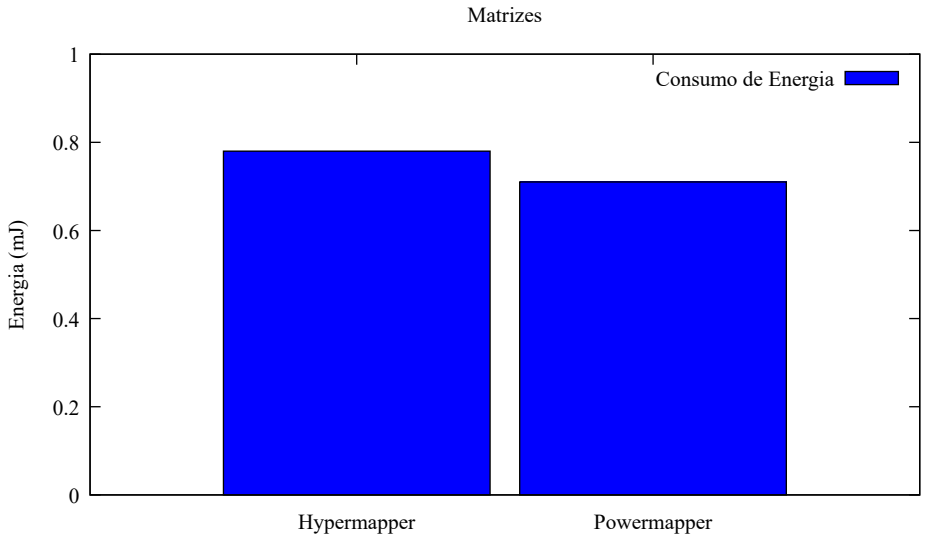
4.5. Como pode ser observado, a potência obtida através do Powermapper tem uma diminuição de 7% em relação ao original. Neste experimento temos uma diminuição de 2% no número de ciclos.

Experimento	Tipo	Potência	Ciclos
Matrizes	Hypermapper	16.130 W	6080
	Powermapper	15.05 W	5977

Tabela 4.5 – Potência e Ciclos - Matrizes

A partir dos valores obtidos de potência e ciclos, utilizando a mesma frequência de 125MHz para o experimento, obtemos o consumo de energia. Os resultados são apresentados na Fig. 4.3, mostrando uma diminuição de 9% no consumo de energia com a nossa proposta.

Figura 4.3 – Eficiência Energética - Multiplicação de Matrizes



Fonte: O Autor

Para observar a diferença completa entre as métricas, são reportados também os valores de uso de recursos, apresentados na Tabela 4.6. Para este experimento, as métricas mais relevantes são LUTs, SRLs, que são registradores de deslocamento de tamanho variável, Flip Flops, blocos de RAM e DSPs, processadores digitais de sinal. Este experimento tem uma diminuição bem menos expressiva se comparado aos outros experimentos, com exceção do uso de DSPs, que foi reduzido pela metade.

	Recursos						
Aplicação	Tipo	Logic LUTs	SRLs	FFs	RAMB36	RAMB18	DSP
Matrizes	Hypermapper	10146	219	9520	54	96	72
	Powermapper	11766	211	8439	54	96	36

Tabela 4.6 – Recursos - Matrizes

4.5 Kmeans

Um dos experimentos de maior complexidade é o Kmeans, com uma lógica bem mais complexa atrelada ao aprendizado de máquina e as diversas iterações necessárias na tomada de decisão.

O tempo de síntese e geração de um design desta aplicação leva 5min 3s, que é o tempo para a geração dos resultados no método Hypermapper original. Para o Powermapper temos 50min 30s. Utilizando o Powermapper sem estimador, o tempo seria de 2h 6min 15s. Caso fosse utilizado o método Força Bruta, o tempo computacional seria de 114d 21h 58m.

Os resultados quanto a potência e número de ciclos são apresentados na Tabela 4.7. Como pode ser observado, a potência obtida através do estimador tem uma diminuição de 20% em relação ao original. Neste experimento temos uma aumento de 0.05% no número de ciclos.

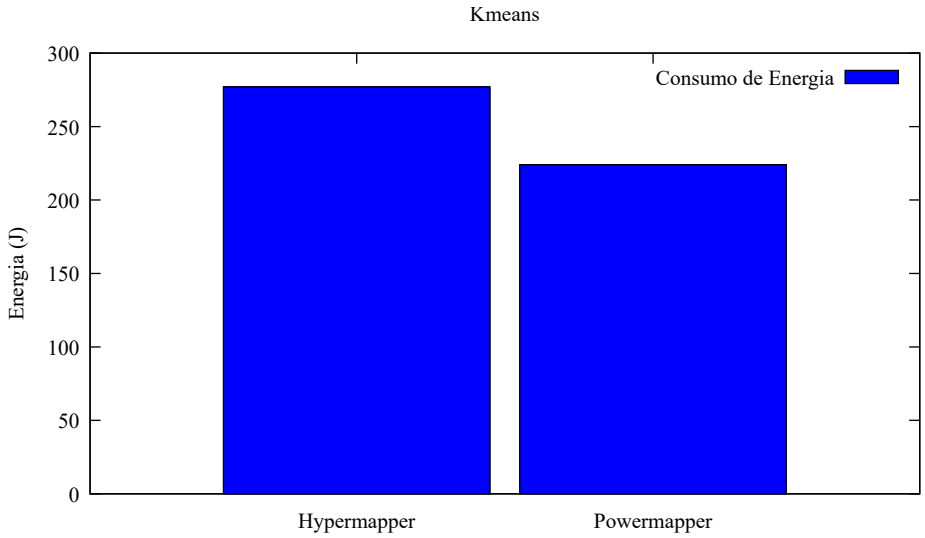
Experimento	Tipo	Potência	Ciclos
Kmeans	Hypermapper	69.49 W	503,043,011
	Powermapper	55.77 W	503,043,267

Tabela 4.7 – Potência e Ciclos - Kmeans

A partir dos valores obtidos de potência e ciclos, utilizando a mesma frequência de 125MHz para os experimentos, obtemos o consumo de energia. Os resultados são apresentados na Fig. 4.3, mostrando uma diminuição de 20% no consumo de energia com a nossa proposta.

Para observar a diferença completa entre as métricas, são reportados também os valores de uso de recursos, apresentados na Tabela 4.8. Para este experimento, as métricas mais relevantes são LUTs, SRLs, Flip Flops, blocos de RAM e DSPs. Este experimento tem uma maior redução no uso dos recursos, e uma maior utilização apenas em DSPs.

Figura 4.4 – Eficiência Energética - Kmeans



Fonte: O Autor

	Recursos						
Aplicação	Tipo	Logic LUTs	LUTRAMs	SRLs	FFs	RAM	DSP
Kmeans	Hypermapper	46735	1920	963	78680	40	0
	Powermapper	43355	1920	355	71498	29	6

Tabela 4.8 – Recursos - Kmeans

4.6 Discussão

Acima foram expostos os resultados individuais dos nossos experimentos, tanto da análise de diferentes propostas de exploração de projeto, como também a comparação da nossa plataforma com a proposta original do Spatial.

Para melhor visualização e discussão dos experimentos, os resultados foram agregados em gráficos sumarizados, mostrando as diferenças entre os experimentos.

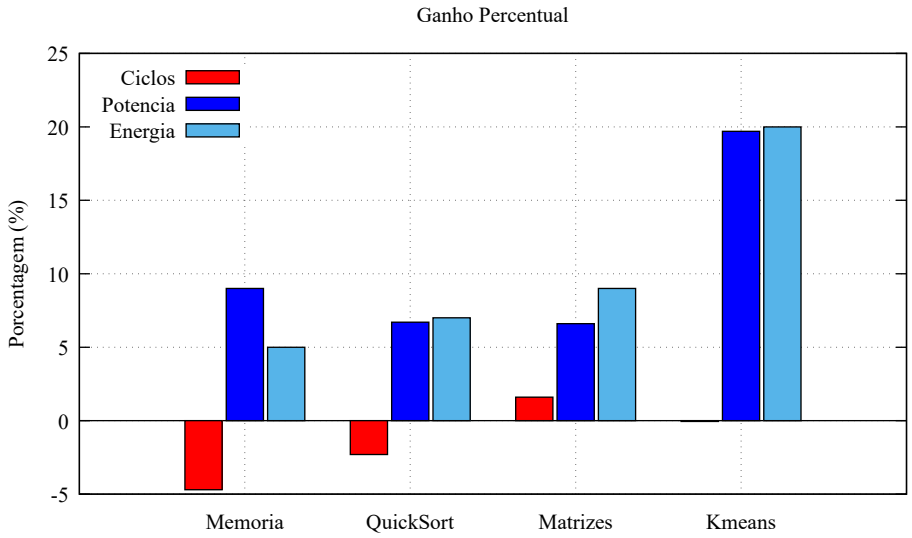
As principais métricas do nosso estudo são potência, ciclos e recursos, com o objetivo sendo diminuição no consumo de energia. A Fig. 4.6 contém a sumarização de todos os experimentos, mostrando o ganho percentual nas métricas de ciclos, potência e energia. Por outro lado, na Tabela A.1, localizada no apêndice, pode ser observada a junção das tabelas apresentadas acima na Seção 4.

Nossa plataforma conseguiu, em todos os experimentos, diminuir o consumo de energia, variando de 5% até 20% nessa redução. Mesmo com o aumento da complexidade, e por consequência o número de pontos, foi possível manter uma melhora na eficiência energética em comparação ao método original.

Uma vez que foi observada a diminuição na potência, é necessário analisar as mé-

tricas que se correlacionam, observando o trade-off gerado na busca da eficiência energética. Como observado existe uma perda percentual em desempenho, com o ciclos aumentando de 0.05% até 5%, tendo ganho apenas no experimento de matrizes, de 1.6%. Na questão de potência, tivemos ganhos em todos os experimentos, variando de 7% até 20%. Com essa sumarização, é possível observar a relação de ganho entre métricas e os trade-offs entre elas.

Figura 4.5 – Ganho Percentual



Fonte: O Autor

Essas informações mostram a relação entre recursos, potência e desempenho de perto, ao passo que a flutuação de uma destas métricas causa a variação das outras. Com a diminuição da potência, foi possível observar dois efeitos principais: alteração no número de ciclos, e variação no uso e quantidade de recursos.

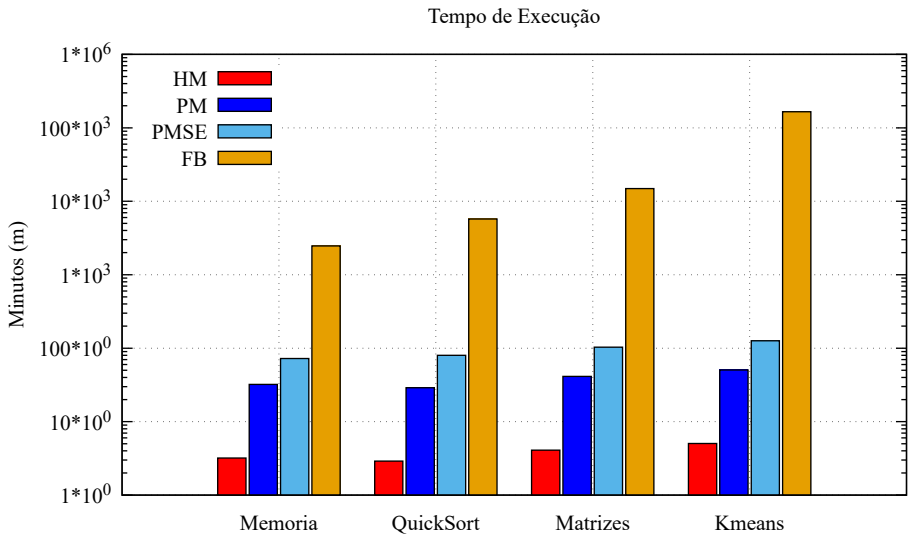
O efeito mais comum nos experimentos foi a diminuição de recursos, aliada a um aumento no número de ciclos, podendo ser observada nas aplicações: Memória, QuickSort e Kmeans. Importante observar que não houve somente a diminuição, mas como também a variação no uso de recursos, mostrando que diminuição da potência não está somente em usar menos, mas também em usar melhor os recursos disponíveis. Todavia, essa diminuição/variação causa, nesses casos, um pior desempenho da aplicação.

O outro efeito observado foi a diminuição no número de ciclos, com variação no uso de recursos, sendo visível na aplicação de Multiplicação de Matrizes. Este experimento mostra o outro lado da diminuição do uso de potência, uma vez que a utilização de recursos não tem uma diminuição expressiva, no entanto temos uma diminuição no número de ciclos. Mesmo que não haja uma diminuição na potência dissipada pelos re-

cursos, menos chaveamentos e transições impactam a potência e o consumo de energia positivamente, podendo ser observado de perto nesse caso.

Por último foram sumarizados os tempos de compilação necessários para cada um dos experimentos, levando em conta o tempo de síntese de um único design e fazendo a extrapolação para as outras três propostas. O resultado das comparações pode ser observadas na Fig. 4.6. Importante observar a disparidade das propostas, em escala logarítmica. Pode-se observar a diferenciação entre as propostas HM e PM, tendo relativas variações mesmo com o aumento da complexidade e dos pontos. Entre as propostas PM e PMSE, como esperado, temos uma relação dobrada entre elas como explicada na seção 4.2, podendo ser observada neste gráfico. A proposta FB, por outro lado, aumenta desproporcionalmente, evidenciando a falta de escalabilidade de um sistema sem otimizações na escolha.

Figura 4.6 – Tempo de Execução Sumarizado



Fonte: O Autor

5 CONSIDERAÇÕES FINAIS

Este trabalho propõe uma plataforma de exploração de espaço de projeto alinhada aos conceitos de HLS e DSL, tendo seu foco na métrica de potência, com objetivo de aumentar a eficiência energética nos designs avaliados.

Foram apresentados os módulos pertencentes a essa plataforma, sendo eles, um estimador de potência e módulos de integração tanto com o Vivado, como com o ciclo original do Spatial, adicionando a otimização em potência em um fluxo já estabelecido de HLS e DSE.

Foram apresentados também aplicações e experimentos com a utilização da nossa plataforma, observando seu desempenho em comparação com o ciclo original. Foi reportado nesses experimentos o uso de potência, eficiência energética, uso de recursos, desempenho e tempo de computação, avaliando todas essas métricas entre as propostas.

Seguido destes experimentos, foi possível observar um ganho em consumo de energia de 5% a 20% com o uso da nossa plataforma.

Através destes estudos e experimentos foi possível avaliar e demonstrar a importância da incorporação da métrica de potência nos ciclos de exploração de espaço de projeto. Em um momento em que HLS e DSE vêm ganhando ampla utilização, verificar as melhores implementações e as melhores métricas para orientar essa exploração é de suma relevância. Além disso, foi apresentada a análise completa dos trade-offs que são gerados ao otimizar o consumo de potência em relação a uso de recursos e desempenho. Por outro lado, foram observadas diferentes propostas de exploração de espaço de projeto, analisando os prós e contras do uso de estimadores e da exploração com força bruta.

Por fim, observando os experimentos de força bruta e de nossa plataforma sem o estimador, é possível observar uma margem de melhora de nossa implementação para alguns designs. Como trabalho futuro temos a diminuição dessa margem, através do aprimoramento do estimador, utilizando técnicas mais avançadas de aprendizado de máquina, como também a expansão dos parâmetros de treino. Outro ponto para melhorias são as heurísticas de DSE utilizadas, melhorando a busca do espaço de projeto.

REFERÊNCIAS

AMARA, A.; AMIEL, F.; EA, T. Fpga vs. asic for low power applications. **Microelectronics Journal**, v. 37, n. 8, p. 669–677, 2006. ISSN 0026-2692. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0026269205003927>.

ANDERSON, J.; NAJM, F. Power estimation techniques for fpgas. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 12, n. 10, p. 1015–1027, 2004.

BACHRACH, J. et al. Chisel: Constructing hardware in a scala embedded language. In: **DAC Design Automation Conference 2012**. [S.l.: s.n.], 2012. p. 1212–1221.

BREBNER, G. Packets everywhere: The great opportunity for field programmable technology. In: **2009 International Conference on Field-Programmable Technology**. [S.l.: s.n.], 2009. p. 1–10.

CALHOUN, B. H.; HONORE, F. A.; CHANDRAKASAN, A. Design methodology for fine-grained leakage control in mtcmos. In: **Proceedings of the 2003 International Symposium on Low Power Electronics and Design**. New York, NY, USA: Association for Computing Machinery, 2003. (ISLPED '03), p. 104–109. ISBN 158113682X. Available from Internet: <https://doi.org/10.1145/871506.871535>.

CHEN, D. et al. High-level power estimation and low-power design space exploration for fpgas. In: **2007 Asia and South Pacific Design Automation Conference**. [S.l.: s.n.], 2007. p. 529–534.

CHEN, D.; CONG, J.; PAN, P. **FPGA Design Automation: A Survey**. [S.l.: s.n.], 2006.

CHOI, Y.-k. et al. Hlscope+: Fast and accurate performance estimation for fpga hls. In: **Proceedings of the 36th International Conference on Computer-Aided Design**. [S.l.]: IEEE Press, 2017. (ICCAD '17), p. 691–698.

DEGALAHAL, V.; TUAN, T. Methodology for high level estimation of fpga power consumption. In: **Proceedings of the 2005 Asia and South Pacific Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2005. (ASP-DAC '05), p. 657–660. ISBN 0780387376. Available from Internet: <https://doi.org/10.1145/1120725.1120986>.

FLYNN, D. et al. **Low power methodology manual: for system-on-chip design**. [S.l.]: Springer Science & Business Media, 2007.

GAYASEN, A. et al. Reducing leakage energy in fpgas using region-constrained placement. In: . New York, NY, USA: Association for Computing Machinery, 2004. (FPGA '04), p. 51–58. ISBN 1581138296. Available from Internet: <https://doi.org/10.1145/968280.968289>.

INTEL. **User Guide - HLS 2003**. 2021. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/hls/ug-hls.pdf>. Accessed: 2021-10-23.

KAPRE, N.; BAYLISS, S. Survey of domain-specific languages for fpga computing. In: **2016 26th International Conference on Field Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2016. p. 1–12.

KOEPLINGER, D. et al. Spatial: A language and compiler for application accelerators. **SIGPLAN Not.**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, p. 296–311, jun. 2018. ISSN 0362-1340. Available from Internet: <https://doi.org/10.1145/3296979.3192379>.

KOEPLINGER, D. et al. Automatic generation of efficient accelerators for reconfigurable hardware. In: **2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)**. [S.l.: s.n.], 2016. p. 115–127.

KULKARNI, C.; BREBNER, G.; SCHELLE, G. Mapping a domain specific language to a platform fpga. In: **Proceedings of the 41st Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2004. (DAC '04), p. 924–927. ISBN 1581138288. Available from Internet: <https://doi.org/10.1145/996566.996811>.

LAHTI, S. et al. Are we there yet? a study on the state of high-level synthesis. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 38, n. 5, p. 898–911, 2019.

LAKSHMINARAYANA, A.; AHUJA, S.; SHUKLA, S. High level power estimation models for fpgas. In: **2011 IEEE Computer Society Annual Symposium on VLSI**. [S.l.: s.n.], 2011. p. 7–12.

LIGNATI, B. N. et al. Exploiting hls-generated multi-version kernels to improve cpu-fpga cloud systems. In: **2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2021. p. 536–541.

LIN, Z. et al. Hl-pow: A learning-based power modeling framework for high-level synthesis. In: **2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2020. p. 574–580.

LIU, D.; SCHAFER, B. C. Efficient and reliable high-level synthesis design space explorer for fpgas. In: **2016 26th International Conference on Field Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2016. p. 1–8.

LIU, H.-Y.; CARLONI, L. P. On learning-based methods for design-space exploration with high-level synthesis. In: **Proceedings of the 50th Annual Design Automation Conference**. New York, NY, USA: Association for Computing Machinery, 2013. (DAC '13). ISBN 9781450320719. Available from Internet: <https://doi.org/10.1145/2463209.2488795>.

LIU, S.; LAU, F. C.; SCHAFER, B. C. Accelerating fpga prototyping through predictive model-based hls design space exploration. In: **2019 56th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2019. p. 1–6.

NANE, R. et al. A survey and evaluation of fpga high-level synthesis tools. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 35, n. 10, p. 1591–1604, 2016.

NARDI, L.; KOEPLINGER, D.; OLUKOTUN, K. Practical design space exploration. In: **2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)**. [S.l.: s.n.], 2019. p. 347–358.

O'NEAL, K.; BRISK, P. Predictive modeling for cpu, gpu, and fpga performance and power consumption: A survey. In: **2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [S.l.: s.n.], 2018. p. 763–768.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PIMENTEL, A. D. Exploring exploration: A tutorial introduction to embedded systems design space exploration. **IEEE Design Test**, v. 34, n. 1, p. 77–90, 2017.

SCHAFER, B. C.; WANG, Z. High-level synthesis design space exploration: Past, present, and future. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 39, n. 10, p. 2628–2639, 2020.

SHAHSHAHANI, M. et al. A framework for modeling, optimizing, and implementing dnns on fpga using hls. In: **2020 IEEE 14th Dallas Circuits and Systems Conference (DCAS)**. [S.l.: s.n.], 2020. p. 1–6.

SIRACUSA, M. et al. Automated design space exploration and roofline analysis for fpga-based hls applications. In: **2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)**. [S.l.: s.n.], 2019. p. 314–314.

SUN, F. et al. Survey of fpga low power design. In: **2010 International Conference on Intelligent Control and Information Processing**. [S.l.: s.n.], 2010. p. 547–550.

SUNWOO, D. et al. Presto: An fpga-accelerated power estimation methodology for complex systems. In: **2010 International Conference on Field Programmable Logic and Applications**. [S.l.: s.n.], 2010. p. 310–317.

SYLVESTER, D.; KAUL, H. Future performance challenges in nanometer design. In: **Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)**. [S.l.: s.n.], 2001. p. 3–8.

WANG, L. et al. FPGA dynamic power minimization through placement and routing constraints. Springer Science and Business Media LLC, v. 2006, p. 1–10, 2006. Available from Internet: <https://doi.org/10.1155/es/2006/31605>.

XILINX. **User Guide 902**. 2021. <https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug902-vivado-high-level-synthesis.pdf>. Accessed: 2021-10-23.

ZHONG, G. et al. Design space exploration of fpga-based accelerators with multi-level parallelism. In: **Design, Automation Test in Europe Conference Exhibition (DATE), 2017**. [S.l.: s.n.], 2017. p. 1141–1146.

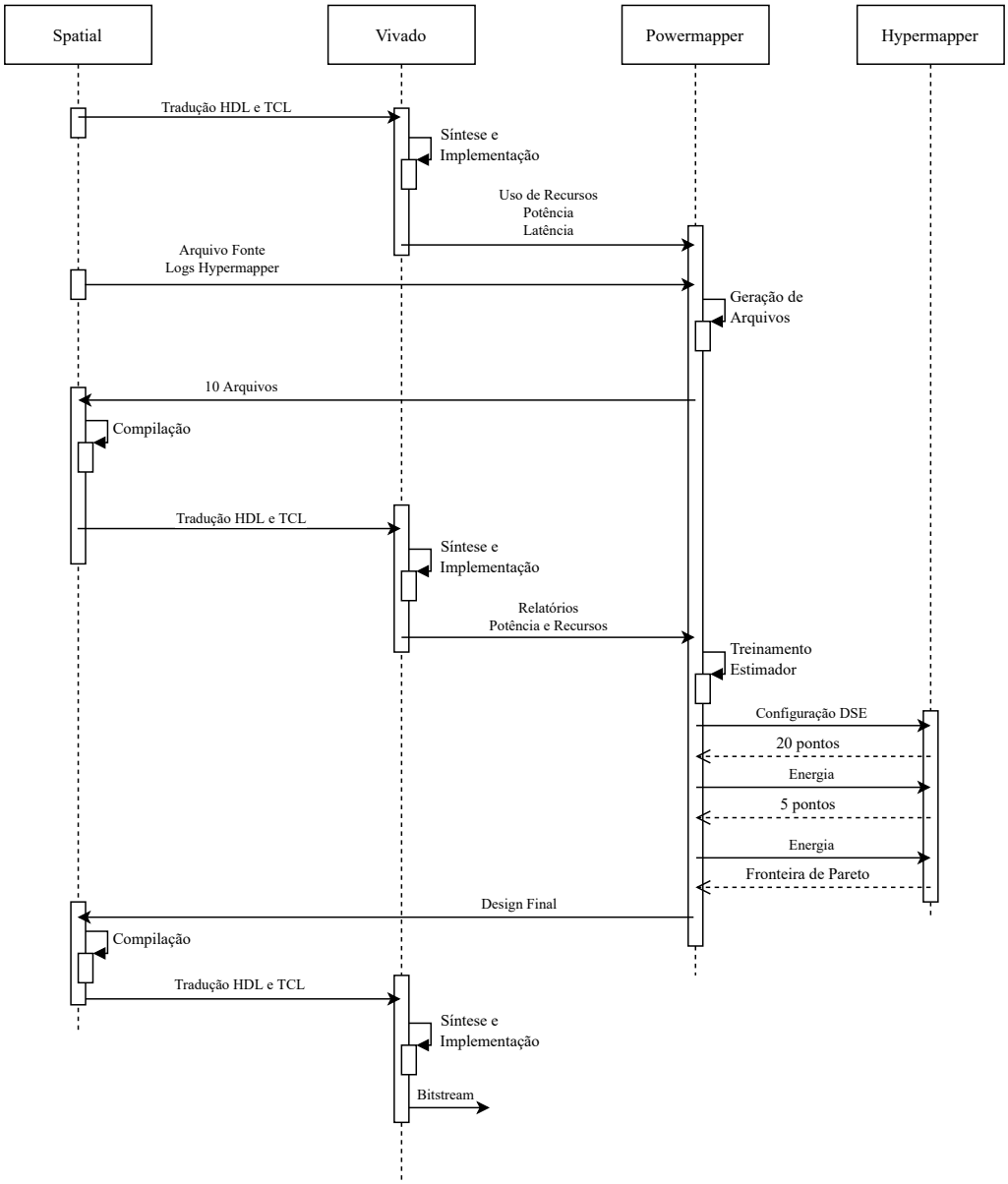
APÊNDICE A — RECURSOS

	Recursos						
Aplicação	Tipo	Logic LUTs	LUTRAMs	SRLs	FFs	RAM	DSP
Memória	Hypermapper	949	0	0	833	1	0
	Powermapper	852	0	0	746	1	0
QuickSort	Hypermapper	1081	4	0	887	1	0
	Powermapper	1014	8	0	818	1	0
Matrizes	Hypermapper	10146	0	219	9520	150	72
	Powermapper	11766	0	211	8439	150	36
Kmeans	Hypermapper	46735	1920	963	78680	40	0
	Powermapper	43355	1920	355	71498	29	6

Tabela A.1 – Recursos Sumarizados

APÊNDICE B — DIAGRAMA DE SEQUÊNCIA

Figura B.1 – Diagrama de Sequência Powermapper



Fonte: O Autor

APÊNDICE C — QUICKSORT - LINGUAGEM SPATIAL

```
@spatial object Q_sort extends SpatialApp {
  def main(args: Array[String]): Unit = {
    type X = FixPt[TRUE,_32,_0]
    val N = 1000
    val a = Array.fill(N){ random[X](16) }
    sort(a)
  }

  def sort[T:Num](aIn: Array[T]): Void = {
    val size = 1000
    val parallel = 10 (10 -> 1000)
    val a = DRAM[T](size)
    setMem(a, aIn)

    Accel {
      val aBlk = SRAM[T](size)
      val sortBlk = SRAM[T](size)
      aBlk load a(0::size par parallel)
      Foreach(size by 1){j =>
        Stream.Foreach(size by 1){i =>
          if(aBlk(i) < aBlk(i + 1)){
            sortBlk(i) = aBlk(i + 1)
          }
          else{
            sortBlk(i) = aBlk(i)
          }
        }
      }
    }
  }
}
```