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

BERNARDO NEUHAUS LIGNATI

MultiVers - Exploração Dinâmica de Espaço de Projeto para Sistemas CPU-FPGA em Cloud Utilizando Síntese de Alto Nível

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

Orientador: Prof. Dr. Antonio Carlos Schneider Beck Filho

Porto Alegre
2021

“‘Prefer knowledge to wealth, for the one is transitory, the other perpetual.” ”

— SOCRATES

“Prefer knowledge to grades and formatting.”

— SELF

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me dado a resiliência e curiosidade para realizar este curso de engenharia.

Agradeço aos meus pais José e Juliana e ao meu irmão Bruno por terem sido minha maior rede de apoio.

Aos meus amigos fica também meu agradecimento, principalmente àqueles da engenharia e computação da UFRGS e da TU Kaiserslautern que compartilharam os diversos bons e maus momentos durante a jornada. Aqueles que me acompanharam desde o começo até o fim, espero mais que todos que se tornem amigos para vida. Citar nomes aqui seria uma injustiça, mas a todos aqueles agradecidos deixo o convite para tomar uma bebida.

Agradeço também aos meus colegas de LSE que foram muito importantes para meu desenvolvimento técnico e cientifico, ajudando um colega menos experiente sempre que possível.

Ao professor Antônio "Caco"Beck, muito obrigado por ter me orientado na minha primeira experiência cientifica nesses últimos anos e por nos últimos meses ter sido um bom amigo.

RESUMO

Sistemas de servidores em *Cloud* têm mostrado cada vez mais importância comercial dentro do mundo da computação. Estes vêm explorando sistemas de execução colaborativa CPU-FPGA onde múltiplos clientes compartilham a mesma infraestrutura para maximizar a eficiência energética e escalabilidade, além de, em alguns casos, aumentar a qualidade de serviço percebida pelo usuário. Porém, o fornecimento de recursos é um desafio nestes ambientes, pois *Kernels* podem ser despachados para ambos, CPU e FPGA, concorrentemente numa grande variabilidade de cenários em termos de disponibilidade de recursos e características de carga de trabalho. Este trabalho primeiramente realiza experimentos para analisar a amplitude desse espaço de exploração e como diferentes versões de um mesmo *Kernel* acabam gerando melhores resultados, dependendo dos parâmetros escolhidos para avaliação.

Decorrente desta análise, é proposto ***Multivers***, uma *Framework* que aproveita o método de Síntese de Alto Nível para permitir maiores ganhos em tais sistemas colaborativos CPU-FPGA. ***Multivers*** explora vantagens da geração automática por Síntese de Alto Nível para produzir diferentes versões de cada requisição de entrada para *Kernels*, aumentando significativamente a exploração do espaço de projeto disponível e passível de otimização pelas estratégias de alocação do provedor de *Cloud*. Além de possuir a biblioteca gerada, que permite uma seleção de *Kernels* a partir de uma interface que permite a comunicação das necessidades atuais do Servidor de *Cloud*, ***Multivers*** também permite que o multi versionamento de *Kernels* e as estratégias de alocação trabalharem juntos, permitindo ajuste fino em termos de utilização de recursos, performance e energia; ou qualquer combinação destes parâmetros. A eficiência de ***Multivers*** é mostrada usando cenários de vida real de requisições de *Cloud*, compostos de uma diversidade de benchmarks e avaliando diferentes frações de FPGA disponíveis via regiões de reconfiguração parcial. Assim, atingindo um melhora média em *makespan* e energia de até 4.62× e 19.04×, respectivamente, sobre estratégias de alocação tradicional executando com *Kernels* não otimizados.

Palavras-chave: Execução Colaborativa. CPU-FPGA. Energia. HLS. Makespan.

MultiVers **- Using HLS for CPU-FPGA Cloud Dynamic Design Space Exploration**

ABSTRACT

Cloud server systems have been growing in commercial importance in the computational field. These systems have been exploring CPU-FPGA collaborative systems in which multiple clients share the same infrastructure to maximize the energy efficiency and scalability and, in some cases, increase the quality of service perceived by the users. However, the providing of resources is a challenge in these environments. The Kernels can be dispatched to run in both CPU and FPGA, concurrently generating a great variety of scenarios in terms of resources and workload characteristics. This work is composed first by experiments to analyze the amplitude of this design space and how different versions of the same Kernel generate different results, according to the parameters chosen to be evaluated.

From this analysis, ***Multivers*** is proposed, a Framework that leverages High-Level Synthesis to allow higher gains in such CPU-FPGA collaborative systems. ***Multivers*** exploits advantages of automatic High-Level Synthesis to generate different versions of each input Kernel request, greatly enlarging the available design space exploration passive of optimization by the allocation strategies in the cloud provider. Besides containing a generated library that allows the selection of Kernels to fit the Cloud provider's current needs, ***Multivers*** makes both kernel multi-versioning and allocation strategy work symbiotically, allowing fine-tuning in terms of resource usage, performance, energy, or any combination of these parameters.The efficiency of **Multivers** is shown using real-life scenarios of Cloud requisitions, composed of a variety of benchmarks and evaluating different fractions of available FPGA in Partial Reconfigurable Regions. This way, achieving average improvements on makespan and energy of up to 4.62× and 19.04×, respectively, over traditional allocation strategies executing non-optimized kernels.

Keywords: collaborative execution, CPU-FPGA, energy, HLS, makespan.

LISTA DE ABREVIATURAS E SIGLAS

ADI	Alternating Direction Implicit solver
B.FCFS	Baseline First Come Fisrt Served
B.GMK-E	Baseline Genetic Multidimensional Knapsack Energy
B.GMK-M	Baseline Genetic Multidimensional Knapsack Makespan
BRAM	Block Random Access Memory
DSP	Digital Signal Processor
FCFS	First Come Fisrt Served
FF	Flip-Flop
GMK	Genetic Multidimensional Knapsack
GMK-E	Genetic Multidimensional Knapsack Energy
GMK-M	Genetic Multidimensional Knapsack Makespan
HDL	Hardware Description Language - Linguagen de Descrição de Hardware
HLS	High-Level Sythesis - Síntese de Alto Nível
K.FCFS	Kernel baseline First Come Fisrt Served
LUT	Look-Up Table
MD5	Message Digest 5
MV	Multiverse Scenario

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Por muitos anos, a melhora no desempenho computacional foi governada pela Lei de Moore, que previa que os dispositivos dobravam sua performance em razão de duas vezes, devido a melhorias no processo de fabricação de dispositivos semicondutores. Com o fim da Lei de Moore, que veio devido às limitações físicas que impediram que os processos de fabricação continuassem a evoluir no mesmo ritmo, novas características arquitetônicas, voltadas ao paralelismo, começaram a ser exploradas, para conseguir novamente se atingir melhora na velocidade de computação. Porém, essa exploração de paralelismo usando processadores tradicionais enfrentam limitações bem definidas pelas Leis de Amdahl e Gustafson (GUSTAFSON, 1988).

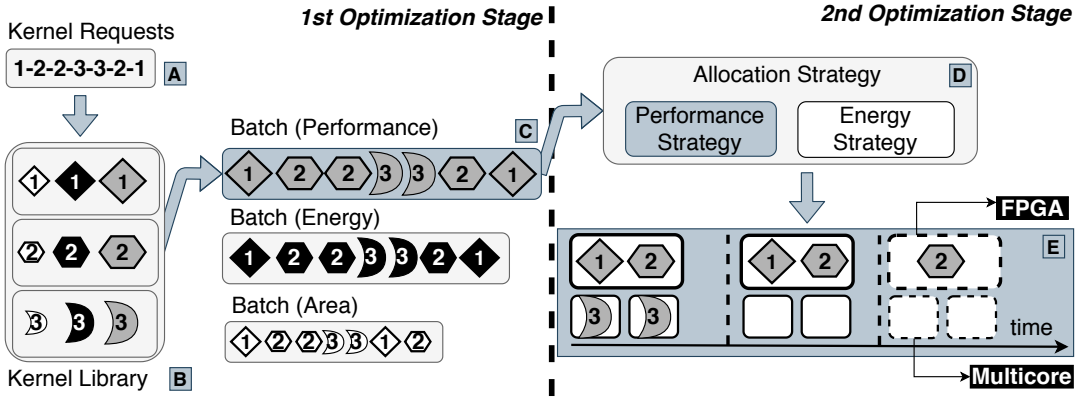
Assim, surgiram estratégias utilizando dispositivos não convencionais para buscar melhora na performance dos sistemas. Nestes, para algumas aplicações, as computações não executam mais em um processador de organização e arquitetura genéricas, mas são despachadas para executarem em um circuito digital de arquitetura específica que, em troca da perda de generalidade, consegue gerar melhores resultados em performance. Para que se possa implementar esses circuitos, tecnologias que permitem a implementação de circuitos reconfiguráveis como as FPGAs (capítulo 2, seção 2.1) acabam sendo uma ótima opção, pois além de apresentarem melhora no desempenho para algumas aplicações, para outras conseguem reduzir o consumo energético, outro ponto fundamental na computação moderna.

Em paralelo a isto, os sistemas de computação em larga escala, como servidores de Cloud (capítulo 2, seção 2.2), vêm apresentando a necessidade da utilização das estratégias de aceleradores específicos, como por exemplo para execução de operações matemáticas complexas e de redes neurais: devido à alta demanda de processamento, estes se beneficiam dos aceleradores para conseguir entregar uma melhor qualidade de serviço ou para reduzir gastos energéticos que são consideráveis nesse tipo de sistema.

Assim, os aceleradores de propósito específico baseados em FPGA têm se tornado cada vez mais viáveis, pois a indústria tem migrado de um fluxo tradicional que tem um alto *time to market* para um modelo de desenvolvimento mais moderno, baseado em HLS (Síntese de Alto Nível)(capítulo 2, seção 2.3). Com HLS, o circuito não é mais descrito fisicamente como em fluxos de HDL (Linguagem de Descrição de Hardware) tradicionais, mas compilado a partir de uma descrição algorítmica usando linguagens de alto nível.

Surge, a partir desse novo modelo de desenvolvimento baseado em HLS, um pro-

Figura 1.1: Visão Geral da Ideia Funcionamento da Framework



blema aberto à exploração de espaço de projeto (capítulo 4). Neste, através de uma mesma descrição algorítmica, pode-se gerar diferentes arquiteturas para um mesmo acelerador. Estas diferenciadas por anotações de código que permitem a exploração de diferentes técnicas de desenvolvimento de circuitos digitais com alta velocidade de projeto.

Neste trabalho foi desenvolvido uma Framework (capítulo 5), que faz esta exploração de forma eficiente baseada na geração de diferentes aceleradores de forma off-line, e a seleção de forma on-line dinamicamente, baseado no estado atual do sistema, podendo orientar os resultados para melhorar o gasto energético, a performance do sistema, a utilização de recursos, ou ainda uma combinação dos três, uma visão geral é expicitada na Figura 1.1.O *Framework* ainda faz uma análise de execução colaborativa, analisando quais computações se beneficiam mais da execução em um circuito digital específico em comparação com um processador de propósito geral, escalonando assim as tarefas em ambos e atingindo uma melhora no desempenho energético de até $19\times$ e no desempenho de até $4.62\times$.

2 BACKGROUND

Neste capítulo será discutida a fundamentação teórica necessária para execução deste trabalho, baseada em conceitos já fundamentados na academia e na indústria. Este trabalho se baseou fortemente na utilização de FPGAs, como aceleradores reconfiguráveis subseção 2.1; Sistemas em Cloud, como ambiente alvo subseção 2.2; Síntese de Alto Nível, como ferramenta para execução de projeto de hardware subseção 2.3. Nas próximas seções estes tópicos serão discutidos em detalhe.

2.1 Field Programmable Gate Arrays

FPGAs (Field Programmable Gate Arrays) são circuitos eletrônicos que podem ser programados após sua confecção. Muito utilizados em prototipagem de hardware, as FPGAs acabaram se tornando também ferramentas poderosas para aceleração de algumas aplicações em hardware, devido ao seu baixo custo unitário e seu alto poder de reconfiguração.

Consistindo de um denso arranjo de elementos lógicos, como os descritos na subseção 2.1.3, a FPGA consegue representar funções lógicas, que implementam descrições de hardware, para assim fazer a execução de forma paralela e orientada ao de fluxo de dados dos algoritmos. Abaixo serão discutidas brevemente as etapas do fluxo de projeto em FPGA.

2.1.1 Configuração

O fluxo de projeto em FPGAs é composto de uma etapa de configuração, que compreende o desenvolvimento do projeto. As partes que compõem essa etapa são descritas nessa subseção nos itens abaixo.

2.1.1.1 Descrição

A FPGA implementa uma descrição de um circuito lógico utilizando alguma ferramenta de descrição de Hardware, como por exemplo uma linguagem de descrição ou alguma ferramenta gráfica (como de projeto em diagrama de blocos). Esse tipo de linguagem de descrição tem a característica de ser paralela ou poder ser compilada para uma

representação paralela, desta forma explorando as características de paralelismo e fluxo orientado a dados característico dos algoritmos executados em hardware.

2.1.1.2 Síntese

A fase de síntese consiste na tradução da descrição fornecida na etapa anterior, para alguma representação lógica que será mapeada para os componentes disponíveis na FPGA descritos na subseção 2.1.3. Na etapa de síntese já existe uma dependência com a placa alvo, já que o algoritmo será mapeado para um conjunto específico de componentes, presentes naquela placa ou família de placas.

2.1.1.3 Implementação

Na implementação das representações mapeadas na síntese, os componentes virtuais são distribuídos em componentes físicos específicos na topologia da placa, de forma a representar o design que será de fato fisicamente implementado. Nessa etapa, já com o posicionamento elaborado, é feito o roteamento entre os diferentes componentes, permitindo então a construção do fluxo de dados. Nessa etapa existe uma complexa iteração dentro das etapas de roteamento de forma otimizar o resultado final.

2.1.1.4 Carregamento de Bitmap

Na última etapa da configuração, a implementação é traduzida de um computador *Host*, que é utilizado para as etapas anteriores para um arquivo de representação binária (*Bitmap* ou *Bitstream*), e então transferido para as memórias de configuração da placa alvo, por meio de uma interface de programação. Existem hoje sistemas FPGAs mais elaborados que permitem uma configuração dinâmica e de parte parcial da placa. A FPGA utilizada para este trabalho (descrita no capítulo 6) contém estas características de reconfiguração parcial.

2.1.2 Execução

As FPGAs podem ser utilizadas para execuções de forma embarcada ou como coprocessadores de um sistema computacional complexo: este segundo modelo é o de interesse para trabalhos focados em Cloud e é o utilizado neste trabalho.

2.1.2.1 Embarcada

A FPGA embarcada pode desempenhar tarefas como: um circuito integrado desempenhando uma tarefa específica independentemente (*Standalone*), por exemplo um decodifcador digital ou com a FPGA trabalhando em um sistema embarcado heterogêneo, normalmente comunicando-se com um microcontrolador ou processador de características embarcadas, como por exemplo como um transmissor de modulação/demodulação em um sistema de navegação. Em ambos os casos a FPGA precisa de alguma maneira se comunicar com uma interface externa, e em ambos os casos a comunicação pode ser feita por dispositivos de entrada e saída conectados a barramentos no circuito integrado na FPGA; sendo que, no segundo caso, esta comunicação também pode ocorrer por intermédio do microcontrolador/processador associado.

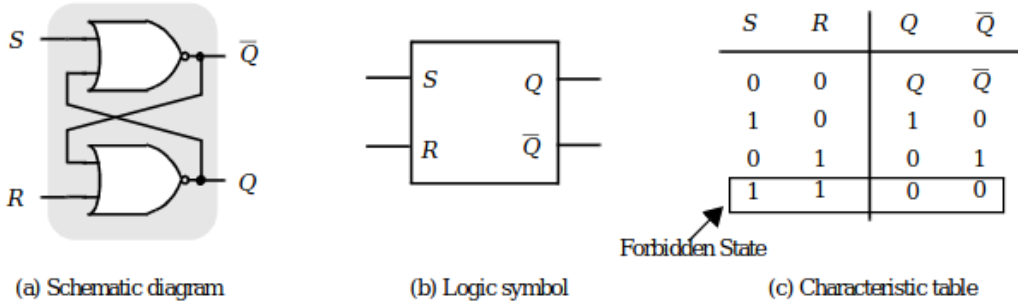
2.1.2.2 Co-Processador

Existe atualmente um aumento na utilização de FPGAs como coprocessadores em sistemas computacionais complexos, como os de Computação de Alta Performance e Sistema *Multi-tenant* em *Cloud*, por exemplo, tem sido muito utilizados na industria e pesquisados ultimamente para acelerar algortimos de aprendizado de máquina. Isso acontece devido à alta capacidade de reconfiguração das FPGAs quando comparadas a outras opções para se implementar hardware e a sua característica arquitetural de processamento em paralelo e orientado a fluxo, que as tornam uma alternativa interessante para substituir a CPU na execução de algumas tarefas. Nesses casos as FPGAs se comunicam com outros nós do sistema heterogêneo (normalmente processadores), descarregando parte das computações deste e devolvendo o resultado final de volta para os processadores.

2.1.3 Componentes

Nesta subseção são explicados os componentes que compõem as FPGAs modernas. Estes que são responsáveis por conseguir fornecer à placa o seu alto poder de reconfiguração e ao mesmo tempo trabalham para melhorar seu potencial de desempenho. Os componentes variam de placa para placa em quantidade e disponibilidade.

Figura 2.1: Flip-Flop(JAN et al., 2003)



2.1.3.1 LUT (Look-Up Table)

As Look-Up-Tables são os componentes básicos de uma lógica programável em FPGA, com circuitos armazenadores de valores (Memórias) de tamanho pequeno e multiplexadores, as LUTs conseguem implementar funções lógicas. Essas implementações são realizadas pela entrada de valores nos bits de controle do multiplexador e a tabela verdade da função armazenada nos circuitos de memória vinculados a cada entrada deste multiplexador, estando assim na saída disponível o valor da função dada uma certa linha da tabela verdade codificada nos bits de controle.

2.1.3.2 DSP (Digital Signal Processor)

É um processador especializado no cálculo de operações aritméticas, e pode ser encontrado embutido dentro da lógica programável das FPGAs. Pode ser roteado dentro do fluxo de dados dentro do chip da FPGA (diretamente junto aos outros componentes) de forma executar operações matemáticas de maior complexidade com menor prejuízo na performance, e se torna mais importante quando a FPGA está desempenho tarefas mais complexas e computacionalmente pesadas.

2.1.3.3 FF (Flip-flop)

São as unidades básicas de memória em FPGA e em diversos componentes de hardware. Nas FPGAs possuem uma alta granularidade de configuração. Os Flip-flops têm as funções nas FPGA de implementar valores de único bit, registradores, e podem ser agrupados para formar blocos de memória em tamanho menor, ou também com características diferentes dos blocos rígidos de memória disponíveis para serem programadas em uma FPGA: as BRAM descritas em 2.1.3.4.

2.1.3.4 BRAM(Block-RAM)

A BRAMs, Memórias RAM em Bloco, são blocos de memória RAM rígidos que estão presentes dentro da lógica programável da FPGA e podem ser configuradas para funcionar junto ao fluxo de dados. Geralmente são uma memória RAM composta de duas portas e são implementadas para partes do algoritmo que tenham grandes requisitos de memória. Apesar de serem blocos rígidos, eles podem ter certa flexibilidade em algumas características que podem ser configuradas.

2.1.3.5 Componentes Acoplados a Lógica Programável

Apesar de não utilizadas neste trabalho, muitas FPGAs apresentam componentes acoplados fora da logica embarcada, mas dentro do mesmo pedaço de silício. Como, por exemplo, FPGAs que acoplam processadores dentro do mesmo Chip para utilização em da categoria descrita em 2.1.2.1, outros processadores como GPUs, ou ainda circuitos lógicos rígidos para operações de Entrada e Saída complexas, como conversão Analógica-Digital/Digital Analógica. Este tipo de FPGAs vem crescendo em quantidade e variedade nos últimos anos.

2.2 Aceleração em Hardware e Sistemas em Cloud

Os grandes núcleos de computação como provedores de Cloud e centros de Computação de Alta Performance se caracterizam por ter uma pesada carga de trabalho. Devido a isto, uma estratégia muito poderosa para poder atingir melhores resultados são arquiteturas heterogêneas de computadores, que utilizam coprocessadores especializados para tarefas especificas como, por exemplo, as GPUs, que são utilizadas para processamento gráfico otimizado e outras aplicações com paralelismo regular. Também, aceleradores baseados em circuitos integrados específicos de aplicação, como aceleradores de rede neural; além de aceleradores diversos baseados em FPGA, que têm ganhado cada vez mais espaço devido a combinar as vantagens de desempenho de um circuito especifico com flexibilidade por causa da alta capacidade de reconfiguração.

Para sistemas de cloud, essa alta capacidade de reconfiguração é muito interessante, pois há uma grande diversidade de clientes realizando requisições ao provedor simultaneamente, de forma que a carga de trabalho se torna muito heterogênea. Ao mesmo tempo que não se tem apenas uma grande tarefa ocupando grandes porções do servidor,

mas sim uma quantidade alta de pequenas tarefas que se dividem com uma maior granularidade no servidor. Para isso, um recurso das FPGAs modernas - PRR (Regiões Reconfiguráveis Parciais) - que são muito úteis, pois se consegue dividir a FPGA entre regiões que podem ser acessadas e reconfiguradas separadamente como dispositivos menores.

A aceleração tem como objetivo melhorar métricas que se tornam cruciais nos sistemas de Computação de Alta Performance e Cloud, relacionadas ao desempenho e ao consumo energético: latência, potencia, vazão, energia consumida pelo sistema e Makespan. Esses dois últimos foram considerados como cruciais para a boa qualidade de serviço de sistemas de Cloud, Multivers sendo as métricas utilizadas e são explicados em detalhes nas subseções 2.2.1 e 2.2.2.

Para que as tarefas sejam escalonadas, usa-se a divisão delas e ***Kernels*** de computação, que são partes de algoritmos ou algoritmos que serão computados e distribuídos entres os nós de aplicações. Os *Kernels* que mais se beneficiam de aceleração em FPGA, são os que possuem alto paralelismo, com baixa dependência de dados interna e com caraterística de serem orientados a dados. Por exemplo, existem diversas aplicações nos campos de Matemática, Simulações Físicas, Criptografia e Grafos que se beneficiam deste tipo de aceleração.

2.2.1 Makespan

O ***Makespan*** é uma métrica definida como o tempo total entre o início e o fim de um trabalho. Esta definição acaba sendo a melhor a ser usada para avaliar o desempenho temporal de sistemas de escalonamento e grandes servidores, pois ele mede o tempo de execução do início da primeira tarefa solicitada até o tempo de conclusão da última tarefa solicitada. Visto isso o ***Makespan*** acaba sendo muito importante para medir o tempo de execução percebido internamente pelo próprio provedor, mas também ao mesmo tempo tendo impacto direto na qualidade de serviço percebida pelo usuário.

2.2.2 Energia

A ***Energia*** tem sido cada vez mais importante como métrica de avaliação em grandes servidores. Apesar de ser uma métrica tradicional em Sistemas Embarcados, ela se torna significativa em grandes servidores que têm: a) Uma grande carga de trabalho; e

b) Sistemas de alto-desempenho que trabalham perto de seus limites físicos. Esses dois fatores juntos acabam gerando uma grande dissipação de potência durante a execução desses computadores ou núcleos de computadores, ou seja, consumo energético. Muito tem se feito para mensurar e mitigar esse grande consumo, pois existe um alto valor econômico associado a altos gastos de energia. Dado esse valor econômico associado, a Energia é percebida neste trabalho como um dos grandes fatores a ser considerados quando se projeta sistemas e arquiteturas para o provedor de *Cloud*, buscando-se reduzir o consumo energético, mas sempre levando em conta os seus impactos na qualidade de serviço oferecida e observando a relação entre o tempo de execução e potência que são os fatores que compõem o gasto energético final, pois a energia consumida é dependente da potência e do tempo de execução como mostrada na equação 2.1.

$$EnergiaConsumida = \int_{T1}^{T2} P\,dt \tag{2.1}$$

2.3 Síntese de Alto Nível

Os fluxos para desenvolvimento clássico de hardware têm longo tempo de desenvolvimento e alto grau de complexidade para iteração do design, o que leva a um *time-to-market* mais longo. O fluxo de síntese de alto-nível é uma alternativa para abstrair algumas das etapas necessárias para o desenvolvimento de hardware reduzindo o tempo de desenvolvimento.

Diferentemente das estratégias clássicas, como desenvolvimento em HDLs, que usam uma descrição em blocos das entidades físicas que compõem o circuito digital, o fluxo HLS explora o uso de um compilador de alto nível. Em HLS, uma descrição em uma linguagem de alto nível (por exemplo C) do problema é fornecida para o compilador. O compilador, então, gera da descrição algorítmica uma descrição em forma de circuito digital, abstraindo do projetista a etapa de projeto em blocos de descrição de baixo nível.

Porém, ao se desenvolver hardware existem diversas soluções equivalentes para o mesmo problema, tornando possível o desenvolvimento de diversos circuitos funcionalmente equivalentes com diferentes arquiteturas. Isso torna possível que se explore diversos níveis de paralelismo espacial ou profundidade em *pipelines*. Essas diferenças na arquitetura do hardware proporcionam diferentes resultados em potência, energia, desempenho, utilização de recursos e outras restrições. No fluxo tradicional de desenvolvimento com HDLs, essas arquiteturas teriam de ser individualmente projetadas e implementa-

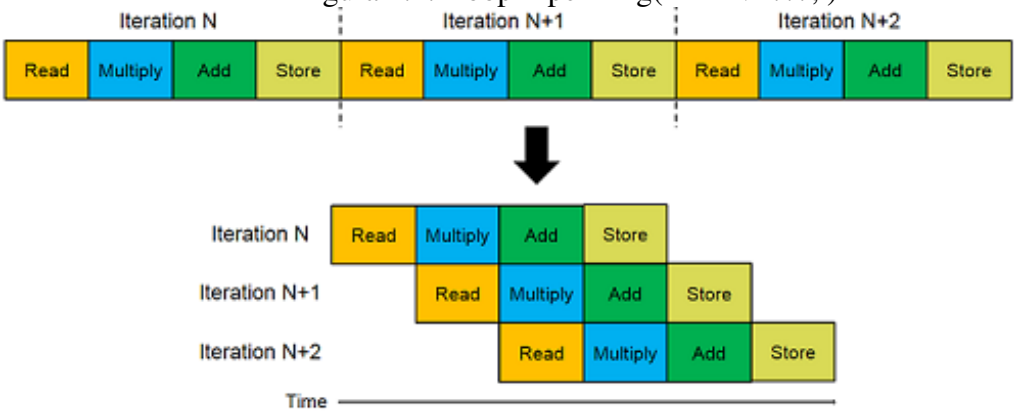
das pelo desenvolvedor de hardware, enquanto nos compiladores de HDL modernos, o mesmo código pode ser sintetizado para arquiteturas distintas usando anotações de alto nível, tornando possível gerar diversas versões de hardware utilizando praticamente o mesmo código.

Além de permitir a rápida exploração do espaço de projeto gerando diversas versões de hardware de alto nível, os compiladores modernos de síntese de alto nível proporcionam também diversas informações geradas em tempo de síntese que possibilitam a avaliação das métricas de cada uma das versões, para que então o projetista possa escolher a que melhor se adapta ao seu projeto. Nas subseções 2.3.1 e 2.3.2 abaixo serão discutidos em maior detalhe dois tipos de modificações arquiteturais provenientes de anotações no código

2.3.1 Loop Unrolling

A técnica de ***Loop Unrolling*** consiste em desenrolar laços. Isto é, a partir de laços que seriam iterados diversas vezes, e repetidos por meio de uma configuração de controle, são transformados em várias instâncias individuais separadas e iguais, compostas pelo equivalente ao que seria executado dentro de cada iteração do loop(laço). No caso do ***Loop Unrolling*** aplicado ao hardware, essas instâncias são executadas em paralelo, por estruturas iguais, explorando o paralelismo espacial para ganhar em performance, em detrimento da área necessária para sintetizar o circuito.

Figura 2.2: Loop Pipelining(XILINX. . . ,)



2.3.2 Loop Pipelining

Enquanto a técnica de ***Loop Unrolling*** 2.3.1 explora o paralelismo espacial, a técnica de ***Loop Pipelining*** explora o paralelismo temporal para melhorar a execução de laços. A técnica de *Loop Pipelining* consiste em dividir uma iteração em diversas etapas, e adiciona registradores para armazenar os resultados parciais da execução da instancia entre etapas. Desta forma, quando a computação de uma etapa é concluída, o registrador pode guardar o resultado parcial de forma a seguir para a próxima etapa, liberando a primeira etapa para executar uma nova iteração, enquanto a primeira iteração está sendo processada pela próxima etapa, fazendo-as ser executadas paralelamente no tempo, porém dentro do mesmo fluxo de dados e das mesmas unidades funcionais.

3 ESTADO DA ARTE

Esse capítulo se dedica a uma revisão da literatura em nível estado da arte para avaliação do impacto deste trabalho no panorama atual. Foram analisados trabalhos de relevância nos campos de Computação Colaborativa (seção 3.1) e de Exploração de Espaço de Projeto em HLS (seção 3.2). Após é feito uma análise do que ***Multivers*** acrescenta no estado atual destes dois campos (seção 3.3).

3.1 Computação Colaborativa

A Computação Colaborativa vem sendo proposta como uma maneira de tirar proveito da eficiência de múltiplas arquiteturas a partir da combinação de seus diferentes elementos computacionais. Isso é atingido particionando tarefas/*Kernels* entre diferentes dispositivos. (HUANG et al., 2019) explora o potencial da execução colaborativa entre CPU e FPGA a partir da comparação de duas abordagens distintas: particionamento de tarefas e dados. Eles mostram que a estratégia correta de particionamento pode aumentar o ganho em performance em ambientes CPU-FPGA.

Os autores em (WEI et al., 2017) otimizam a vazão de aplicações de *streaming* em sistemas heterogêneos CPU-FPGA usando algoritmos de mapeamento de tarefas. Eles também empregam *pipelining* para melhor a vazão e frequência escalando para economia energética.

O trabalho proposto por (SHAN et al., 2019) foca na otimização de mapeamento de algoritmos multi-kernel de alta performance para Redes Neurais e Aprendizado de Máquina para plataformas multi-FPGA. Escolhendo a melhor configuração de paralelismo para a aplicação multi *Kernel*, baseado na disponibilidade de recursos em cada dispositivo utilizando aproximação por Programação Geométrica. O algoritmo de alocação produz uma solução que otimiza a vazão da plataforma.

3.2 Exploração de Espaço de Projeto com HLS

Para reduzir o esforço de programação, a Síntese de Alto Nível (HLS) vem sendo constantemente melhorada nos anos recentes, enquanto disponibiliza diversas otimizações de hardware. O uso de diferentes otimizações e suas combinações resultam em di-

ferentes versões de um mesmo *Kernel*. Como discutido na seção 2.3 do capítulo 2, essas implementações podem gerar variações em termo de performance/área/potência/energia.

Pham et al. propõem uma Exploração Dinâmica de Espaço de projeto que aproveita as dependências de *Loop-array* para achar melhor combinação de grão grosso em otimizações de HLS (*loop unrolling, pipelining, e array partitioning*) (KHANH et al., 2015). A *Framework* propõem uma análise de *Kernels* construída a partir de um grafo de dependências que aumenta a velocidade média de exploração em 14x.

Lin-Analyzer (ZHONG et al., 2016) é uma ferramenta de análise que desempenha estimativas rápidas e precisas de performance em FPGA, assim como Exploração de Espaço de Projeto de acordo com diversas otimizações de grão grosso sem geração de implementação RTL. Ele identifica gargalos em diferentes implementações de FPGA quando aplicadas a estas otimizações, auxiliando o projetista em avaliar diferentes arquiteturas disponíveis pela Síntese de Alto Nível.

MP-Seeker (ZHONG et al., 2017) é uma ferramenta de análise em alto nível, que avalia métricas de performance/área de diversas opções de aceleradores para uma aplicação em estados iniciais da exploração de espaço de projeto, antes da chamada de ferramentas HLS para a etapa final de síntese. Contrariamente ao Lin-Analyzer, MP-Seeker proporciona uma avaliação de parâmetros em grão fino como *tile size* e número de *PEs*, e parâmetros de grão grosso. Sua rápida Exploração de Espaço de Projeto acha uma combinação quase-ótima das opções de paralelismo.

COMBA (ZHAO et al., 2019) é uma *Framework* baseada em modelo que avalia os efeitos de variedade de diretivas relacionadas a funções, laços e arranjos pelo uso de modelos analíticos agregáveis, um coletor de dados recursivo, é um algoritmo de Exploração de Espaço de Projeto orientado a métrica. Dadas diferentes restrições de recursos, COMBA encontra uma configuração de *Kernel* de performance quase-ótima.

3.3 Contribuições ao Estado da Arte

Como discutido acima, mesmo havendo trabalhos que avaliam Execuções colaborativas em CPU-FPGA (HUANG et al., 2019; WEI et al., 2017) e ambientes multi-FPGA (SHAN et al., 2019)), e outros tenham explorado apenas Síntese de Alto Nível para otimizar a geração de *Kernels* (KHANH et al., 2015; ZHONG et al., 2016; ZHONG et al., 2017; ZHAO et al., 2019), ***Multivers*** é o primeiro trabalho que constrói uma ponte entre Síntese de Alto Nível e Execução Colaborativa CPU-FPGA em ambientes de *Cloud*, ti-

rando vantagem da geração automática e rápida de versões para vários *Kernels* via HLS. Isso aumenta significativamente as possibilidades na Exploração de Espaço de de Projeto, e proporciona ao provedor de Cloud a oportunidade de priorizar: paralelismo (requisitos para baixa área), energia ou performance de tempo de execução, de acordo com os requisitos do provedor e carga de trabalho em dado momento.

4 EXPLORAÇÃO DE ESPAÇO DE PROJETO

Devido à variabilidade de circuitos lógicos equivalentes para implementar o mesmo algoritmo em hardware e aos métodos de desenvolvimento que estão disponíveis para o fluxo de projeto rápido dos mesmos, foram realizados experimentos para avaliar se haveria espaço de exploração ao se usar diferentes diretivas de desenvolvimento via síntese de alto nível com objetivo de gerar amplitude nos resultados topológicos resultando em diferenças nos parâmetros de energia, performance e utilização de recursos entre diferentes versões. Sendo estes parâmetros interessantes para aplicações nos mais diversos casos de uso de aceleradores em FPGA, incluindo os emergentes aceleradores em Cloud.

Foram avaliados 8 benchmarks buscando diversidade tanto do ponto de vista computacional, quanto do ponto de vista de área de aplicação. Estes são explicados na seção 4.1. Para realizar a exploração inicial do espaço de projeto foram realizados experimentos utilizando a ferramenta de Síntese de Alto Nível **Vivado HLS** e a ferramenta de síntese **Vivado**, ambas da Xilinx. Para dados de performance e de utilização de recursos foram considerados os resultados do relatório de síntese do Vivado HLS e para os dados de energia foram considerados os resultados do relatório de síntese da ferramenta Vivado. Como os experimentos, explicados em maiores detalhes na seção 4.2, têm como objetivo validar os espaços de projeto em Cloud, o dispositivo utilizado foi a placa de aceleração **Xilinx Alveo A200**. Estes mesmos experimentos mostram também indícios que o mesmo espaço de exploração existe para outros nichos em que há um *trade-off* entre performance, recursos e energia, como sistemas embarcados.

4.1 Benchmarks

4.1.1 ADI (Alternating Direction Implicit solver)

ADI (Alternating Direction Implicit solver) (Louis-Noel Pouchet, 2019) é um algoritmo numérico de álgebra linear, que utiliza métodos iterativos para solucionar equações matriciais Sylvester. É um método popular para solucionar grande equações matriciais que aparecem em teoria de sistemas e controle.

4.1.2 Floyd-Warshall

Floyd-Warshall (Louis-Noel Pouchet, 2019) é um algoritmo que resolve o problema de caminho mais curto em um grafo direcionado e com pesos (negativos ou positivos). O algoritmo calcula a distância do caminho mais curto entre todos os pares de vértices, porém sem retornar detalhes de cada par.

4.1.3 MD5

MD5(RIVEST, 1992) é o quinto de uma série de algoritmos estilo *message digest* desenvolvido por Ronald Rivest. Foi projetado como função de *Hash* criptográfico, porém devido a vulnerabilidades hoje é utilizado como função de *Checksum* para verificação de integridade de dados.

4.1.4 Matrix Multiplication

Matrix Multiplication (Louis-Noel Pouchet, 2019) é um algoritmo que desempenha papel central em muitos algoritmos numéricos, como por exemplo em computação científica e reconhecimento de padrões. O algoritmo implementado para realizar estas operações no *Benchmark* utilizado é algoritmo básico baseado na definição da operação.

4.1.5 Pivot

Pivot (LIU; BAYLISS; CONSTANTINIDES, 2015) é a implementação da operação de pivotamento presente na eliminação de Gauss. É um algoritmo para resolver sistemas de equações lineares através da aplicação de operações matemáticas elementares.

4.1.6 RowCol

RowCol (LIU; BAYLISS; CONSTANTINIDES, 2015) é um algoritmo que realiza a implementação de um loop 2D simplificado. O algoritmo foi utilizado das referências

de utilização das ferramentas da Xilinx.

4.1.7 Seidel

Seidel (Louis-Noel Pouchet, 2019) é um algoritmo para o solucionar o problema de caminho mais curto entre todos os pares para grafos conectados não direcionados e sem peso. Mesmo sendo projetado para grafos conectados, pode ser aplicado para cada componente conectado de um grafo.

4.1.8 Triangular Solver

Triangular Solver (Louis-Noel Pouchet, 2019) é um algoritmo para solução de matrizes triangulares. Na álgebra linear, matrizes triangulares são um tipo especial de matriz quadrada. Por serem mais fáceis de serem solucionadas, essas matrizes se tornam muito importantes na análise numérica.

4.2 Experimentos Iniciais

Para avaliar o espaço de exploração, foram sintetizados, utilizando HLS, códigos em linguagem C dos *Benchmarks* descritos na seção 4.1, nas condições descritas no começo deste capítulo. Os experimentos foram executados de forma a utilizar diretivas de compilação que indicam ao compilador de HLS como realizar a síntese da linguagem algorítmica para o circuito lógico equivalente. Para este trabalho foram utilizadas as diretivas de ***loop unrolling*** e ***loop pipelining***, que foram descritas em trabalhos anteriores como sendo as que mais impactam o resultado do hardware gerado (ZHONG et al., 2016). Para isso foram explorados os laços dentro dos *Kernels* presentes nos *Benchmarks* avaliados, em que versões foram geradas através das combinações de ***loop unrolling***, ***loop pipelining*** e nenhuma diretiva, descartando as combinações que seriam ignoradas pelo compilador (diretivas de *loop unrolling* aninhadas dentro de um laço mais externo com *loop pipelining*).

A exploração foi realizada utilizando uma ferramenta desenvolvida como parte deste trabalho que a partir de diretivas de pré-compilação do código e scripts de manipulação, consegue utilizar de forma automática as ferramentas **Vivado HLS** e **Vivado**.

Desta forma, ela gera a partir de um único código e de um arquivo com diretivas de pré compilação diversas versões de Hardware Sintetizado e seus respectivos relatórios, assim como relatórios de comparação entre as versões. Essa mesma ferramenta foi utilizada para gerar o *Kernels* que compõem a *Kernel Library* descrita no capítulo 5 como parte fundamental da *Framework* ***Multivers***.

Desta forma foram encotradas 6 diferentes versões de acelerador para o *Benchmark* com menos versões e 20 diferentes versões de acelerador para o *Benchmark* com mais versões, variação essa diretamente decorrente das diretivas a serem exploradas nos laços de cada algoritmo.

Após, foram classificadas as versões conforme suas características, onde para cada versão foi atribuído um peso calculado através de uma combinação linear entre os dados gerados pela ferramenta para cada uma das versões em Energia, Performance e Utilização de Recurso (Área) e pesos cada uma destas (equação 4.1), com os pesos representando a maior importância no espaço de exploração para cada uma das dimensões. Sendo (γ) referente à Energia, (β) referente à performance e α referente à área.

$$KernelBenefitValue \rightarrow \alpha\ Area + \beta\ Exec.Time + \gamma\ Energy \tag{4.1}$$

onde, Área, Tempo de Execução e Energia são normalizados considerando os máximo e mínimos valores de cada. A Área é dado pelo pior caso da porcentagem de utilização de recursos da FPGA (BRAM, LUT, DSP e FF)

As diferentes versões que atingiram os melhores resultados para cada peso são mostradas nas figuras 4.1 até 4.8 com granularidade de 0,25 para cada um dos recursos, e com os números dentro das células sendo usados como identificadores arbitrários de versão.

4.3 Análise

Ao analisarmos na Figura 4.3, vemos, por exemplo, que para o Benchmark MD5 cada uma das tabelas é configurada para um valor α de Área, e em cada uma das colunas existe um valor β de performance e em cada uma das linhas um valor γ de energia. Dentro das células onde há o número da melhor versão conforme a equação 4.1 para aqueles valores. Para o MD5 percebemos que para os pesos $\alpha = 0$; $\beta = 0.5$; e $\gamma = 0.25$ a versão número 17 seria a melhor para este cenário. Assim como percebemos algumas versões que dominam os casos de contorno, (como por exemplo foco total em performance $\alpha =$

0; $\beta = 1$; $\gamma = 0$), e outras versões, como a versão 3, dominam as condições mais equilibradas (por exemplo a versão 3 com $\alpha = 1$).

Ainda é possível perceber que alguns *Benchmarks* acabam tendo mais versões relevantes que outros, como MD5 e Floyd-Warshall, que têm 5 e 4 versões relevantes para as métricas apresentadas; e outros que apresentam menos versões relevantes, como algoritmo de Seidel, que apresenta apenas 3, sendo que 2 das versões com uma prevalência muito mais importante.

Ao analisarmos as tabelas referentes à distribuição das versões, podemos notar que existe um espaço de projeto a ser explorado para as diferentes versões geradas por HLS, e que elas se distribuem conforme se varia quais dimensões entre Energia, Performance e Área no que diz respeito a sua importantes. A partir do capítulo 5 é explicado uma *Framework* que explora dinamicamente o esse espaço de projeto decorrente do fluxo HLS para que se obtenha melhores resultados para sistemas em *Cloud*.

Figura 4.1: ADI

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	1	1	1	1
0.25	0	0	0	0	3
0.5	0	0	0	0	0
0.75	0	0	0	0	0
1.0	0	0	0	0	0

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	1	1	1	1
0.25	3	3	3	3	3
0.5	0	0	0	3	3
0.75	0	0	0	0	0
1.0	0	0	0	0	0

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	1	1	1	1
0.25	0	0	3	3	3
0.5	0	0	0	0	0
0.75	0	0	0	0	0
1.0	0	0	0	0	0

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	1	1	1	1
0.25	3	3	3	3	3
0.5	0	3	3	3	3
0.75	0	0	0	0	3
1.0	0	0	0	0	0

Figura 4.2: Floyd Warshall

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	8	8	8	8
0.25	7	8	8	8	8
0.5	7	8	8	8	8
0.75	7	8	8	8	8
1.0	7	7	8	8	8

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	5	5	8	8
0.25	5	5	5	8	8
0.5	5	5	5	5	8
0.75	5	5	5	5	8
1.0	5	5	5	5	8

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	5	8	8	8
0.25	5	5	8	8	8
0.5	5	5	8	8	8
0.75	5	5	5	8	8
1.0	5	5	5	8	8

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	5	5	5	5
0.25	5	5	5	5	5
0.5	5	5	5	5	5
0.75	5	5	5	5	5
1.0	5	5	5	5	5

Figura 4.3: MD5

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	17	17	17	17
0.25	7	7	17	17	17
0.5	7	7	7	17	17
0.75	7	7	7	7	17
1.0	7	7	7	7	7

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	0	3	3	3
0.25	3	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	6
1.0	6	6	6	6	6

α:0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	3	3	3	3
0.25	3	3	3	3	6
0.5	6	6	6	6	6
0.75	6	6	6	6	6
1.0	6	6	6	6	6

α:0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	0	0	3	3
0.25	0	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	3
1.0	3	3	3	3	3

Figura 4.4: MM

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	3	3	3	3
0.25	14	14	14	14	14
0.5	14	14	14	14	14
0.75	14	14	14	14	14
1.0	14	14	14	14	14

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	3	3	3
0.25	1	3	3	3	3
0.5	1	3	3	3	3
0.75	3	3	3	3	3
1.0	3	3	3	3	3

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	3	3	3	3
0.25	1	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	3
1.0	3	3	3	3	3

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	3	3	3
0.25	1	1	3	3	3
0.5	1	3	3	3	3
0.75	1	3	3	3	3
1.0	3	3	3	3	3

Figura 4.5: Pivot

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	4	4	4	4
0.25	1	1	1	1	1
0.5	1	1	1	1	1
0.75	1	1	1	1	1
1.0	1	1	1	1	1

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	0	0	0	0
0.25	1	1	1	1	1
0.5	1	1	1	1	1
0.75	1	1	1	1	1
1.0	1	1	1	1	1

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	0	0	0	0
0.25	1	1	1	1	1
0.5	1	1	1	1	1
0.75	1	1	1	1	1
1.0	1	1	1	1	1

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	1	0	0	0	0
0.25	1	1	1	1	1
0.5	1	1	1	1	1
0.75	1	1	1	1	1
1.0	1	1	1	1	1

Figura 4.6: RowCol

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	1	1	1	1
0.25	5	5	1	1	1
0.5	5	5	5	1	1
0.75	5	5	5	5	5
1.0	5	5	5	5	5

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	1	1	1
0.25	3	3	1	1	1
0.5	3	3	3	1	1
0.75	3	3	3	3	1
1.0	3	3	3	3	3

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	1	1	1
0.25	3	3	1	1	1
0.5	3	3	3	1	1
0.75	5	3	3	3	1
1.0	5	5	5	3	3

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	1	1	1
0.25	3	3	1	1	1
0.5	3	3	3	1	1
0.75	3	3	3	3	1
1.0	3	3	3	3	3

Figura 4.7: Seidel

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	3	3	3	3
0.25	3	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	3
1.0	3	3	3	3	3

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	1	3	3
0.25	1	1	3	3	3
0.5	1	1	3	3	3
0.75	1	1	3	3	3
1.0	1	3	3	3	3

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	3	3	3
0.25	1	3	3	3	3
0.5	1	3	3	3	3
0.75	1	3	3	3	3
1.0	3	3	3	3	3

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	1	1	1	3
0.25	1	1	1	3	3
0.5	1	1	1	3	3
0.75	1	1	1	3	3
1.0	1	1	3	3	3

Figura 4.8: Trisolv

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	8	8	8	8
0.25	8	8	8	8	8
0.5	8	8	8	8	8
0.75	8	8	8	8	8
1.0	8	8	8	8	8

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	19	1	8	8	8
0.25	15	8	8	8	8
0.5	15	8	8	8	8
0.75	15	8	8	8	8
1.0	8	8	8	8	8

α: 0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	19	8	8	8	8
0.25	15	8	8	8	8
0.5	8	8	8	8	8
0.75	8	8	8	8	8
1.0	8	8	8	8	8

α: 0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	19	19	1	8	8
0.25	7	8	8	8	8
0.5	15	8	8	8	8
0.75	15	8	8	8	8
1.0	15	8	8	8	8

5 SISTEMA MULTIVERS

Baseado na análise feita a partir dos experimentos descritos no capítulo 4, desenvolveu-se o *Framework* ***MultiVers***, que a partir da exploração de espaço de projeto em HLS e do alto poder de reconfiguração das FPGAs, consegue fornecer dinamicamente diferentes versões dos aceleradores de hardware para um sistema de *Cloud*, e realizar o escalonamento de tarefas de forma colaborativa entre os processadores de propósito geral tradicionais e os aceleradores. Conseguindo assim melhores resultados combinando uma primeira etapa de melhores escolhas locais, focadas em cada acelerador, com uma segunda etapa que busca um melhor escalonamento global utilizando o resultante da primeira etapa.

A Figura 5.1 mostra uma visão geral da *Framework* ***MultiVers***. ***MultiVers*** foi projetado para executar como um serviço para *Cloud*, permitindo ao provedor facilmente configurar: a) O alvo de otimização, o qual determina o melhor *Kernel* a ser utilizado e a estratégia de otimização a ser utilizada para a alocação dos *Kernels* requisitados pelos clientes e, b) o tamanho do *batch*, o qual é dado pelo número de requisições de *Kernels* a serem otimizados. Esses são os parâmetros de configuração da Figura 5.1. ***MultiVers*** também recebe como entrada uma restrição dos recursos da FPGA. Este é um importante parâmetro, visto que sistemas *Cloud* baseados em FPGA podem compartilhar a mesma FPGA entre diversos clientes, dividindo-a em múltiplas regiões de diferentes capacidades(PRRs, (NGUYEN; KUMAR, 2020)) para maximizar a utilização da FPGA.

A *Framework* ***MultiVers*** é dividida em 5 passos(*steps*), como mostrado na Figura 5.1. Primeiro, ***MultiVers*** coleta as requisições de *Kernel* dos clientes atuais (**Collecting Tenant Requests**). Durante o segundo passo (**Kernel Selection**), os *Kernels* que chegam são selecionados e seus *designs* para FPGA são selecionados baseado no alvo de otimização do provedor de Cloud, o qual é definido pela tupla de pesos já utilizada para avaliar os Kernels na seção 4.2 do capítulo 4. Dependendo da importância dada a cada parâmetro, uma diferente implementação do *Kernel* será selecionada e encaminhada para

Figura 5.1: Visão Geral da Framework Multivers

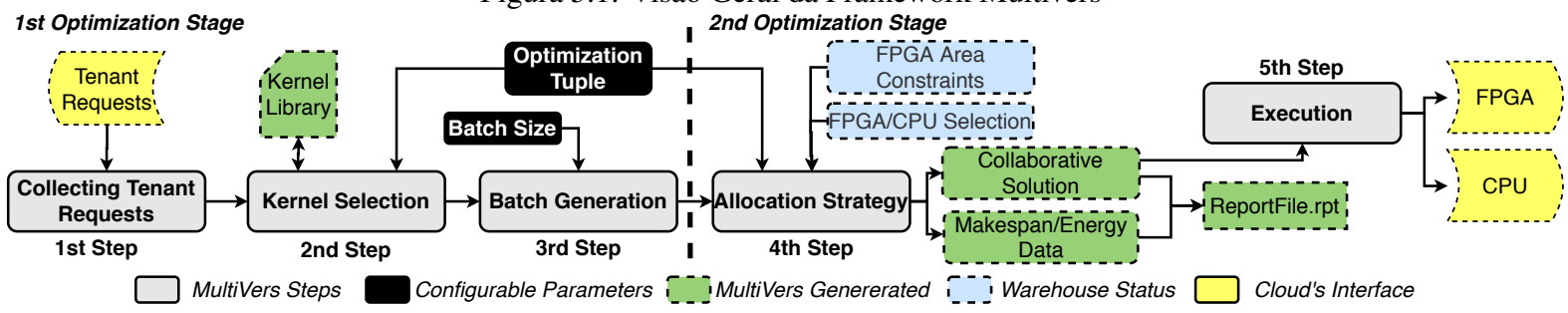
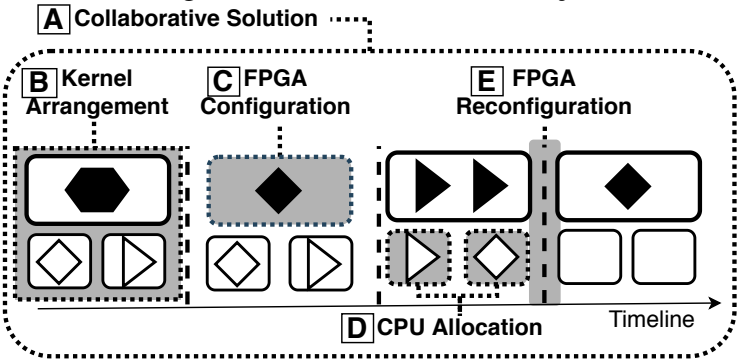


Figura 5.2: Conceitos de Alocação



o próximo passo, diretamente impactando o *Makespan* e a Energia da execução. Portanto, dada uma tupla de otimização, o algoritmo de seleção de *Kernel* (**Kernel Selection Algorithm**, detalhada na seção 5.4) seleciona a versão mais adequada do *Kernel* da biblioteca de Kernels (**Kernel Library**), a qual é gerada pela ***MultiVers***, baseada na exploração feita no capítulo 4. A biblioteca de *Kernels* armazena para as versões dos *Kernels* seus *bitmaps* e binários para CPU, junto com informações adicionais como seus tempos de execução em CPU e FPGA, utilização de recursos e consumo energético.

Na Geração de Batch (**Batch Generation**) no terceiro passo, as versões do *Kernels* de entrada (selecionadas no passo anterior de acordo com os parâmetros de otimização) são enviados para uma fila (*First In, First Out*). Uma vez que *Kernels* suficientes são reunidos (por exemplo, o número de *Kernels* é igual ao tamanho do Batch), o *Kernel* é enviado para a heurística colaborativa de alocação, recebendo de entrada: a) o Batch contendo as informações dos *Kernels*; e b) informações sobre a FPGA (por exemplo, restrição de recursos coletadas a partir do estado atual do provedor) e CPU (por exemplo, número de *Cores* disponíveis), que correspondem ao **Warehouse Status** da Figura 5.1.

O quarto passo de ***MultiVers*** entrega como saída uma Solução colaborativa (**Collaborative Solution**), que também é gerada pelo ***MultiVers***. A solução colaborativa é explicada em mais detalhes na Figura 5.2, em que uma FPGA e duas CPUs estão disponíveis no provedor. A solução colaborativa consiste em um conjunto de arranjos de *Kernels* (**Kernel Arrangements**) Figura 5.2.B distribuídos sobre o tempo (eixo-x). Cada arranjo de *Kernel* inclui um conjunto de *Kernels* descarregado para a FPGA - **FPGA Configuration** (Figura 5.2.C) - e outros *kernels* que serão executados na CPU - **CPU Allocation** ((Figura 5.2.D) para cada novo Arranjo de Kernels haverá uma nova **FPGA Reconfiguration** (Figura 5.2.E). Também pode-se notar que alguns Arranjos de *Kernel* podem não ter *Kernels* executado na CPU, como o último na Figura 5.2, devido a nem sempre haver benefício em despachar *Kernels* para a CPU.

No último passo Execução (**Execution**) ***MultiVers*** é responsável por disparar a execução no ambiente CPU-FPGA de acordo com a solução previamente encontrada (saídas em amarelo na Figura 5.1). Um relatório contendo o *Makespan* e o Consumo de Energia resultantes da execução dos *Kernels* no ambiente colaborativo é, então, produzido. Nas próximas seções, são apresentados o algoritmo para seleção de *Kernels* e a as estratégias de alocação disponíveis nos segundos e quartos passos de ***MultiVers***.

5.1 Passo 1 - Coleta de Requisições dos Clientes (Collecting Tenant Request)

As plataformas de *Cloud* vem ganhando cada vez mais força nos dias atuais devido a possibilidade de dividir dinamicamente diversos recursos de um sistema computacional, utilizando estratégias como Software-as-a-Service (Software como um Serviço). A decorrência desse compartilhamento de recursos é uma base de clientes com necessidades heterogêneas, o que acaba por gerar diversas solicitações de serviços. A primeira etapa da Framework então tem como objetivo recolher essas requisições de diversos clientes por meio de uma interface com o provedor, de forma a fornecer a informar para a *Framework* quais serviços, processados como *Kernels*, precisam ser processados por ***Multivers*** e entregues ao cliente após o processamento nas etapas seguintes da *Framework*

5.2 Passo 2 - Seleção de Kernel (Kernel Selection)

Esse passo é responsável por escolher a versão da biblioteca de *Kernels* para cada requisição de *Kernel* de acordo com os objetivos de otimização do provedor de *Cloud*. O Algoritmo de Seleção de *Kernel*: **a)** recebe uma identificação do *Kernel* requisitado e o conjunto de pesos da tupla de otimização (α,β,γ), onde analogamente ao Capítulo 4 os pesos representados são α representa a área, β a performance e γ a energia. **b)** para cada versão de *Kernel* na Kernel Library (formada pelas versões descritas no Capítulo 4 para análise de DSE) é calculado o Kernel Benefit Value, também descrito na análise do Capítulo 4 na equação 4.1, para cada uma versão do *Kernel* é então é selecionada a versão com menor **Kernel Benefit Value** (quanto melhor o valor melhor); **c)**o versão selecionada é retornada para o passo seguinte da *Framework*, conforme figura (5.1).

5.3 Passo 3 - Geração de Batch (Batch Generation)

Essa etapa tem como objetivo o agrupamento dos *Kernels* escolhidos pela etapa anterior em *Batches* de tamanho definido pela entrada do provedor, para que os *Kernels* depois de agrupados possam ser alocados no passo 4, descritos na seção 5.4.

5.4 Passo 4 - Estratégias de Alocação (Allocation Strategies)

Para o quarto passo, Estratégias de Alocação, estão implementadas utilizando algoritmos de *Genetic Multidimensional Knapsack (GMKs)* e *First-Come First-Served (FCFS)*, os algoritmos GMKs são implementados como no trabalho (Jordan et al., 2021), que trata de algoritmos de alocação para *Multi-tenant* e estão brevemente descritos abaixo, assim como o FCFS.

As estratégias hoje implementadas em ***Multiverse*** são as seguintes: FCFS, *Genetic Multidimensional Knapsack para Makespan* (GMK-M) ou *Genetic Multidimensional Knapsack para Energia* (GMK-E). O uso de tanto GMK-M ou GMK-E é baseado na tupla de otimização. Sempre que o provedor de *Cloud* focar em Makespan(por exemplo, o maior peso da tupla é o de performance), o algoritmo utilizado é o de GMK-M. Nos outros casos (quando foco é em Energia ou Energia e Performance têm pesos iguais), o algoritmo de GMK-E é utilizado como estratégia de alocação otimizada padrão. Devido a suas equações de ganho, GMKM e GMK-E consideram a área igualmente nas suas estratégias de otimização, como explicado a seguir(Subsection 5.4.1).

5.4.1 Genetic Multidimensional Knapsack (GMK)

Usar o algoritmo de alocação GMK é muito eficiente nesse caso específico, pois ele é capaz de maximizar um objetivo dadas múltiplas restrições (BRAM,LUTs,DSPs, e FFs) (KELLERER; PFERSCHY; PISINGER, 2004); e sua propriedade de mutação genética aumenta a qualidade da solução entregue. No caso de ***Multivers***, o GMK foi adaptado para minimizar o número de *FPGA reconfigurations* (Knapsacks). Para isso, ele irá maximizar o provisionamento de recursos de FPGA para *Kernels* que ao mesmo tempo: a) São mais significativos em termos de Makespan ou Energia; b) Se beneficiam mais da aceleração em FPGA; e c) necessitam menos provisionamento de recursos, o

que habilita mais *Kernels* para serem executados em paralelo. *Kernels* que não atinjam nenhum desses requisitos, e podem ser executados colaborativamente serão despachados para serem executados na CPU.

A partir disso são utilizadas duas versões de GMK: uma para reduzir o Makespan (GMK-M) e outra que otimiza o consumo energético (GMK-E). A equação 5.1 modelos de ganho para aceleração de Makespan considera a relevância do tempo de execução do *Kernel* avaliado (k_{TR}), a aceleração do *Kernel* avaliado em FPGA (k_A), e o seu consumo de recursos (r_{ck}). A Relevância do tempo de execução do *Kernel* (k_{TR}) é dada pela equação 5.2, onde $\sum_{i=1}^{n} k_{TC_n}$ é o tempo total de execução dado por todos os *Kernels* de um *Batch*, e k_{TC} é o tempo de execução do *Kernel* atual na CPU. A aceleração do *Kernel* (k_A) é dada pela equação 5.4 e apresenta o pior caso em utilização de recursos (entre BRAM/LUT/DSP/FF). Em outras palavras, é utilizada a razão entre os recursos disponíveis e utilizados de cada tipo.

O principal objetivo de tal modelagem é maximizar o número de *Kernels* alocados na FPGA, priorizando a execução do *Kernels* mencionados anteriormente.

$$\textit{GMK-M profit} \rightarrow 1/(k_{TR} + 1/k_A + r_{ck}) \tag{5.1}$$

$$k_{TR} \rightarrow (\sum_{i=1}^{n} k_{TC_n} - k_{TC}) / \sum_{i=1}^{n} k_{TC} \tag{5.2}$$

$$k_A \rightarrow k_{TC}/k_{TF} \tag{5.3}$$

$$r_{ck} \rightarrow max(k_{(RES)}/F_{(RES)}) \tag{5.4}$$

5.4.1.1 Parte 1

O Modelo de ganho para otimização energética segue a mesma ideia do modelo anterior, focando em consumo energético em vez de aceleração do Makespan. A seguir, são detalhadas as duas principais fases da estratégia de GMK: **Geração de Configuração da FPGA** (Parte 1) usa o Algoritmo Genético para gerar uma Configuração de FPGA quase-ótima, que é representada por um conjunto de *Kernels* que, baseado nas restrições mencionadas acima (LUTs,DSPs,BRAMs, e FF), são alocadas na FPGA. Possuindo as seguintes etapas: **a)** geração de uma população inicial representada por uma *FPGA configuration* set; **b)** recombina as configurações a partir do conjunto atual para produzir uma nova configuração; **c)** aplica a função de ajuste para avaliar a nova configuração, com objetivo de maximizar performance ou energia, ao mesmo tempo que respeita as restri-

ções de recursos da FPGA; **d)** aplica a mutação na configuração selecionada e substitui todas as configurações do conjunto inicial pelas novas;**e)** confirma se ambos, o número de gerações e a convergência foram atingidas (SAFE et al., 2004) e invoca a Parte II. Se não retorna para o para etapa b.

5.4.1.2 Parte 2

A Otimização Colaborativa (Parte 2) avalia *Kernels* que não foram selecionados para serem executados na FPGA. Esses *Kernels* são então alocados na CPU, tal que o Makespan achado na Fase 1 *Geração de Configuração da FPGA* não seja aumentado. Seguindo as seguintes etapas:**a)** avalia se o Makespan da FPGA é maior do que o apresentado pelo core da CPU. Se esse for o caso, e a estratégia tem como objetivo Makespan (GMK), vai para a etapa B. Se não, a estratégia tem como objetivo energia (GMK-E), e vai para etapa c; **b)** aloca os *Kernels* remanescentes para os cores da CPU tal que o Makespan não aumente. Isso é atingido por alocar cada *Kernel* na CPU com menor Makespan; vai para etapa d; **c)** atribui *Kernels* para a CPU se eles consumirem menos energia do que quando alocados na próxima *Geração de Configuração da FPGA*;**d**) gera um Arranjo de Kernels e remove os *Kernels* já atribuídos do *Batch* inicial. Se ainda existem *Kernels* a serem alocados, o algoritmo invoca a Parte 1 para produzir uma nova *Geração de Configuração da FPGA*. Se não, a *Geração Colaborativa* é criada.

5.4.2 First-Come First-Served (FCFS)

O algoritmo de FCFS é um algoritmo de complexidade polinomial, comumente aplicado em ambientes não colaborativos (MARPHATIA et al., 2013; HU et al., 2009; SUSILA; CHANDRAMATHI, 2016). Essa heurística distribui os *Kernels* sobre a FPGA e a CPU sem considerar nenhum modelo de ganho. Seguindo estas etapas:**a)** atribui os *kernels* na ordem de chegada para a FPGA conforme eles cabem. Se ainda existem *Kernels* não alocados, ele vai para a etapa b. Se não, ele vai para etapa d;**b)** distribui os próximos *Kernels* para os CPUs cores (um *Kernel* em cada). Se ainda existem *Kernels* remanescentes, vai para c. Se não, ele vai para d;**c)** verifica se o Makespan da CPU é maior que o da FPGA. Se for verdadeiro, cria o atual Arranjo de *Kernels* e os remove da lista de *Kernels* inicial todos os *Kernels* já atribuídos, então retorna para etapa a. Se não, ele para a etapa B;*d*) salva a *solução* atual e termina o algoritmo.

5.5 Passo 5 - Execução (Execution)

No último passo da execução ja foram gerados relatórios de saída para avaliação do funcionamento da *Framework*, estes relatórios podem ser usados em algum sistema externo do provedor de *Cloud* para realizar a análise necessária para saber quais os Objetivos de Otimização (α, β,γ) que devem ser utilizados como entrada na Etapa 1 (Seção 5.1) nas próximas execuções. Ainda nessa etapa é feito o retorno ao cliente que vai ter suas tarefas executadas no sistema colaborativo após os binários serem executados na CPU, e os Bitmaps carregados nas FPGAs, sendo nesta execução final percebida pelo cliente a melhoria de Qualidade do Serviço Fornecida pelas etapas anteriores da *Framework*.

6 METODOLOGIA

Para avaliar a *Framework* foi elaborado um sistema que representasse uma configuração real de sistema em *Cloud*. O ambiente é composto de um processador Intel Core i7-8700 e uma FPGA Xilinx Alveo U200 (Tabela 6.1). As ferramentas utilizadas para gerar as informações da biblioteca de *Kernels* no segundo passo e para gerar os relatórios de alocação no quarto passo da Framework foram: **perf tool**, para performance em CPU, **Intel RAPL library** para energia da CPU, **Vivado HLS** para síntese e performance em FPGA e **Xilinx Vivado** para implementação em FPGA e extração de potência. Os *Kernels* avaliados foram os mesmos utilizados no Capítulo 4, descritos em detalhes na seção 4.1, resultando em mais de 90 versões de *Kernels* entre os diferentes *Benchmarks*.

6.1 Cenários

Com o objetivo de reproduzir a carga de trabalho heterogênea de um provedor de *Cloud* real, foram consideradas 10 000 requisições de *Kernels*, como no trabalho (NGUYEN; KUMAR, 2020), para entrada do primeiro passo. Os *Batches* foram configurados de maneira a ter tamanho fixo de 1 000 *Kernels*, gerando 10 *Batches* de execução para cada cenário avaliado, descritos abaixo e na Tabela (6.2).

6.1.1 B.FCFS

É usada nos experimentos como baseline. Considera *Kernels* sem diretivas de síntese (*loop pipelining* e *loop unrolling*), e usa a estratégia de alocação FCFS (Primeiro a Chegar, Primeiro Servido) comumente aplicada em Ambientes *Cloud* (MARPHATIA et al., 2013; HU et al., 2009; SUSILA; CHANDRAMATHI, 2016).

Tabela 6.1: Ambiente de Avaliação

CPU Specifications		**FPGA Specifications**	
Model	Intel i7-8700	**Model**	Xilinx Alveo U200
Frequency	3.2 GHz	**BRAM**	4320
L1 Caches	32Kb Inst/32Kb Data	**DSP**	6840
L2 Cache	256Kb	**FF**	2364480
L3 Cache	12288Kb	**LUT**	1182240
Number of Cores	6	**Off-chip Bandwidth**	77 GB/s
TDP	65W	**PCI Express**	Gen3x16

6.1.2 B.GMK-M e B.GMK-E:

Esses cenários usam apenas a estratégia de alocação GMK, sem as vantagens da biblioteca de *Kernels*. Comparando-a com ***Multivers***, pode-se medir os benefícios de utilizar a biblioteca de *Kernels*

6.1.3 K.FCFS

Ao contrário do cenário anterior, este avalia apenas o potencial de usar a biblioteca de *Kernels*. Para isso, este cenário seleciona a melhor versão de *Kernel* de acordo com um objetivo de otimização de pesos equilibrados ($\alpha = \beta = \gamma = 1/3$), mas usando a estratégia de alocação mais simples: FCFS

6.1.4 MV 1-9

Nove cenários que utilizam ***Multivers*** com diferentes tuplas de otimização. Esses cenários selecionam o melhor *Kernel* da biblioteca para cada requisição de *Kernel* com respeito ao objetivo de otimização apresentado e então utiliza as estratégias de GMK-E ou GMK-M de acordo.

6.2 Regiões de Reconfiguração Parcial

Outra estratégia usada para reproduzir situações reais de ambientes em *Cloud* foi a consideração de duas situações: Uma onde 100% da FPGA está disponível para ser explorada pelo sistema ***Multivers*** e outra onde a apenas parte da FPGA está disponível para ser explorada pela *Framework*. Para a situação proposta em que a FPGA está apenas parcialmente disponível, foram elaboradas duas situações, uma em que a região disponível corresponde a 80% da área total da FPGA e outra onde a região corresponde a 20% da área

Tabela 6.2: Configurações de avaliação.

Scenario	B.FCFS	B.GMK-M	B.GMK-E	K.FCFS	MV 1	MV 2	MV 3	MV 4	MV 5	MV 6	MV 7	MV 8	MV 9
Area Weight	-	-	-	1/3	0	0	0	1/3	1/3	1/3	2/3	2/3	2/3
Perf. Weight	-	-	-	1/3	0	1/2	1	0	1/3	2/3	0	1/6	1/3
Energy Weight	-	-	-	1/3	1	1/2	0	2/3	1/3	0	1/3	1/6	0
HLS Synthesis Directives	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Allocation Strategy	FCFS	GMK-M	GMK-E	FCFS	GMK-E	GMK-E	GMK-M	GMK-E	GMK-E	GMK-M	GMK-E	GMK-E	GMK-M

da FPGA, com intuito de explorar como ***Multivers*** evolui conforme aumenta a restrição de recursos disponíveis para serem alocados em relação aos outros cenários descritos na Seção 6.1.

7 RESULTADOS

7.1 Avaliação de Diferentes Cenários de Otimização

A figura 7.1 apresenta a melhora em Makespan e a Energia de cada cenário com relação ao B.FCFS. O uso de ***Multivers*** resulta, em média, em melhorias de 1,42x e 4,78x em Energia e Makespan, respectivamente. Quando o cenário é configurado para focar completamente em performance (MV 3 com $\beta = 1$), a *Framework* atinge o ganho máximo em Makespan: 1,93x sobre o B.FCFS. Quando há equilíbrio entre os pesos de Energia e Performance no cenário MV 2 ($\beta = 1/2$ e $\gamma = 1/2$), os ganhos em Makespan são levemente reduzidos (de 1,93x para 1,40x), mas a os ganhos em energia aumentam significativamente (de 3.36x para 6.56x).

Isso acontece devido ao comportamento do algoritmo GMK-M, que evita reconfigurações frequentes da FPGA, alocando colaborativamente mais *Kernels* na CPU, que gasta mais energia que a FPGA. Para as PRRs 100%, a maioria dos cenários executando GMK-M necessita de 2 reconfigurações, enquanto alocações usando GMK-E, e B.FCFS, e K.FCFS necessitam 3,4 e 3 reconfigurações, respectivamente. Como resultado, pode-se ver uma melhora no Makespan ao custo do aumento da energia consumida em cenários GMK-M (MV 3, MV 6, MV 9, e B.GMK-M).

Para cenários focados em otimização de área (por exemplo, MV 7,8, e 9) ***Multivers*** apresenta ganhos pequenos quando a PRR ocupa 100% dos recursos da FPGA. Isso ocorre principalmente porque quando é considerado que toda a FPGA está disponível para a alocação de Kernels, o número de configurações necessárias não aumenta significativa-

Figura 7.1: Gráfico Energia e Makespan para os Cenários

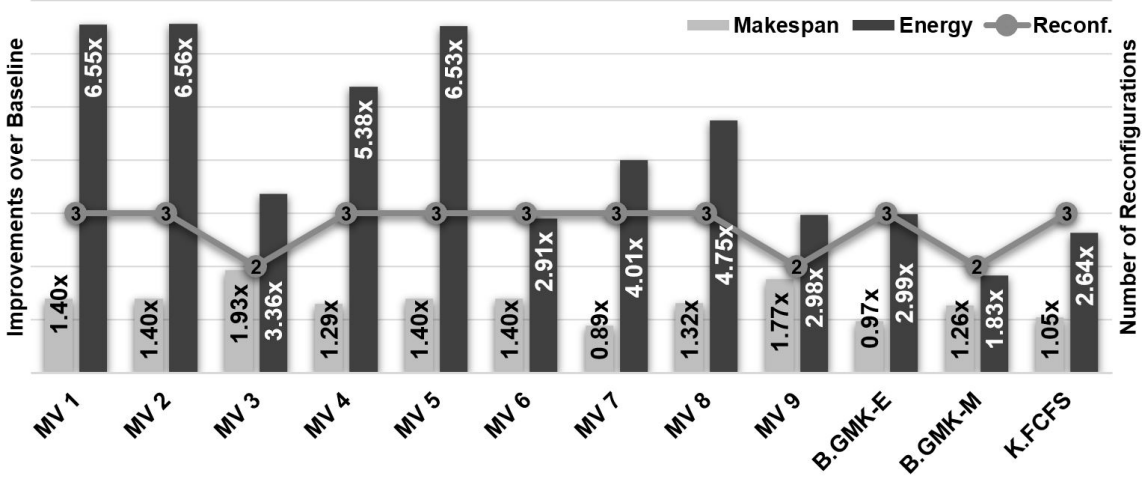


Tabela 7.1: Versão de Kernels para cada cenário MV comparando Performance e Energia sobre a versão sem diretivas (Versão 0).

MV	ADI			Floyd-W			MD5			MM			Pivot			RowCol			Seidel			TriSolv		
	Ver	P	E	Ver	P	E	Ver	P	E	Ver	P	E	Ver	P	E	Ver	P	E	Ver	P	E	Ver	P	E
1	0	1.00x	1.00x	7	1.19x	1.32x	7	1.17x	1.96x	14	16.04x	22.95x	1	1.000x	1.14x	5	3.18x	3.61x	3	8.86x	10.72x	8	2.11x	3.06x
2	0	1.00x	1.00x	8	1.46x	1.23x	7	1.17x	1.96x	14	16.04x	22.95x	1	1.000x	1.14x	5	3.18x	3.61x	3	8.86x	10.72x	8	2.11x	3.06x
3	1	1.01x	0.64x	8	1.46x	1.23x	17	1.37x	1.43x	3	17.70x	12.49x	4	1.001x	0.59x	1	3.51x	2.47x	3	8.86x	10.72x	8	2.11x	3.06x
4	0	1.00x	1.00x	5	1.20x	1.30x	6	1.17x	1.95x	3	17.70x	12.49x	1	1.000x	1.14x	3	3.34x	2.88x	1	1.22x	1.72x	8	2.11x	3.06x
5	0	1.00x	1.00x	5	1.20x	1.30x	3	1.09x	1.29x	3	17.70x	12.49x	1	1.000x	1.14x	3	3.34x	2.88x	3	8.86x	10.72x	8	2.11x	3.06x
6	1	1.01x	0.64x	8	1.46x	1.23x	3	1.09x	1.29x	3	17.70x	12.49x	0	1.000x	1.00x	1	3.51x	2.47x	3	8.86x	10.72x	8	2.11x	3.06x
7	3	1.01x	0.64x	5	1.20x	1.30x	3	1.09x	1.29x	1	2.28x	3.04x	1	1.000x	1.14x	3	3.34x	2.88x	1	1.22x	1.72x	15	0.83x	1.84x
8	3	1.01x	0.64x	5	1.20x	1.30x	0	1.00x	1.00x	1	2.28x	3.04x	1	1.000x	1.14x	3	3.34x	2.88x	1	1.22x	1.72x	8	2.11x	3.06x
9	1	1.01x	0.64x	5	1.20x	1.30x	0	1.00x	1.00x	1	2.28x	3.04x	0	1.000x	1.00x	1	3.51x	2.47x	1	1.22x	1.72x	1	2.10x	2.02x

mente, inclusive para os *Kernels* que exigem mais recursos. Com restrições mais rígidas na área, se torna essencial para o Makespan geral dos sistemas que seja considerada a utilização de recursos dos *Kernels* (discutido em mais detalhes na subseção **??**. Considerando cenários que focam 100% em Energia (MV 1 com $\gamma = 1$), Multivers consome 6.55x energia que o baseline B.FCFS. Mesmo em casos como MV7 em que o peso é completamente focado em performance ($\alpha = 2/3$, $\gamma = 0$, e $\gamma = 1/3$), ***Multivers*** acaba conseguindo ganhos em Makespan e ainda alcança uma melhora em Energia de 4.01x sobre B.FCFS.

Em diversos cenários onde alocações com GMK-E são utilizadas, observa-se ganhos elevados de energia com relação ao baseline B.FCFS. Contrariamente ao GMK-M, GMK-E evita despachar *Kernels* para execução de menor eficiência energética na CPU, usando mais reconfigurações da FPGA, como mostrado na Figura 7.1. Quando nenhum dos dois GMKs é empregado, no cenário K.FCFS, ainda se atinge uma redução de 2,64x graças às otimizações do 1º estágio (ver Capítulo 5). Porém, o K.FCFS apresenta o menor ganho em Makespan, devido ao FCFS gulosamente distribuir os *Kernels* entre FPGA e CPU.

Apesar de estratégias de alocação especializadas terem um impacto significativo nos resultados, o versionamento de *Kernels* é uma etapa fundamental do *Framework*. A Tabela 7.1 caracteriza os *Kernels* gerados por HLS para cada *Benchmark*. É mostrado a versão de *Kernel* de acordo com pesos de performance, área e energia para cada cenário MV, e as melhorias em performance e energia de cada versão sobre a versão baseline, o *Kernel* sem diretivas (Versão 0). Por exemplo, a **versão 8** é ideal para o *Kernel* Floyd-Warshall no cenário MV 3, apresentando um ganho de performance de 1.46x e um ganho de energia de 1.23 sobre a versão baseline.

Como pode ser observado, em todos cenários MV a maioria dos *Benchmarks* apresenta alta variabilidade em performance e energia, habilitando um Exploração Extensiva de Espaço de Projeto para a *Framework* ***Multivers***. E ainda, seis dos oito *Benchmarks*

apresentam ganhos em performance ou energia comparado com a versão baseline (sem anotações de síntese). Devido à diversidade das aplicações em Cloud, nem todas as aplicações atingem ganhos significativos nos *Kernels* do espaço multi-kernel. Por exemplo, ADI e Pivot não têm ganhos consideráveis em versões de configurações orientadas a performance. Esse *Kernels* então se tornam candidatos a serem executados na CPU no ambiente colaborativo de alocação que ***Multivers*** proporciona.

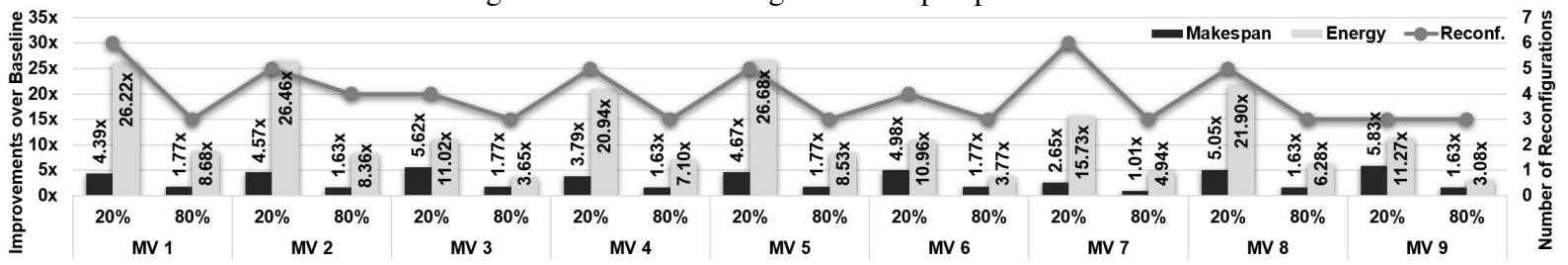
É possível perceber o impacto de aplicar diferentes versões de *Kernels* quando se compara B.GMK-E e B.GMK-M contra cenários MV na Figura 7.1. Apenas aplicando o segundo estágio de otimização nos cenários B.GMK-E e B.GMK-M, existem melhoras de até 2.99x e 1.26x em Energia e Makespan, comparado com 6.56x e 1.93x fornecidos pela combinação de versionamento de *Kernels* e algoritmos de alocação especializados (Cenários MV). *Isso demonstra o quanto é essencial a etapa extra de escolher a versão mais otimizada de um* Kernel *aumentando o Espaço de Exploração e extrair a máxima eficiência das estratégias de alocação com GMK*. Na próxima seção é avaliado o impacto do versionamento de *Kernels* e alocação em PRRs de menor tamanho.

7.2 Influência dos tamanhos das PRRs

Na Figura 7.2, são mostrados os mesmos 9 cenários MV para avaliar Multivers em PRRs de 20% e 80% da área da FPGA. É possível perceber que ***Multivers*** alcança melhor resultado para menores PRR. Quando o tamanho da PRR é reduzido para 20% da FPGA, ***Multivers*** atinge melhorias de em média 19.04x e 4.62x em Energia e Makespan, respectivamente. Essas melhorias se reduzem para 6.04x e 1.62x quando consideramos a PRR ocupando 80% da FPGA. Isso é explicado por duas razões: primeiro, a seleção correta de Kernels tem um grande impacto, visto que há menos área disponível e isso pode resultar em menos reconfigurações da FPGA; e, segundo, diferentemente do algoritmo de alocação FCFS (usado em B.FCFS e K.FCFS), o passo de Otimização Colaborativa com GMK garante o despacho colaborativo de *Kernels* para CPU, sem aumentar o Makespan global (GMK-M), e com mínimo impacto energético (GMK-E)

Como pode-se perceber, quando ***Multivers*** é configurado para focar completamente em performance (MV 3 com $\beta = 1$) é atingido o maior ganho em Makespan para uma PRR ocupando 80% da FPGA. Porém, o mesmo não se aplica para a PRR ocupando 20% da FPGA, onde a melhora mais significativa em Makespan ocorre no cenário MV 9 (com $\alpha = 2/3$ e $\beta = 1/3$). Quando o tamanho da PRR disponível é reduzido, se ob-

Figura 7.2: Gráfico Energia e Makespan para PRRs

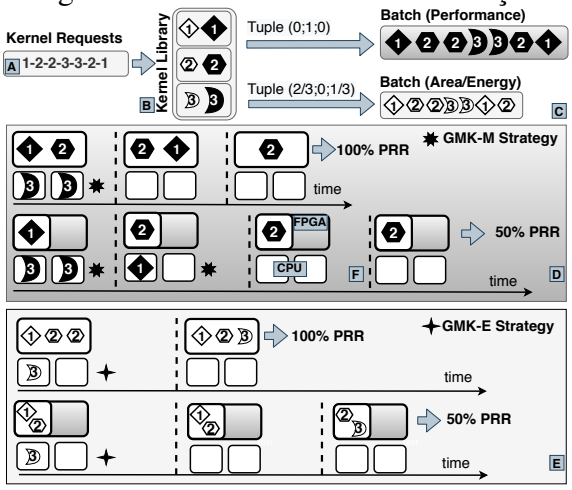


serva que o peso dado à área, assim como o peso dado à performance, tem maior impacto no Makespan da aplicação (por exemplo MV 3,6,9). Isso ocorre principalmente porque selecionar Kernels que ocupam menos área possibilita que mais *Kernels* sejam alocados na mesma configuração de FPGA, aumentando o paralelismo, e consequentemente implicando em menos reconfigurações da FPGA. Para MV 5, onde ambos performance e área são priorizadas, a PRR ocupando 20% da FPGA apresenta o maior ganho em energia (26.68x).

7.2.1 Análise Crítica

Conforme discutido anteriormente, ganhos em Makespan e Energia podem variar com os tamanhos das PRRs presentes nas FPGAs do sistema em *Cloud* e pesos da tupla de otimização. Para entender tal comportamento, a Figura 7.3 demonstra o fluxo de alocação de ***Multivers*** para sete requisições de *Kernel* (Fig 7.3.A). Existem três diferentes *Kernels* dentre as requisições, cada destas duas versões sintetizadas, as quais estão armazenadas na Biblioteca de *Kernels* (Fig. 7.3.B). Nesse exemplo são executadas as mesmas requisições de *Kernels* considerando dois objetivos de otimização diferentes: 1) Foco completamente em performance; e 2) Foco em divídido entre Energia-Área (Fig. 7.3.C). Para execuções focadas completamente em performance, apenas *Kernels* com menor tempo de execução são escolhidos, enquanto para execuções área-energia, apenas *Kernels* que equilibram requisitos de baixa ocupação de área e baixo consumo energético são escolhidos, desta forma então construindo dois *Bathches* diferentes.Ambos os *Batches*, focado em performance e focado área-energia, são executadas em um ambiente colaborativo CPU-FPGA em *Cloud* (Fig. 7.3.F) considerando PRRs de 100% (Fig. 7.3.D) e 50% da FPGA (Fig. 7.3.E). Como discutido anteriormente, configurações focadas totalmente em performance utilizam algoritmo GMK-M, enquanto os outros utilizam o algoritmo de alocação GMK-E.

Figura 7.3: Análise Crítica de Alocação



Como pode ser percebido, executando a otimização completamente focada em performance para PRR ocupando 100% da FPGA, o resultado é o de no menor Makespan entre todos os cenários. Isso acontece devido a mesmo os *Kernels* de alta performance (alta demanda de recursos) terem recursos o suficiente para serem executados em paralelo, necessitando menos reconfigurações da FPGA. Por esta razão, cenários MV que não tem nenhuma restrição de área conseguem entregar melhor performance, e os ganhos locais de cada instância podem ser executados sem muito prejuízo ao paralelismo devido ao tamanho dos *Kernels*, mostrando, assim, a importância do primeiro estágio de otimização. Além disso, Kernels de baixa aceleração são despachados em paralelo para a CPU, pelo uso de alocação do algoritmo de GMK-M, dessa forma não afetando o Makespan global.

Porém, quando consideramos um cenário com 50% da FPGA disponível, diversas reconfigurações são necessárias, devido aos pesados requisitos para área destes *Kernels*. Para conseguir prover melhorias no Makespan nesse cenário de recursos escassos, o algoritmo de GMK-M despacha mais Kernels para a CPU, aumentando a energia global consumida. Por outro lado, quando o cenário da tupla de otimização é focado em energia-área, para a PRR ocupando 100% da FPGA, o Makespan é aumentado significativamente, pois mesmo que conseguindo evitar reconfigurações de FPGA, os *Kernels* requisitados demandam de um longo tempo de execução. Além disso, a estratégia de alocação GMK-E apenas despacha *Kernels* para a CPU caso eles não aumentem o consumo global de energia, o que gera necessidade de grandes recursos de área pejudicando ambos resultados em performance e energia quando grandes quantidades de área estão disponíveis. Porém quando a área disponível na FPGA é reduzida (PRR ocupando 50%), o uso de configurações focadas em energia-área pode se tornar vantajoso para: Makespan, devido a menos reconfigurações da FPGA serem necessárias devido aos baixos requisitos de área

dos *Kernels* quando se aumenta o peso da área no primeiro estágio de otimização; e Energia, pois o menor despacho *Kernels* para CPU devido a estratégia de alocação utilizando o algoritmo GMK-E durante o segundo estágio de alocação.

8 CONCLUSÃO

Este trabalho se torna relevante para perceber-se que existe um grande custo energético e de Makespan para grandes servidores em Cloud. ***Multivers*** inteligentemente explora o multiversionamento de *Kernels* gerado a partir de síntese de alto-nível para alocação de *Kernels*, diminuindo significativamente o Makespan (em até $4.62\times$ em média para PRRs de 20%) e energia (em até $19\times$ em média para PRRs de 20%) em sistemas de Cloud baseados em FPGA, quando comparado com *Kernels* gerados a partir de síntese de alto-nível sem otimização e alocados com algoritmos tradicionais para estratégias de alocação em sistemas de Cloud.

Como continuação a esse trabalho, serão desenvolvidos mecanismos que permitam a ***Multivers*** trabalhar sem uma entrada explícita de metas de otimização pelo provedor de Cloud, detectando as necessidades automaticamente.

REFERÊNCIAS

GUSTAFSON, J. L. Reevaluating amdahl's law. **Communications of the ACM**, ACM New York, NY, USA, v. 31, n. 5, p. 532–533, 1988.

HU, Y. et al. Resource provisioning for cloud computing. In: **CASCON**. [S.l.: s.n.], 2009. p. 101–111.

HUANG, S. et al. Analysis and modeling of collaborative execution strategies for heterogeneous cpu-fpga architectures. In: ACM. **ICPE**. [S.l.], 2019. p. 79–90.

JAN, M. R. et al. **Digital integrated circuits: a design perspective**. [S.l.]: Prentice Hall Upper Saddle River, NJ, 2003.

Jordan, M. G. et al. Resource-aware collaborative allocation for cpu-fpga cloud environments. **IEEE Transactions on Circuits and Systems II: Express Briefs**, p. 1–1, 2021.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. Multidimensional knapsack problems. In: **Knapsack problems**. [S.l.]: Springer, 2004. p. 235–283.

KHANH, P. N. et al. Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis. In: **DATE**. [S.l.: s.n.], 2015. p. 157–162.

LIU, J.; BAYLISS, S.; CONSTANTINIDES, G. A. Offline synthesis of online dependence testing: Parametric loop pipelining for hls. In: IEEE. **FCCM**. [S.l.], 2015. p. 159–162.

Louis-Noel Pouchet. **PolyBench/C: the Polyhedral Benchmark suite**. 2019. Http://web.cse.ohio-state.edu/ pouchet.2/software/polybench/.

MARPHATIA, A. et al. Optimization of fcfs based resource provisioning algorithm for cloud computing. **IOSR-JCE**, v. 10, n. 5, p. 1–5, 2013.

NGUYEN, T. D.; KUMAR, A. Maximizing the serviceability of partially reconfigurable fpga systems in multi-tenant environment. In: **FPGA**. [S.l.: s.n.], 2020. p. 29–39.

RIVEST, R. **RFC1321: The MD5 message-digest algorithm**. [S.l.]: RFC Editor, 1992.

SAFE, M. et al. On stopping criteria for genetic algorithms. In: SPRINGER. **SBIA**. [S.l.], 2004. p. 405–413.

SHAN, J. et al. Exact and heuristic allocation of multi-kernel applications to multi-fpga platforms. In: **DAC**. [S.l.: s.n.], 2019. p. 1–6.

SUSILA, N.; CHANDRAMATHI, S. Energy efficient extended fcfs load balancing in data centers of cloud. **IJAER**, v. 11, n. 1, p. 599–605, 2016.

WEI, X. et al. Throughput optimization for streaming applications on cpu-fpga heterogeneous systems. In: IEEE. **ASP-DAC**. [S.l.], 2017. p. 488–493.

XILINX.COM. <https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/jyk1504034310030.html>. Acessado: 28-02-2021.

ZHAO, J. et al. Performance modeling and directives optimization for high level synthesis on fpga. **IEEE TCAD**, IEEE, 2019.

ZHONG, G. et al. Lin-analyzer: a high-level performance analysis tool for fpga-based accelerators. In: IEEE. **(DAC)**. [S.l.], 2016. p. 1–6.

ZHONG, G. et al. Design space exploration of fpga-based accelerators with multi-level parallelism. In: IEEE. **(DATE)**. [S.l.], 2017. p. 1141–1146.

APÊNDICE A — TRABALHO DE GRADUAÇÃO 1

MultiVers - Exploração Dinâmica de Espaço de Projeto para Sistemas CPU-FPGA em Cloud Utilizando Síntese de Alto Nível

Bernardo Neuhaus Lignati[1], **Antonio Carlos Schneider Beck Filho**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre – RS – Brazil

{bnlignati,caco}@inf.ufrgs.br

Resumo. *Sistemas de servidores em* Cloud *têm mostrado cada vez mais importância comercial dentro do mundo da computação. Estes vêm explorando sistemas de execução colaborativa CPU-FPGA onde múltiplos clientes compartilham a mesma infraestrutura para maximizar a eficiência energética e escalabilidade, além de, em alguns casos, aumentar a qualidade de serviço percebida pelo usuário. Porém, o fornecimento de recursos é um desafio nestes ambientes, pois* Kernels *podem ser despachados para ambos, CPU e FPGA, concorrentemente numa grande variabilidade de cenários em termos de disponibilidade de recursos e características de carga de trabalho. Este trabalho primeiramente realiza experimentos para analisar a amplitude desse espaço de exploração e como diferentes versões de um mesmo* Kernel *acabam gerando melhores resultados, dependendo dos parâmetros escolhidos para avaliação.*
Decorrente desta análise, é proposto **Multivers***, uma* Framework *que aproveita o método de Síntese de Alto Nível para permitir maiores ganhos em tais sistemas colaborativos CPU-FPGA.* **Multivers** *explora vantagens da geração automática por Síntese de Alto Nível para produzir diferentes versões de cada requisição de entrada para* Kernels*, aumentando significativamente a exploração do espaço de projeto disponível e passível de otimização pelas estratégias de alocação do provedor de* Cloud*. Além de possuir a biblioteca gerada, que permite uma seleção de* Kernels *a partir de uma interface que permite a comunicação das necessidades atuais do Servidor de* Cloud*,* **Multivers** *também permite que o multi versionamento de* Kernels *e as estratégias de alocação trabalharem juntos, permitindo ajuste fino em termos de utilização de recursos, performance e energia; ou qualquer combinação destes parâmetros.*

Abstract. *Cloud server systems have been growing in commercial importance in the computational field. These systems have been exploring CPU-FPGA collaborative systems in which multiple clients share the same infrastructure to maximize the energy efficiency and scalability and, in some cases, increase the quality of service perceived by the users. However, the providing of resources is a challenge in these environments. The Kernels can be dispatched to run in both CPU and FPGA, concurrently generating a great variety of scenarios in terms of resources and workload characteristics. This work is composed first by experiments to analyze the amplitude of this design space and how different versions of the same Kernel generate different results, according to the parameters chosen to be evaluated.*

From this analysis, **Multivers** *is proposed, a Framework that leverages High-Level Synthesis to allow higher gains in such CPU-FPGA collaborative systems.* **Multivers** *exploits advantages of automatic High-Level Synthesis to generate different versions of each input Kernel request, greatly enlarging the available design space exploration passive of optimization by the allocation strategies in the cloud provider. Besides containing a generated library that allows the selection of Kernels to fit the Cloud provider's current needs,* **Multivers** *makes both kernel multi-versioning and allocation strategy work symbiotically, allowing fine-tuning in terms of resource usage, performance, energy, or any combination of these parameters.*

1. Introdução

Por muitos anos, a melhora no desempenho computacional foi governado pela Lei de Moore, que previa que os dispositivos dobravam sua performance em razão de duas vezes, devido a melhorias no processo de fabricação de dispositivos semicondutores. Com o fim da Lei de Moore, que veio devido às limitações físicas que impediram que os processos de fabricação continuassem a evoluir no mesmo ritmo, novas características arquitetônicas, voltadas ao paralelismo, começaram a ser exploradas, para conseguir novamente se atingir melhora na velocidade de computação. Porém, essa exploração de paralelismo usando processadores tradicionais têm limitações matemáticas bem definidas pelas Leis de Amdahl e Gustafson [Gustafson 1988].

Assim, surgiram estratégias utilizando dispositivos não convencionais para buscar melhora na performance dos sistemas. Nestes, as computações não executam mais em um processador de organização e arquitetura genérica, mas são despachadas para executarem em um circuito digital de arquitetura específica que, em troca da perda de generalidade, para algumas aplicações consegue gerar melhores resultados em performance. Para que se possa implementar esses circuitos, tecnologias que permitem a implementação de circuitos reconfiguráveis como as FPGAs (seção 2, subseção 2.1) acabam sendo uma ótima opção, pois além de apresentarem melhora no desempenho para algumas aplicações, para outras conseguem reduzir o consumo energético, outro ponto fundamental na computação moderna.

Em paralelo a isto, os sistemas de computação em larga escala, como servidores de Cloud (seção 2, subseção 2.2), vêm apresentando a necessidade da utilização das estratégias de aceleradores específicos, como por exemplo para execução de operações matemáticas complexas e de redes neurais: devido à alta demanda de processamento, estes se beneficiam dos acelerados para conseguir entregar uma melhor qualidade de serviço ou para reduzir gastos energéticos que são consideráveis nesse tipo de sistema.

Assim, os aceleradores de propósito específico baseados em FPGA tem se tornado cada vez mais viáveis, pois a indústria tem migrado de um fluxo tradicional que tem um alto *time to market* para um modelo de desenvolvimento mais moderno, baseado em HLS (Síntese de Alto Nível)(seção 2, subseção 2.3). Com HLS, o circuito não é mais descrito fisicamente como em fluxos de HDL (Linguagem de Descrição de Hardware) tradicionais, mas compilado a partir de uma descrição algorítmica usando linguagens de alto nível.

Surge, a partir desse novo modelo de desenvolvimento baseado em HLS, um problema aberto à exploração de espaço de projeto (capítulo 4). Neste, através de uma mesma

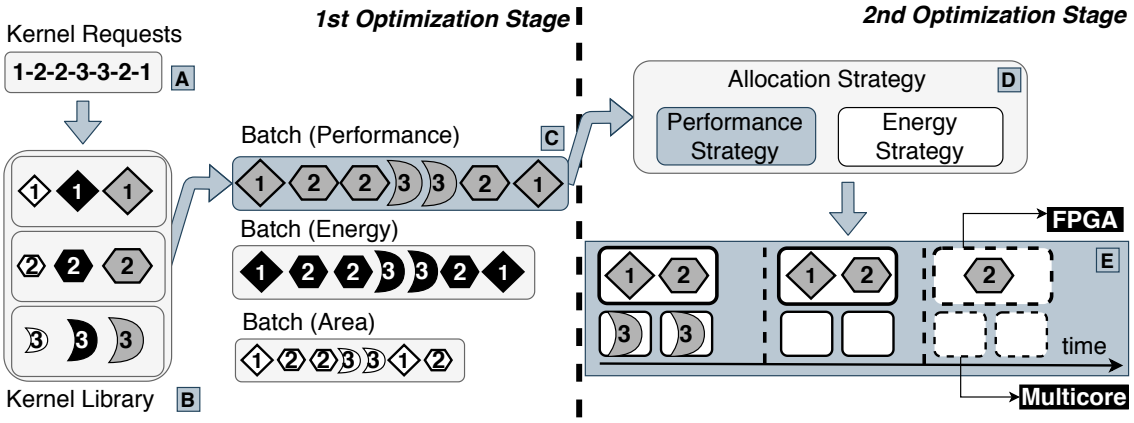


Figure 1. Visão Geral da Ideia Funcionamento da Framework

descrição algorítmica, pode-se gerar diferentes arquiteturas para um mesmo acelerador. Estas diferenciadas por anotações de código que permitem a exploração de diferentes técnicas de desenvolvimento de circuitos digitais com alta velocidade de projeto.

A ideia deste projeto é apresentar a contextualização e análise inicial necessários para avaliar se existe indícios necessários de que um sistema utilizando HLS para conseguir obter melhor resultados em Cloud é interessante de ser implementado. Pelos resultados parciais obtidos na seção 4 mostra-se que existe um espaço de exploração de projeto nas métricas descritas como importantes para sistemas em Cloud: Energia e Makespan, dando assim fortes indícios de que tais sistemas poderão obter bons resultados.

2. Background

Nesta seção será discutida a fundamentação teórica necessária para execução deste trabalho, baseada em conceitos já fundamentados na academia e na indústria. Para que fosse desenvolvido este trabalho se baseou fortemente na utilização de FPGAs, como aceleradores reconfiguráveis subseção 2.1,Sistemas em Cloud, como ambiente alvo subseção 2.2, Síntese de Alto Nível, como ferramenta para execução de projeto de hardware subseção 2.3. Nas próximas subseções estes tópicos serão discutidos em detalhe.

2.1. Field Programmable Gate Arrays

FPGAs (Field Programmable Gate Arrays) são circuitos eletrônicos que podem ser programados após sua confecção. Muito utilizados em prototipagem de hardware, as FPGAs acabaram se tornando também ferramentas poderosas para aceleração de algumas aplicações em hardware, devido ao seu baixo custo unitário e seu alto poder de reconfiguração.

Consistindo de um denso arranjo de elementos lógicos, como os descritos na subseção 2.1.3, a FPGA consegue representar funções lógicas, que implementam descrições de hardware, para assim fazer a execução de forma paralela e orientada ao de fluxo de dados dos algoritmos. Abaixo serão discutidas brevemente as etapas do fluxo de projeto em FPGA.

2.1.1. Configuração

Descrição:

A FPGA implementa uma descrição de um circuito lógico utilizando alguma ferramenta de descrição de Hardware, como por exemplo uma linguagem de descrição ou alguma ferramenta gráfica (como de projeto em diagrama de blocos). Esse tipo de linguagem de descrição tem a característica de ser paralela ou poder ser compilada para uma representação paralela, desta forma explorando as características de paralelismo e fluxo orientado a dados característico dos algoritmos executados em hardware.

Síntese:

A fase de síntese consiste na tradução da descrição fornecida na etapa anterior, para alguma representação lógica que será mapeada para os componentes disponíveis na FPGA descritos na subseção 2.1.3. Na etapa de síntese já existe uma dependência com a placa alvo, já que o algoritmo será mapeado para um conjunto específico de componentes, presentes naquela placa ou família de placas.

Implementação:

Na implementação das representações mapeadas na síntese, os componentes virtuais são distribuídos em componentes físicos específicos na topologia da placa, de forma a representar o design que será de fato fisicamente implementado. Nessa etapa, já com o posicionamento elaborado, é feito o roteamento entre os diferentes componentes, permitindo então a construção do fluxo de dados. Nessa etapa existe uma complexa iteração dentro das etapas de roteamento de forma otimizar o resultado final.

Carregamento de Bitmap:

Na última etapa da configuração, a implementação é traduzida de um computador *Host*, que é utilizado para as etapas anteriores para um arquivo de representação binária (*Bitmap* ou *Bitstream*), e então transferido para as memórias de configuração da placa alvo, por meio de uma interface de programação. Existem hoje sistemas FPGAs mais elaborados que permitem uma configuração dinâmica e de parte parcial da placa.

2.1.2. Execução

As FPGAs podem ser utilizadas para execuções de forma embarcada ou como coprocessadores de um sistema computacional complexo: este segundo modelo é o de interesse para trabalhos focados em Cloud.

Embarcada:

A FPGA embarcada pode desempenhar tarefas como: um circuito integrado desempenhando uma tarefa específica independentemente (*Standalone*) ou com a FPGA trabalhando em um sistema embarcado heterogêneo, normalmente comunicando-se com um microcontrolador ou processador de características embarcadas. Em ambos os casos a FPGA precisa de alguma maneira se comunicar com uma interface externa, e em ambos os casos a comunicação pode ser feita por dispositivos de entrada e saída conectados a barramentos no circuito integrado na FPGA; sendo que, no segundo caso, esta comunicação

também pode ocorrer por intermédio do microcontrolador/processador associado.

Coprocessador:

Existe atualmente um aumento na utilização de FPGAs como coprocessadores em sistemas computacionais complexos, como os de Computação de Alta Performance e Sistema *Multi-tenant* em *Cloud*. Isso acontece devido à alta capacidade de reconfiguração das FPGAs quando comparadas a outras opções para se implementar hardware e a sua característica arquitetural de processamento em paralelo e orientado a fluxo, que as tornam uma alternativa interessante para substituir a CPU na execução de algumas tarefas. Nesses casos as FPGAs se comunicam com outros nós do sistema heterogêneo (normalmente processadores), descarregando parte das computações deste e devolvendo o resultado final de volta para os processadores.

2.1.3. Componentes

LUT (Look-Up Table):

As Look-Up-Tables são os componentes básicos de uma lógica programável em FPGA, com circuitos armazenadores de valores (Memórias) de tamanho pequeno e multiplexadores, as LUTs conseguem implementar funções lógicas. Essas implementações são realizadas pela entrada se valores nos bits de controle do multiplexador e a tabela verdade da função armazenada nos circuitos de memória vinculados a cada entrada deste multiplexador, estando assim na saída disponível o valor da função dado uma certa linha da tabela verdade codificada nos bits de controle.

DSP (Digital Signal Processor):

É um processador especializado no cálculo de operações aritméticas, e pode ser encontrado embutido dentro da lógica programável das FPGAs. Pode ser roteado dentro do fluxo de dados dentro do chip da FPGA (diretamente junto aos outros componentes) de forma executar operações matemáticas de maior complexidade com menor prejuízo na performance, e se torna mais importante quando a FPGA está desempenho tarefas mais complexas e computacionalmente pesadas.

FF (Flip-flop:)

São as unidades básicas de memória em FPGA e em diversos componentes de hardware. Nas FPGAs possuem uma alta granularidade de configuração. Os Flip-flops têm as funções nas FPGA de implementar valores de único bit, registradores, e podem ser agrupados para formar blocos de memória em tamanho menor, ou também com características diferentes dos blocos rígidos de memória disponíveis para serem programadas em uma FPGA: as BRAM, descritas abaixo.

BRAM(Block-RAM):

A BRAMs, Memórias RAM em Bloco, são blocos de memória RAM rígidos que estão presentes dentro da lógica programável da FPGA e podem ser configuradas para funcionar junto ao fluxo de dados. Geralmente são uma memória RAM composta de duas portas e são implementadas para partes do algoritmo que tenham grandes requisitos de memória. Apesar de serem blocos rígidos, eles podem ter certa flexibilidade em algumas

características que podem ser configuradas.

Componentes Acoplados a Lógica Programável:

Muitas FPGAs apresentam componentes acoplados fora da logica embarcada, mas dentro do mesmo pedaço de silício. Como, por exemplo, FPGAs que acoplam processadores dentro do mesmo Chip para utilização em da categoria descrita em 2.1.2, outros processadores como GPUs, ou ainda circuitos lógicos rígidos para operações de Entrada e Saída complexas, como conversão Analógica-Digital/Digital Analógica. Este tipo de FPGAs vem crescendo em quantidade e variedade nos últimos anos.

2.2. Aceleração em Hardware e Sistemas em Cloud

Os grandes núcleos de computação como provedores de Cloud e centros de Computação de Alta Performance se caracterizam por ter uma pesada carga de trabalho. Devido a isto uma estratégia muito poderosa para poder atingir melhores resultados são arquiteturas heterogêneas de computadores, que utilizam coprocessadores especializados para tarefas especificas como, por exemplo, as GPUs, que são utilizadas para processamento gráfico otimizado e outras aplicações com paralelismo regular. Também, aceleradores baseados em circuito integrados específicos de aplicação, como aceleradores de rede neural; além de aceleradores diversos baseados em FPGA, que têm ganhado cada vez mais espaço devido a combinar as vantagens de desempenho de um circuito especifico com flexibilidade por causa da alta capacidade de reconfiguração.

Para sistemas de cloud, essa alta capacidade de reconfiguração é muito interessante, pois há uma grande diversidade de clientes realizando requisições ao provedor simultaneamente, de forma que a carga de trabalho se torna muito heterogênea. Ao mesmo tempo que não se tem apenas uma grande tarefa ocupando grandes porções do servidor, mas sim uma quantidade alta de pequenas tarefas que se dividem com uma maior granularidade no servidor. Para isso, um recurso das FPGAs modernas - PRR (Regiões Reconfiguráveis Parciais) - são muito uteis, pois se consegue dividir a FPGA entre regiões que podem ser acessadas e reconfiguradas separadamente como dispositivos menores.

A aceleração tem como objetivo melhorar duas métricas que se tornam cruciais nos sistemas de Computação de Alta Performance e Cloud, relacionadas ao desempenho e ao consumo energético: latência, potencia, vazão, energia consumida pelo sistema e Makespan. Esses dois últimos foram considerados como cruciais para a boa qualidade de serviço de sistemas de Cloud, Multivers sendo as métricas utilizadas e são explicados em

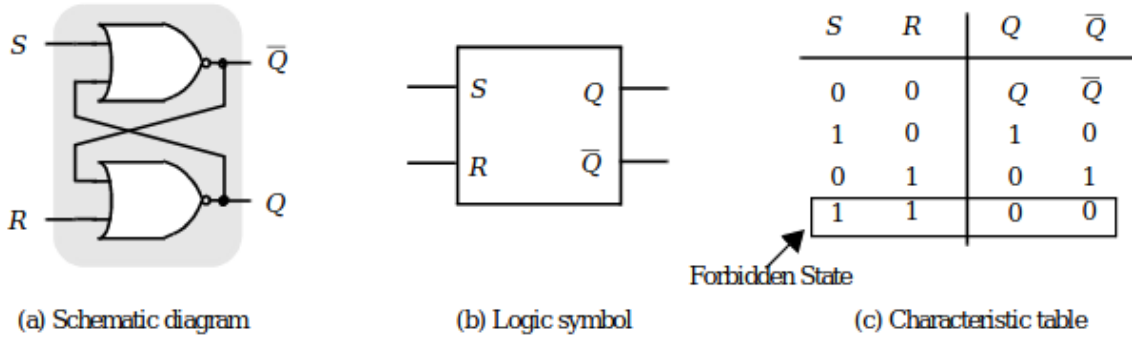


Figure 2. Flip-Flop[Jan et al. 2003]

detalhes nas subseções 2.2.1 e 2.2.2.

Para que as tarefas sejam escalonadas, usa-se a divisão delas e ***Kernels*** de computação, que são partes de algoritmos ou algoritmos que serão computados e distribuídos entres os nós de aplicações. Os *Kernels* que mais se beneficiam de aceleração em FPGA, são os que possuem alto paralelismo, com baixa dependência de dados interna e com caraterística de ser orientada a dados. Por exemplo, existem diversas aplicações nos campos de Matemática, Simulações Físicas, Criptografia e Grafos que se beneficiam deste tipo de aceleração.

2.2.1. Makespan

Makespan o Makespan é uma métrica definida como o tempo total entre o início e o fim de um trabalho. Esta definição acaba sendo a melhor a ser usada para avaliar o desempenho temporal de sistemas de escalonamento e grandes servidores, pois ele mede o tempo de execução do início da primeira tarefa solicitada até o tempo de conclusão da última tarefa solicitada. Visto isso o *textbfMakespan* acaba sendo muito importante para medir o tempo de execução percebido internamente pelo próprio provedor, mas também ao mesmo tempo tendo impacto direto na qualidade de serviço percebida pelo usuário.

2.2.2. Energia

A ***Energia*** tem sido cada vez mais importante como métrica de avaliação em grandes servidores. Apesar de ser uma métrica tradicional em Sistemas Embarcados, ela se torna significativa em grandes servidores que têm: a) Um grande de trabalho; e b) Sistemas de alto-desempenho que trabalham perto de seus limites físicos. Esses dois fatores juntos acabem gerando uma grande dissipação de potência de ao longo do tempo de serviço desses computadores ou núcleos de computadores, ou seja, consumo energético. Muito têm se feito para mensurar e mitigar esse grande consumo, pois existe um alto valor econômico associado a altos gastos de energia, dado esse valor econômico associado, a Energia é percebida neste trabalho como um dos grandes fatores a ser considerados quando se projeta sistemas e arquiteturas para o provedor de *Cloud*, buscando-se reduzir o consumo energético, mas sempre levando em conta os seus impactos na qualidade de serviço oferecida e observando a relação entre o tempo de execução e potência que são os fatores que compõem o gasto energético final, pois a energia consumida é dependente da potência e do tempo de execução como mostrada na equação 1.

$$EnergiaConsumida = \int_{T1}^{T1} P\,dt \tag{1}$$

2.3. Síntese de Alto Nível

Os fluxos para desenvolvimento clássico de hardware têm longo tempo de desenvolvimento e alto grau de complexidade para iteração do design, o que leva a um *time-to-market* mais longo. O fluxo de síntese de alto-nível é uma alternativa para abstrair algumas das etapas necessárias para o desenvolvimento de hardware reduzindo o tempo de desenvolvimento.

Diferentemente das estratégias clássicas, como desenvolvimento em HDLs, que usam uma descrição em blocos das entidades físicas que compõem o circuito digital, o fluxo HLS explora o uso de um compilador de alto nível. Em HLS, uma descrição em uma linguagem de alto nível (por exemplo C) do problema é fornecida para o compilador. O compilador, então, gera da descrição algorítmica uma descrição em forma de circuito digital, abstraindo do projetista a etapa de projeto em blocos de descrição de baixo nível.

Porém, ao se desenvolver hardware existem diversas soluções equivalentes para o mesmo problema, tornando possível o desenvolvimento de diversos circuitos funcionalmente equivalentes com diferentes arquiteturas. Isso torna possível que se explore diversos níveis de paralelismo espacial ou profundidade em emphpipelines. Essas diferenças na arquitetura do hardware proporcionam diferentes resultados em potência, energia, desempenho, utilização de recursos e outras restrições. No fluxo tradicional de desenvolvimento com HDLs, essas arquiteturas teriam de ser individualmente projetadas e implementadas pelo desenvolvedor de hardware, enquanto nos compiladores de HDL modernos, o mesmo código pode ser sintetizado para arquiteturas distintas usando anotações de alto nível, tornando possível gerar diversas versões de hardware utilizando praticamente o mesmo código.

Além de permitir a rápida exploração do espaço de projeto gerando diversas versões de hardware de alto nível, os compiladores modernos de síntese de alto nível proporcionam também diversas informações geradas em tempo de síntese que possibilitam a avaliação das métricas de cada uma das versões, para que então o projetista possa escolher a que melhor se adapta ao seu projeto. Abaixo serão discutidos em maior detalhe dois tipos de modificações arquiteturais provenientes de anotações no código.

2.3.1. Loop Unrolling

A técnica de ***Loop Unrolling*** consiste em desenrolar laços. Isto é, a partir de laços que seriam iterados diversas vezes, e repetidos por meio de uma configuração de controle, são transformados em várias instâncias individuais separadas e iguais, compostas pelo equivalente ao que seria executado dentro de cada iteração do loop(laço). No caso do ***Loop Unrolling*** aplicado ao hardware, essas instâncias são executadas em paralelo, por estruturas iguais, explorando o paralelismo espacial para ganhar em performance, em detrimento da área necessária para sintetizar o circuito.

2.3.2. Loop Pipelining

Enquanto a técnica de ***Loop Unrolling*** 2.3.1 explora o paralelismo espacial, a técnica de ***Loop Pipelining*** explora o paralelismo temporal para melhorar a execução de laços. A técnica de *Loop Pipelining* consiste em dividir uma iteração em diversas etapas, e adiciona registradores para armazenar os resultados parciais da execução da instancia entre etapas. Desta forma, quando a computação de uma etapa é concluída, o registrador pode guardar o resultado parcial de forma a seguir para a próxima etapa, liberando a primeira etapa para executar uma nova iteração, enquanto a primeira iteração está sendo processada pela próxima etapa, fazendo-as ser executadas paralelamente no tempo, porém dentro do mesmo fluxo de dados e das mesmas unidades funcionais.

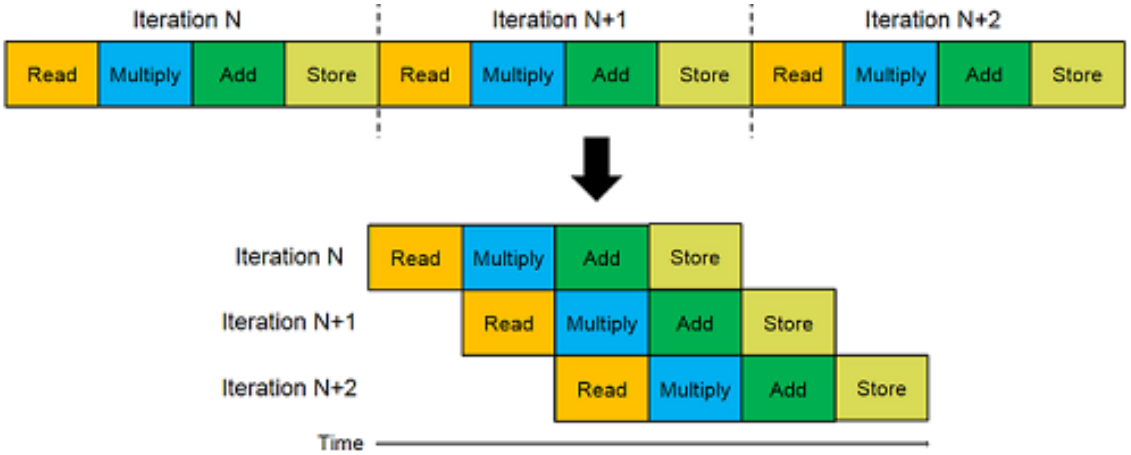


Figure 3. Loop Pipelining[xil]

3. Estado da arte

3.1. Computação Colaborativa

A Computação Colaborativa vem sendo proposta como uma maneira de tirar proveito da eficiência de múltiplas arquiteturas a partir da combinação de seus diferentes elementos computacionais. Isso é atingido particionando tarefas/*Kernels* entre diferentes dispositivos. [Huang et al. 2019] explora o potencial da execução colaborativa entre CPU e FPGA a partir da comparação de duas abordagens distintas: particionamento de tarefas e dados. Eles mostram que a estratégia correta de particionamento pode aumentar o ganho em performance em ambientes CPU-FPGA.

Os autores em [Wei et al. 2017] otimizam a vazão de aplicações de *streaming* em sistemas heterogêneos CPU-FPGA usando algoritmos de mapeamento de tarefas. Eles também empregam *pipelining* para melhor a vazão e frequência escalando para economia energética.

O trabalho proposto por [Shan et al. 2019] foca na otimização de mapeamento de algoritmos multi-kernel de alta performance para Redes Neurais e Aprendizado de Máquina para plataformas multi-FPGA. Escolhendo a melhor configuração de paralelismo para a aplicação multi *Kernel*, baseado na disponibilidade de recursos em cada dispositivo utilizando aproximação por Programação Geométrica. O algoritmo de alocação produz uma solução que otimiza a vazão da plataforma.

3.2. Exploração de Espaço de Projeto com HLS

Para reduzir o esforço de programação, a Síntese de Alto Nível (HLS) vem sendo constantemente melhorada nos anos recentes, enquanto disponibiliza diversas otimizações de hardware. O uso de diferentes otimizações e suas combinações resultam em diferentes versões de um mesmo emphKernel. Como discutido na Seção 2.3 do seção 2, essas implementações podem gerar variações em termo de performance/área/potência/energia.

Pham et al. propõem uma Exploração Dinâmica de Espaço de projeto que aproveita as dependências de *Loop-array* para achar melhor combinação de grão grosso em otimizações de HLS (*loop unrolling, pipelining, and array partitioning*)

[Khanh et al. 2015]. A *Framework* propõem uma análise de *Kenrels* construída a partir de um grafo de dependências que habilita a velocidade média de exploração em 14x.

Lin-Analyzer [Zhong et al. 2016] é uma ferramenta de análise que desempenha estimativas rápidas e precisas de performance em FPGA, assim como Exploração de Espaço de Projeto de acordo com diversas otimizações de grão grosso sem geração de implementação RTL. Ele identifica gargalos em diferentes implementações de FPGA quando aplicadas a estas otimizações, auxiliando o projetista em avaliar diferentes arquiteturas disponíveis pela Síntese de Alto Nível.

MP-Seeker [Zhong et al. 2017] é uma ferramenta de análise em alto nível, que avalia métricas de performance/área de diversas opções de aceleradores para uma aplicação em estados iniciais da exploração de espaço de projeto, antes da chamada de ferramentas HLS para a etapa final de síntese. Contrariamente ao Lin-Analyzer, MP-Seeker proporciona uma avaliação de parâmetros em grão fino como *tile size* e número de *PEs*, e parâmetros de grão grosso. Sua rápida Exploração de Espaço de Projeto acha uma combinação quase-ótima das opções de paralelismo.

COMBA [Zhao et al. 2019] é uma *Framework* baseada em modelo que avalia os efeitos de variedade de diretivas relacionadas a funções, laços e arranjos pelo uso de modelos analíticos agregáveis, um coletor de dados recursivo, é um algoritmo de Exploração de Espaço de Projeto orientado a métrica. Dadas diferentes restrições de recursos, COMBA encontra uma configuração de *Kernel* de performance quase-ótima.

3.3. Contribuições ao Estado da Arte

Como discutido acima, mesmo havendo trabalhos que avaliam Execuções colaborativas em CPU-FPGA [Huang et al. 2019, Wei et al. 2017] e ambientes multi-FPGA [Shan et al. 2019]), e outros tenham explorado Síntese de Alto Nível Sozinha para otimizar a geração de *Kernels* [Khanh et al. 2015, Zhong et al. 2016, Zhong et al. 2017, Zhao et al. 2019],***Multivers*** é o primeiro trabalho que tentará construir uma ponte entre a Síntese de Alto Nível e Execução Colaborativa CPU-FPGA em ambientes de *Cloud*, tirando vantagem da geração automática e rápida de versões para vários *Kernels* via HLS. Isso aumenta significativamente as possibilidades no Espaço de Exploração de Projeto, e proporciona ao provedor de Cloud a oportunidade de priorizar: paralelismo (requisitos para baixa área), energia ou performance de tempo de execução, de acordo com os requisitos do provedor e carga de trabalho em dado momento.

4. Experimentos Iniciais

Devido a variabilidade de circuitos lógicos equivalentes para implementar o mesmo algoritmo em hardware e aos métodos de desenvolvimento que estão disponíveis para o fluxo de projeto rápido dos mesmos, foram realizados experimentos de um estudo de caso para avaliar se haveria indícios espaço de exploração ao se usar diferentes diretivas de desenvolvimento via síntese de alto nível com objetivo de gerar amplitude nos resultados topológicos resultando em diferenças nos parâmetros de energia, performance e utilização de recursos entre diferentes versões.

Foi avaliado o algoritmo do domínio de criptografia MD5 que será explicado na subseção 4.1. Para realizar a exploração inicial do espaço de projeto foram realizados

experimentos utilizando a ferramenta de Sínteses de Alto Nível **Vivado HLS** e a ferramenta de síntese **Vivado**, ambos da Xilinx. Para dados de performance e de utilização de recursos foram considerados os resultados do relatório de síntese do Vivado HLS e para os dados de energia foram considerados os resultados do relatório de síntese da ferramenta Vivado. Como os experimentos, explicados em maior detalhes na subseção 4.2, tem como objetivo validar os espaços de projeto em Cloud, o dispositivo utilizado foi a placa de aceleração **Xilinx Alveo A200**, estes mesmos experimentos mostram também indícios que o mesmo espaço de exploração existe para outros nichos em que há um *trade-off* entre performance, recursos e energia, como sistemas embarcados.

4.1. MD5

MD5[Rivest 1992] é o quinto de uma série de algoritmos estilo *message digest* desenvolvido por Ronald Rivest. Foi projetado como função de *Hash* criptográfico, porém devido a vulnerabilidades hoje é utilizado como função de *Checksum* para verificação de integridade de dados.

4.2. Metodologia

Para avaliar o espaço de exploração, foram sintetizados códigos em linguagem C do algoritmo de MD5, nas condições descritas no começo deste capítulo. Os experimentos foram executados de forma a utilizar diretivas de compilação que indicam ao compilador de HLS como realizar a síntese da linguagem algorítmica para o circuito lógico equivalente. Para este trabalho foram utilizadas as diretivas de ***loop unrolling*** e ***loop pipelining***, que foram descritas em trabalhos anteriores como sendo as que mais impactam o resultado do hardware gerado [Zhong et al. 2016]. Para isso foram explorados os laços dentro dos *Kernels* presentes nos *Benchmarks* avaliados, em que versões foram geradas através das combinações de ***loop unrolling***, ***loop pipelining*** e nenhuma diretiva, descartando as combinações que seriam ignoradas pelo compilador (diretivas de *loop unrolling* aninhadas dentro de um laço mais externo com *loop pipelining* não são implementadas pelo compilador).

A exploração foi realizada a utilizando de uma ferramenta desenvolvida como parte deste trabalho que a partir de diretivas de pré-compilação do código e scripts de manipulação, conseguia utilizar de forma automática as ferramentas **Vivado HLS** e **Vivado**. Desta forma gerando a partir de um único código e de um arquivo com diretivas de pré compilação diversas versões de Hardware Sintetizado e seus respectivos relatórios, assim como relatórios de comparação entre as versões. Desta forma foram geradas 20 versões para o algoritmo de MD5 decorrente das diretivas a serem exploradas nos laços do algoritmo.

Após, foram classificadas as versões conforme suas características, onde para cada versão foi atribuído um peso calculado através de uma combinação linear entre os dados gerados pela ferramenta para cada uma das versões em Energia, Performance e Utilização de Recurso (Área) e pesos cada uma destas (equação 4.2), com os pesos representando à maior importância no espaço de exploração para cada uma das dimensões. Sendo (γ) referente a Energia, (β) referente a performance e α referente a área. As diferentes versões que atingiram os melhores resultados para cada peso são mostradas na figura 4 com granularidade de 0,25 para cada um dos recursos e com os números dentro das células sendo identificadores arbitrários de versão.

α: 0.0					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	-	17	17	17	17
0.25	7	7	17	17	17
0.5	7	7	7	17	17
0.75	7	7	7	7	17
1.0	7	7	7	7	7

α:0.5					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	0	3	3	3
0.25	3	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	6
1.0	6	6	6	6	6

α:0.25					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	3	3	3	3
0.25	3	3	3	3	6
0.5	6	6	6	6	6
0.75	6	6	6	6	6
1.0	6	6	6	6	6

α:0.75					
γ/β	0.0	0.25	0.5	0.75	1.0
0.0	0	0	0	3	3
0.25	0	3	3	3	3
0.5	3	3	3	3	3
0.75	3	3	3	3	3
1.0	3	3	3	3	3

Figure 4. Distribuições de Versões para o Algoritmo de MD5

$$\text{Kernel Benefit Value} \rightarrow \alpha\ Area + \beta\ Exec.Time + \gamma\ Energy \quad (2)$$

onde, Área,Tempo de Execução e Energia são normalizados considerando os máximo e mínimos valores de cada. A Área é dado pelo pior caso da porcentagem de utilização de recursos da FPGA (BRAM, LUT, DSP e FF)

4.3. Análise

Por exemplo, ao analisarmos na Figura 4,vemos que para o Benchmark MD5, onde cada uma das tabelas é configurada para um valor α de Área, e em cada uma das colunas existe um valor β de performance e em cada uma das linhas um valor γ de energia. Dentro das células onde há versão ótima conforme a equação 4.2. Para o MD5 percebemos que para os pesos $\alpha = 0$; $\beta = 0.5$; e $\gamma = 0.25$ a versão número 17 seria a melhor para este cenário. Assim como percebemos algumas versões que dominam os casos de contorno, (como por exemplo foco total em performance $\alpha = 0$; $\beta = 1$; $\gamma = 0$), e outras versões como a versão 3 dominam as condições mais equilibradas (por exemplo a versão 3 com $\alpha = 1$).

5. Planejamneto de Projeto de Trabalho

O trabalho desenvolvido neste projeto focará primeiramente em expandir os experimentos realizados na seção 4 de forma a generalizar as análises para conjuntos maiores de Benchmarks. Com esses experimentos, busca-se também revelar o potencial do HLS para exploração de espaço de projeto em grupos de algoritmos que sejam interessantes de serem executados em sistemas de Cloud.

Destes primeiros experimentos, espera-se que se desenvolva uma Framework de nome Multivers a partir da qual, através das técnicas descritas para geração de diferentes versões de *Kernels*, e utilizando os métodos descritos na subseção 2.3 e brevemente

explorados no seção 4, tirar-se proveito de forma a obter ganho de performance em ambientes de Cloud reais. Também será buscada a associação de outras técnicas relevantes em computação para outras etapas do sistema de forma a potencializar os ganhos com Multivers.

Será realizado também uma análise crítica do impacto real desta Framework em sistemas reais, utilizando métodos de avaliação que buscarão avaliar o aumento de eficiência individual dos métodos de HLS e de outros métodos associados, assim como a melhora global obtida pela associação dos métodos. As tarefas e suas distribuições em semanas são discutidas em maior detalhe nos próximos paragráfos.

Essas tarefas serão divididas em 10 semas para execução do projeto descritas na tabela 1, onde as duas primeiras semanas (S1 e S2) serão dedicadas a revisão dos conceitos discutidos neste planejamento e escolha de Benchmarks de características interessantes conforme o descrito. Após as ferramentas desenvolvidas serão utilizadas para levantar dados a cerca destes Benchmarks para prepara-los para serem utilizados na *Framework*(S2) e logo em seguida a partir das ferramentas desenvolvidas, haverá mais uma etapa de desenvolvimento para o encapsulamento destes para que possam ser utilizadas como parte integrantes da Framework (S3 e S4).

Na semana S5 serão exploradas outras técnicas presentes no estado da arte, como por exemplo, algoritmos de alocação de forma que possam ser utilizados para maximizar os efeitos proporcionados pelas etapas de exploração de espaço de projeto, e na próxima etapa (S6), estas técnicas serão integradas há *Framework* e os testes de integração serão feitos para valida-la semana 7.

Com a *Framwork* funcional e validada serão realizados experimentos para prova de conceito em ambientes que simulam a execução em um ambiente de Cloud real (S8 e S9). Destes experimentos serão levantados dados individuais de cada etapa, assim como da *Framework* funcionando como um todo de forma a se analisar a relevância das técnicas utilizadas e do escopo proposto durante o trabalho (S 10).

Atividade/Semana	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8	S 9	S 10
Revisão das Estratégias de Exploração de Espaço de Projeto E Escolha de Novos Benchmarks	■	■								
Aplicação Dos Métodos de Exploração de Espaço de Projeto nos Novos Benchmarks		■								
Desenvolvimento de uma Interface para Acesso às Diferentes Versões de Kernel Geradas			■	■						
Estudo de Associação de Técnicas Para Outras Etapas da Framework					■					
Integração das Técnicas Associadas						■				
Teste de Integração Da Framework							■			
Realização dos Experimentos na Framework Funcional								■	■	
Análise Crítica dos Resultados										■

Table 1. Planejamento de Execução do Projeto

References

Xilinx.com. https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/jyk1504034310030.html. Acessado: 28-02-2021.

Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533.

Huang, S., Chang, L.-W., El Hajj, I., Garcia de Gonzalo, S., Gómez-Luna, J., Chalamalasetti, S. R., El-Hadedy, M., Milojicic, D., Mutlu, O., Chen, D., et al. (2019). Analysis and modeling of collaborative execution strategies for heterogeneous cpu-fpga architectures. In *ICPE*, pages 79–90. ACM.

Jan, M. R., Anantha, C., Borivoje, N., et al. (2003). Digital integrated circuits: a design perspective.

Khanh, P. N., Singh, A. K., Kumar, A., and Aung, K. M. M. (2015). Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis. In *DATE*, pages 157–162.

Rivest, R. (1992). Rfc1321: The md5 message-digest algorithm.

Shan, J., Casu, M. R., Cortadella, J., Lavagno, L., and Lazarescu, M. T. (2019). Exact and heuristic allocation of multi-kernel applications to multi-fpga platforms. In *DAC*, pages 1–6.

Wei, X., Liang, Y., Wang, T., Lu, S., and Cong, J. (2017). Throughput optimization for streaming applications on cpu-fpga heterogeneous systems. In *ASP-DAC*, pages 488–493. IEEE.

Zhao, J., Feng, L., Sinha, S., Zhang, W., Liang, Y., and He, B. (2019). Performance modeling and directives optimization for high level synthesis on fpga. *IEEE TCAD*.

Zhong, G., Prakash, A., Liang, Y., Mitra, T., and Niar, S. (2016). Lin-analyzer: a high-level performance analysis tool for fpga-based accelerators. In *(DAC)*, pages 1–6. IEEE.

Zhong, G., Prakash, A., Wang, S., Liang, Y., Mitra, T., and Niar, S. (2017). Design space exploration of fpga-based accelerators with multi-level parallelism. In *(DATE)*, pages 1141–1146. IEEE.