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

ARTHUR KASSICK FERREIRA

Otimização ciente de topologia do particionamento de FPGAs para alocação de funções virtualizadas de rede

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

Orientador: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
2024

“Please look forward to it”

— NAOKI YOSHIDA

AGRADECIMENTOS

Agradeço a minha família pelo apoio e encorajamento para entrar no curso de Engenharia de Computação e conseguir chegar a essa reta final do curso depois de todos esses anos de dedicação. Agradeço também os professores instituto de informática e da escola de engenharia que me deram a base necessária para a realização desse trabalho de graduação. Agradecimentos especiais ao meu orientador Gabriel Nazar pela excelente orientação nesse trabalho, e ao colega Victor Guerra pelo auxílio e fornecimento das aplicações que foram essenciais para o desenvolvimento desse trabalho.

RESUMO

Virtualização de funções de rede (NFV) é uma prática que vem recebendo um grau crescente de adoção pela indústria de telecomunicações pelo potencial de eliminar a necessidade de hardware dedicado. Implementações iniciais em software sofreram com grande perda de desempenho, fomentando pesquisas em plataformas de virtualização de maior desempenho. Como alternativa está a implementação de funções de redes virtualizadas (VNFs) em FPGAs, que combinam a flexibilidade das Central Processing Units(CPUs) com a performance exigida para redes de alta velocidade. Contudo, o avanço da utilização de Field Programmable Gate Arrays (FPGAs) na área de NFVs esbarra na necessidade de que a malha do FPGA tenha um alto grau de ocupação de recursos para que se tire todo o potencial da tecnologia. Ferramentas atuais de particionamento de FPGAs não levam em consideração a presença do dispositivo FPGA numa rede conectada a outros dispositivos FPGAs, o que pode levar a desperdícios e utilização subótima de seus recursos. Este trabalho tem como principal objetivo o desenvolvimento de uma heurística de particionamento ciente de topologia, a qual busca maximizar a alocação de requisições de rede nos FPGAs pelo emprego de um algoritmo genético.
exemplo (NIEMIEC et al., 2020)

Palavras-chave: FPGA. NFV. VNF. Particionamento. Topologia de Rede. Topologia. Algoritmo genético.

Topology-aware FPGA partitioning optimization for the allocation of virtualized network functions

ABSTRACT

Network Function Virtualization (NFV) is a practice with growing adoption by the telecommunication industry due to its potential to eliminate dedicated hardware. Initial implementations in software suffered a great loss of perfomance, which sparked research into virtualization platforms with improved performance. Among the alternatives is the implementation of Virtualized Network Functions (VNFs) in FPGA, which combines the flexiblity of CPUs with the desired performance of the high speed networks. However, the expansion of FPGA adoption in the field of NFVs is barred by the necessity of high utilization of the FPGA fabric to extract the full potential of the technology. The current FPGA partitioning toolset does not take into account the presence of the FPGA device in a network with other FPGAs, which may result in waste of resources and suboptimal utilization of the device. This work's major objective has been the development of a topology-aware partitioning heuristic which seeks to maximize the allocation of network requests on the FPGAs thorugh the utilization of a genetic algorithm.

Keywords: FPGA,NFV,VNF,Partitioning,Network Topology,Topology,Genetic Algorithm,Field Programmable Gate Array.

LISTA DE ABREVIATURAS E SIGLAS

FPGA	Field Programmable Gate Array
NFV	Network Function Virtualization
VNF	Virtual Network Function
ASIC	Application-Specific Integrated Circuit
DPR	Dynamic Partial Reconfiguration
DPI	Deep Packet Inspection
TTM	Time to Market
KVM	Kernel-based Virtual Machine
QEMU	Quick Emulator
CPU	Central Processor Unit
PRR	Partial Reconfigurable Region
BRAM	Block Random Access Memory
DSP	Digital Signal Processing
CLB	Configurable Logic Block
SRAM	Static Random Access Memory
HDL	Hardware Description Language
MUX	Multiplexador
BLE	Basic Logic Element
CMOS	Complementary metal–oxide–semiconductor
MILP	Mixed Integer Linear Programming
SPR	Static Partial Reconfiguration
PR	Partial Reconfiguration
VM	Virtual Machine
LUT	Look up Table

SFC Service Function Chaining

GA Genetic Algorithm

ISP Internet Service Provider

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Field-Programable Gate Arrays (FPGAs) consistem em um tipo de hardware programável baseado em silício de alta flexibilidade de uso geral, o que permite o seu uso na implementação de qualquer circuito de hardware digital, conforme (BOUTROS; BETZ, 2021) e (FAROOQ; MARRAKCHI; MEHREZ, 2012). São largamente usados no desenvolvimento e prototipagem de dispositivos eletrônicos pelo seus menores custos de engenharia comparados aos Application-Specific Integrated Circuits (ASICs), assim como um menor do time-to-market (TTM), e pelo maior desempenho, em muitas aplicações, que soluções em software rodando em CPUs. A sua natureza reprogramável os torna uma opção atraente para a substituição de hardware dedicado para aplicações específicas, dentre elas aprendizado de máquina, redes neurais, processamento de vídeo e streaming, funções de rede e muitas outras. Tais características e casos de uso solidificaram a tecnologia FPGA como uma abordagem visada para problemas que demandam flexibilidade e desempenho em conjunto.

Inúmeros autores, dentre eles (WANG et al., 2021), (RUAN et al., 2022), (RINGLEIN et al., 2019) e (JORDAN et al., 2021), citam a crescente demanda computacional de servidores na nuvem e de datacenters. Tais ambientes, tanto pela sua natureza de constante mudança como a rápida evolução de suas aplicações, exigem grande desempenho, flexibilidade e adaptabilidade de seu hardware. Isso ocasiona uma intensa busca pela implementação de dispositivos FPGA nessas plataformas e empresas como Amazon, Microsoft, Meta e Alibaba têm investido grande quantidade de recursos financeiros, tecnológicos e humanos para desenvolver arquiteturas e frameworks que utilizem o hardware FPGA da forma mais eficiente e otimizada possível. Estudos como o de (DHAR et al., 2022) demonstram a atenção que as comunidades de pesquisa em FPGA e em computação na nuvem têm na elaboração de soluções para aumentar a adoção dos FPGAs como aceleradores de alta performance em ambientes compartilhados, citando como principal vetor de otimização o uso eficiente da Reconfiguração Parcial Dinâmica (DPR) em ambientes multiusuário, multiaplicação compartilhados como é o caso da computação na nuvem.

Nesse contexto, a virtualização de funções de rede (NFV) se torna essencial para suportar a expansão da infraestrutura de rede em tais ambientes. NFV, de acordo com (MIJUMBI et al., 2016) é o processo de executar funções de rede como firewalls e deep-packet inspection (DPI), que antes exigiam um tipo de hardware especializado conhecido como middlebox para sua execução, em servidores comuns. Abordagens iniciais envol-

veram o uso de processadores de uso geral como CPUs, os quais emulam por meio de software as funções de rede desejadas. Isso, no entanto, acarretou grande perda de desempenho perante os middleboxes. Vista a demanda pela flexibilidade sem abrir mão do desempenho dos ASICs, o uso da tecnologia FPGA se tornou extremamente atraente para a indústria de telecomunicações. Entretanto, como denota (RUAN et al., 2022) e (DHAR et al., 2022), as soluções para FPGAs atualmente disponíveis no mercado sofrem de problemas de portabilidade envolvendo designs de aplicações entre dispositivos FPGAs, de gerência das partições reconfiguráveis dos dispositivos para diferentes usuários e cargas de trabalho, e de sinergia com outras ferramentas atualmente empregadas em datacenters como hypervisors KVM/QEMU. Visto que os recursos computacionais atuais encontrados em FPGAs modernos frequentemente excedem as exigências individuais de cada aplicação, o manejamento do particionamento da malha do FPGA e de suas partições reconfiguráveis é um dos maiores empecilhos para que a tecnologia atinja o ápice de seu potencial.

Reconfiguração parcial é uma funcionalidade relevante para o uso de FPGAs em ambientes compartilhados por múltiplas aplicações. Sem ela, como denota (BANERJEE; SANGTANI; SUR-KOLAY, 2011), não é possível reconfigurar somente uma aplicação sem que todos os outros designs rodando no dispositivo sejam afetados. Para implementação dessa técnica de configuração de FPGAs, é necessário o uso de *floorplanning*. É nele que se que faz o particionamento das regiões, define-se quantos e quais recursos serão alocados para cada design, onde que será feito o *placement* de cada circuito e como diferentes módulos irão comunicar-se entre si. Existe uma vasta literatura no estudo de frameworks e métodos para automatização de *floorplanning* para FPGAs parcialmente reconfiguráveis, entre eles (RABOZZI; LILLIS; SANTAMBROGIO, 2014), (GALEA; CARPOV; ZAOURAR, 2018) e (RABOZZI; MIELE; SANTAMBROGIO, 2015). As principais preocupações desses trabalhos estão em como lidar com a heterogeneidade dos dispositivos e em como maximizar a eficiência de espaço e recursos das alocações das partições.

Uma limitação importante envolvendo a aplicação dos conhecimentos agregados de *floorplanning* e gerência de regiões reconfiguráveis para o uso de FPGAs para virtualização de funções de rede em servidores está no fato de que nenhum desses estudos considera o cenário no qual se desconhece quais aplicações especificamente rodarão naquele dispositivo. Como fazer o *floorplanning* para designs que ainda não se sabem se serão implementados apresenta um grande desafio que ainda não foi superado. Trabalhos

como (JORDAN et al., 2021), (DHAR et al., 2022) e (RINGLEIN et al., 2019) lidam com o problema do *floorplanning* ou com partições de tamanho estático e uniforme, ou delegam para o administrador responsável em implementar o sistema de configurar e definir as partições. Outra limitação atual dos trabalhos presentes na literatura está no fato de nenhum lidar com o particionamento dos FPGAs tendo em mente a sua utilização em rede. Isto é, tratam o FPGA como o único dispositivo presente e, por consequência, não fazem nenhum tipo de ajuste que leve em consideração a topologia da rede e os demais FPGAs existentes de forma distribuída na infraestrutura. Isso pode levar à situação em que, embora fosse possível alocar um conjunto de VNFs nessa rede, o particionamento agnóstico à topologia impede isso.

Tendo isso em vista, esse trabalho tem como objetivo buscar e desenvolver um método de *floorplanning* ciente da topologia para implementação de VNFs (Funções de rede virtualizadas) a fim de maximizar o uso de recursos e área de dispositivos FPGAs. Para alcançar essa meta, implementou-se um algoritmo genético cujos indivíduos representam diferentes redes seguindo a mesma topologia, isto é, as mesmas interconexões e distribuição de FPGAs em cada nodo. O *fitness* nesse caso representa a porcentagem de requisições VNFs geradas aleatoriamente que essa rede conseguiu implementar. Com isso conseguiu-se particionar cada FPGA de uma rede qualquer de acordo com a topologia apresentada por essa rede.

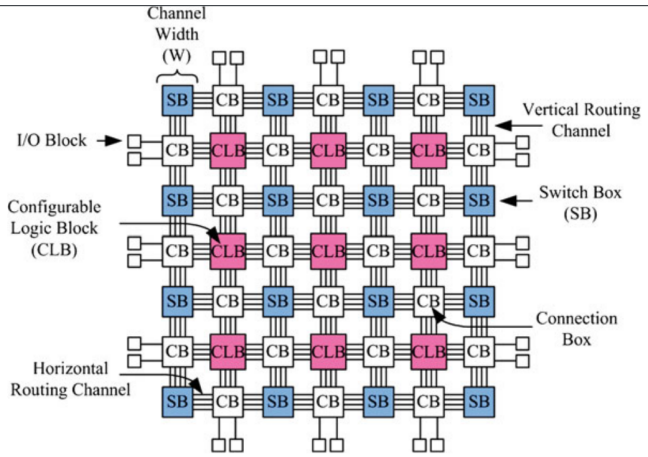
Este trabalho foi organizado de forma a apresentar desde a base motivacional e teórica do problema e tecnologias empregadas, até a tentativa de resolução e os resultados obtidos. O capítulo 2 desenvolve a fundamentação teórica e a revisão bibliográfica referente ao trabalho. Os capítulos 3 e 4 detalham como o problema foi modelado e qual a solução proposta com base nessa modelagem. O capítulo 5 apresenta a metodologia empregada, enquanto que os capítulos 6 e 7 trazem os resultados e a conclusão, respectivamente.

2 FUNDAMENTAÇÃO TEÓRICA E REVISÃO BIBLIOGRÁFICA

2.1 Field Programmable Gate Array (FPGA)

FPGA é definido por (FAROOQ; MARRAKCHI; MEHREZ, 2012) e (BOUTROS; BETZ, 2021) como dispositivos de silício reprogramáveis que podem assumir a função de quase qualquer circuito eletrônico digital ou sistema. É visto como uma alternativa a ASICs devido ao seu menor tempo de desenvolvimento e flexibilidade de poder ser reconfigurado para desempenhar diferentes funções sem custo adicional. O desenvolvimento de ASICs exige um intenso e custoso processo de engenharia que envolve o design físico, layout, fabricação e verificação. E qualquer atualização do design resulta em mais custos e um novo processo. Uma das desvantagens de FPGAs, todavia, se dá pelo alto custo de área das interconexões entre componentes necessária para prover essa maleabilidade: até 90% da área de um FPGA pode estar comprometida para tal propósito. Outro ponto fraco dos FPGAs comparados com ASICs está na perda de desempenho, a qual pode ser significativa e proibitiva em certos casos. Por tal razão, também é comum o uso de dispositivos FPGA na etapa de prototipagem de ASICs antes de uma empresa se comprometer com um design específico. Na Figura 2.1 encontra-se um exemplo de arquitetura FPGA.

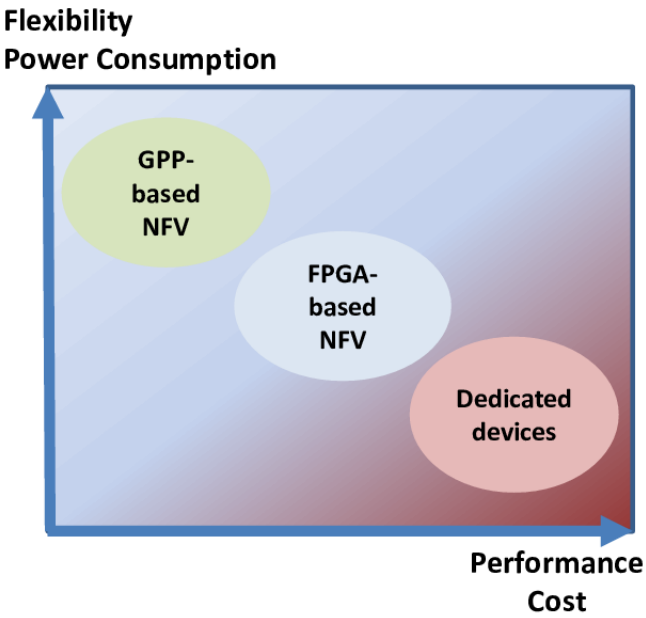
Figura 2.1: Exemplo de arquitetura FPGA.(FAROOQ; MARRAKCHI; MEHREZ, 2012)



Comparando a tecnologia FPGA com os processadores de uso geral como CPUs, contudo, a relação de flexibilidade e desempenho se inverte: FPGAs geralmente apresentam ganho de desempenho significativo a soluções puramente em software rodando em CPUs pois o FPGA pode ser configurado para ter exatamente o hardware necessário para a aplicação, porém FPGAs não têm a maleabilidade de implementar qualquer aplicação

sem configuração prévia como uma CPU rodando um código arbitrário. Dessa forma, FPGAs apresentam-se como uma solução intermediária comparada com ASICs e processadores de uso geral. A figura 2.2 ilustra como as diferentes soluções em NFV comparam com equipamentos dedicados.

Figura 2.2: Espaço de design para NFV. (KACHRIS; SIRAKOULIS; SOUDRIS, 2014)



Entre as tecnologias de programação para FPGAs, estão as baseadas em células SRAM (essas compondo a vasta maioria das plataformas FPGAs disponíveis no mercado), as baseadas em células flash e as baseadas na tecnologia anti-fusível. Embora a tecnologia SRAM não seja a mais eficiente em termos de área comparada com as demais, de acordo com (FAROOQ; MARRAKCHI; MEHREZ, 2012), ela se tornou a principal escolha para a construção da malha de dispositivos FPGAs devido à sua extrema flexilidade para reprogramação, assim como o uso de processos padrão de fabricação CMOS, o que facilita a integração com demais tecnologias, e o fato de células SRAM terem uma vida útil longa, o que elimina a preocupação com qualquer limite da quantidade de escritas que se pode fazer.

Para atingir as funcionalidades desejadas de hardware desejadas, o desenvolvedor de hardware tradicionalmente faz uso de uma linguagem de descrição de hardware (HDL), como VHDL ou Verilog, para delimitar os recursos utilizados pela sua aplicação assim como o comportamento alvo. É a partir desse código que programas de design para FPGAs, como o software Vivado da Xilinx, iniciam um processo de múltiplas etapas que culmina na geração de um arquivo de configuração chamado de bitstream. É nesse bitstream que estão todos os bits de configuração da memoria do FPGA e que determinam

os valores armazenados nas Lookup Tables (LUTs) e demais componentes.

Em (FAROOQ; MARRAKCHI; MEHREZ, 2012) um dispositivo FPGA é definido como sendo dividido em três partes principais:

1. Blocos lógicos programáveis que implementam funções lógicas (CLBs). CLBs são compostos por um ou mais Basic Logic Element (BLE) organizados em clusters. BLEs, por sua vez, são formados por outros subcomponentes lógicos como:

    - LUT: Array de células SRAM cujos valores representam a tabela verdade de uma dada função lógica a ser representada. Geralmente recebem a nomenclatura referente ao seu tamanho, dessa forma uma K-LUT consiste de um array 2^K posições que representa qualquer função de K entradas.
    - MUX: Multiplexadores tem como papel a seleção dos valores de saída da função implementada dado um certo conjunto de entrada.
    - Flip-Flop: atuam no armazenamento de bits do BLE e na sincronia do mesmo com o restante da malha do FPGA.

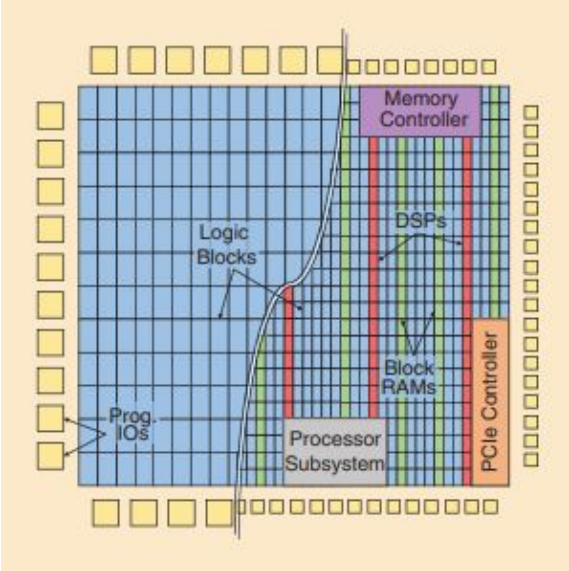
2. Roteamento programável que conecta tais funções lógicas

3. Blocos de entrada e saída (IO) que são conectados pelo mesmo roteamento interno aos blocos lógicos e fazem a conexão do chip com o mundo externo.

Em complementação a esses componentes, (BOUTROS; BETZ, 2021) denota que para FPGAs atuais, a malha dos dispositivos FPGAs ganhou um caráter heterogêneo e evoluiu para incluir uma vasta gama de blocos para execução de tarefas diversas como Block RAMs (BRAMs), as quais conferem aos dispositivos atuais a capacidade de armazenamento de dados em memória internamente no chip, sem necessidade de componentes externos, Digital Signal Processing (DSP), bloco responsável por operações aritméticas envolvendo multiplicação e adição, e outros tipos de blocos dedicados. São esses blocos especializados, em adição aos CLBs, que dão ao FPGA o seu poder computacional pareado com grande flexibilidade de desempenhar o papel de quase qualquer circuito digital, tornando a tecnologia tão visada em datacenters e grandes servidores. Abaixo a Figura 2.3 ilustra essa evolução da tecnologia FPGA.

Da mesma forma que aplicações de software suficientemente complexas necessitam ser divididas em subcomponentes a fim de que seja possível aos desenvolvedores lidarem com a enorme escala de trabalho imposta pelo código, aplicações em FPGA são divididas em módulos responsáveis por emular um circuito específico cada. Contudo,

Figura 2.3: Evolução da malha FPGA.(BOUTROS; BETZ, 2021)



diferentemente das CPUs, os FPGAs exigem um particionamento e configurações prévias no dispositivo para o compartilhamento dinâmico de recursos, enquanto que no caso das CPUs isso fica totalmente a cargo do sistema operacional e das aplicações que serão executadas.

2.1.1 Floorplanning

Em (MURRAY; BETZ, 2015), *floorplanning* é descrito como uma abordagem que melhora a escalabilidade de algoritmos CAD existentes para o desenvolvimento de designs para FPGAs, além de ser uma parte essencial da reconfiguração parcial. É uma técnica que permite que se implemente uma estratégia de dividir para conquistar para lidar com a implementação de grandes circuitos ao fragmentá-los em componentes menores e alocar cada componente do design a regiões específicas do dispositivo FPGA.

Os autores, assim como outros pesquisadores da área de *floorplanning* para FPGAs entre eles (BANERJEE; SANGTANI; SUR-KOLAY, 2011), (GALEA; CARPOV; ZAOURAR, 2018) e (RABOZZI; LILLIS; SANTAMBROGIO, 2014), definem o problema de *floorplanning* como a divisão da malha FPGA em blocos de recursos de vários tipos como CLBs, BRAMs, DSPs, etc., e o agrupamento desses blocos em regiões para as quais cada módulo do design a ser desenvolvido será alocado. Conforme (RABOZZI; MIELE; SANTAMBROGIO, 2015) tal alocação, ocasionalmente também chamada de *placement*, deve, no contexto de reconfiguração parcial, obedecer uma série de requisitos:

1. Cada *placement* atribuído deve conter a quantidade mínima de recursos exigidas pelo módulo

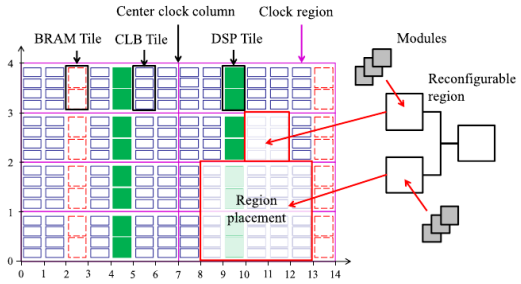
2. Nenhum *placement* deve violar as regras de atribuição de regiões reconfiguráveis definidas pelo dispositivo. Isso ocorre porque alguns dispositivos restringem onde uma região configurável pode ser posta, devido a presença de hardware dedicado (hard processors), lógica estática ou blocos de IO.

3. As fronteiras laterais da região não podem dividir recursos de interconexão do dispositivo. Devido a isso, as bordas esquerda e direita devem estar alinhadas com coordenadas específicas

4. Dependendo das regras de configuração do modelo, recursos do tipo CLB nas margens da coluna central de clock não podem ser usadas pela região reconfigurável. Esses recursos devem estar na parte estática do design. É possível que a região reconfigurável englobe essa área, porém esses CLBs não estarão de fato disponíveis para a região reconfigurável.

5. *Placements* de duas regiões diferentes não podem se sobrepor.

Figura 2.4: Representação do problema de *floorplanning*.(RABOZZI; MIELE; SANTAMBROGIO, 2015)



Há de se notar que a reconfiguração parcial envolve um caso específico de *floorplanning* para dispositivos FPGAs, especificamente o *floorplanning* de regiões com capacidade de serem reconfiguradas e terem o circuito nelas implementado alterado após a primeira configuração.

2.1.2 Reconfiguração Parcial (PR)

(HASSAN et al., 2015) define Reconfiguração Parcial como a reconfiguração de uma parte de um FPGA após a sua configuração inicial, podendo ser de dois tipos: Re-

configuração Parcial Dinâmica (DPR) ou Reconfiguração Parcial Estática (SPR). Há de se observar, contudo, que a vasta literatura no assunto trata DPR e PR como termos equivalentes e SPR como uma tecnicalidade. A principal diferença entre as duas técnicas está no fato de DPR permitir que a reescrita da configuração da região afetada ocorra em tempo de execução sem interromper o funcionamento das demais, o que é de extrema importância para sistemas de múltiplas aplicações simultâneas ou de alta modularidade. Não só porque permite atualizar designs já implementados em tempo real, mas também porque aumenta a eficiência e ocupação de área da malha FPGA de forma significativa: aplicações ociosas no FPGA podem ser substituídas por outras que aguardavam oportunidade de execução, eliminando o problema de apenas uma pequena parte de um dispositivo FPGA esteja sendo de fato usada.

Entretanto, a técnica possui um conjunto de limitações envolvidas como denota (RABOZZI; MIELE; SANTAMBROGIO, 2015). A primeira está na exigência do dispositivo FPGA em questão ter suporte para tal, o que limita o uso dos modelos mais antigos. A segunda limitação está no software de design e configuração do FPGA ter o suporte para reconfiguração parcial. A terceira está na necessidade de intervenção manual por parte do designer para o uso da técnica com os softwares de design disponíveis comercialmente. Existem soluções propostas para tais problemas, como a feita pelo próprio autor, e as propostas por (WANG et al., 2021) e (BANERJEE; SANGTANI; SUR-KOLAY, 2011). Contudo, tais trabalhos lidam apenas com o problema de como particionar as regiões reconfiguráveis de forma automática para um conjunto de designs já conhecidos. Caso se deseje implementar novos circuitos, cuja exigência de recursos pode ser completamente diferente dos designs já implementados, não há garantia que o particionamento será compatível e muito menos eficiente.

A quarta e última limitação, e uma das mais importantes no uso dessa técnica e a mais importante para esse trabalho, está no fato que mesmo com PR, é obrigatório o particionamento a priori de todo o dispositivo, tanto a delimitação da região estática, quanto a definição das Regiões Parcialmente Reconfiguráveis (PRRs). Alterar esse particionamento requer a reconfiguração completa do FPGA, o que afeta a principal vantagem da técnica comparada com a reconfiguração completa e limita a aplicação da DPR em ambientes dinâmicos, como servidores na nuvem. No cenário em que o dispositivo precise ser reparticionado com frequência devido a alterações frequentes no conjunto de módulos a ser implementado, não é incomum abandonar o uso de DPR devido aos benefícios não se mostrarem presentes.

Outras abordagens ao problema de ineficiência de utilização do FPGA assim como as limitações associadas a DPR, estão em frameworks como MUTECO, elaborado por (JORDAN et al., 2021), DML por (DHAR et al., 2022) e a plataforma Sled por (RINGLEIN et al., 2019). Todas têm em comum o objetivo de reduzir ao mínimo possível a relação de recursos ociosos dentro do FPGA, alocando na medida do possível diferentes regiões da malha FPGA para as aplicações que necessitam, e de abstrair ao máximo a parte de configuração e reprogramação do FPGA. Esses frameworks têm como vantagem a capacidade de lidar com o problema de não se saber exatamente quais circuitos precisarão ser implementados. Todavia, possuem a limitação de, que para atingir tal flexibilidade, precisaram recorrer a regiões de tamanho uniforme, não suportam DPR ou lidam com particionamento diretamente e delegam a tarefa ao usuário, o que compromete em parte a sua eficiência.

2.1.3 Cloud FPGA

Entre os setores com maior demanda pelo poder computacional com flexibilidade para cargas de trabalho variáveis está o de computação na nuvem. Em (RINGLEIN et al., 2019) são listados, como exemplo de aplicações na nuvem que fazem uso de FPGAs para suprimir deficiências impostas por sistemas tradicionais, dois projetos da Microsoft: o projeto Catapult de 2014 que visou utilizar FPGAs como aceleradores de hardware para o sistema de busca Bing e o projeto Brainwave, continuação do de 2014, o qual incorporou também inteligência artificial. Outra empresa que emprega a tecnologia FPGA é a Amazon por meio do seu serviço de hosteamento AWS, no qual além de máquinas virtuais (VMs) com CPUs convencionais, disponibiliza VMs que rodam sobre dispositivos FPGAs. Outras aplicações incluem frameworks de computação distribuída como Hadoop, Spark e Tensorflow.

Em (JORDAN et al., 2021) é demonstrado que atualmente em servidores da nuvem e em grandes datacenters estão cada vez mais populares os ambientes de hardware heterogêneos, nos quais tanto CPUs quanto FPGAs são utilizados e compartilhados para a execução de diversas aplicações por múltiplos usuários. Como os autores enfatizam, a gestão e configuração eficiente de partições reconfiguráveis na malha FPGA é de fundamental importância para o funcionamento desses sistemas. Tal visão é compartilhada por (DHAR et al., 2022), que chama a atenção para a necessidade de isolar o impacto das alterações a uma aplicação em execução no dispositivo das outras como principal fator

que torna uso de DPR indispensável. Dessa forma, o uso eficiente de DPR, e os desafios associados a tal feito, é um ponto chave para que se consiga atingir maiores níveis de eficiência no mapeamento da grande variedade de aplicações encontradas num ambiente de computação na nuvem, o qual se mostra no presente como um dos grandes obstáculos para a adoção de FPGAs como aceleradores de alta performance na nuvem.

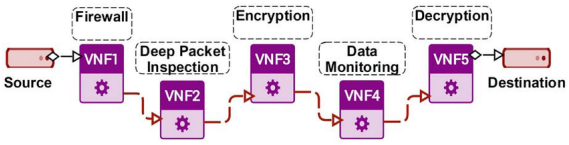
2.2 Network Function Virtualization (NFV)

De acordo com (MIJUMBI et al., 2016), a virtualização de funções de redes (NFV) insere-se no contexto dos desafios encontrados pela indústria de telecomunicações para eliminar a baixa agilidade a mudanças nas exigências e necessidades dos usuários apresentada pelos serviços de rede e para lidar com a forte dependência de hardware especializado. Tais problemas resultam não só no alto custo com mão de obra, pois as contínuas mudanças nos equipamentos utilizados exige uma alta capacitação em constante mudança para os técnicos que operam sobre esse equipamento, como também o alto custo envolvido na substituição de hardware e adaptação da infraestrutura. NFV entra como uma solução que visa eliminar o uso de hardware dedicado, substituindo-os por serviços virtualizados rodando em hardware padrão para servidores. Com o uso de tecnologias de virtualização de funções de rede, empresas podem facilmente alterar ou aumentar a capacidade e organização de suas redes sem precisar investir em novo hardware.

Implementar NFV no contexto de um certo serviço oferecido pela rede significa decompor esse serviço em um conjunto de funções virtualizadas de rede (VNFs) as quais podem por sua vez serem implementadas em software. Em (HERRERA; BOTERO, 2016) os autores estabelecem que, num ambiente NFV, um serviço de rede consiste de um conjunto de VNFs encadeadas, enquanto que sua construção e despacho ocorre pela definição do número de VNFs contidas pelo serviço e da ordem das VNFs dentro do encadeamento e pela alocação dessa cadeia de VNFs dentro da infraestrutura da rede. É o encadeamento de VNFs, também chamado de Service Function Chaining (SFC), que dá ao serviço de rede a flexibilidade e funcionalidades necessárias para desempenhar a função desejada, pois é essa técnica que permite que o mesmo serviço contenha mais que uma VNF, conforme exemplifica a Figura 2.5.

Entre as mudanças que o emprego de NFV traz à entrega de serviços de rede estão:

Figura 2.5: Service Function Chaining.(HERRERA; BOTERO, 2016)



1. o desacoplamento entre hardware e software, pois, sem a necessidade de desenvolver funções de rede para rodar em um hardware específico, é possível dividir a atenção no desenvolvimento e evolução do hardware e software utilizado.

2. Alocação flexível de funções de rede, pois não há mais a necessidade de adquirir ou transferir equipamento específico e lidar com a sua instalação.

3. Escalabilidade dinâmica por meio da rápida e flexível alocação e desalocação de funções de rede conforme a demanda.

Entretanto, entre os principais obstáculos enfrentados no uso de NFV está a perda de desempenho encontrada na transição do emprego de hardware dedicado, o qual por sua natureza recebe grande atenção para otimizar sua performance para as tarefas as quais se tem como objetivo, para hardware genérico que roda as funções de rede de forma virtualizada em software. Para contornar tal problema, uma das soluções propostas na literatura está no emprego de FPGAs como plataforma de virtualização para as funções de rede. Em (NIEMIEC et al., 2020), os autores denotam que FPGAs têm o potencial de prover a flexibilidade e performance exigida pelo NFV no contexto de redes de alta velocidade. E é por tal razão que a indústria de telecomunicações tem tido tanto interesse pela tecnologia em adição ao tempo de desenvolvimento reduzido comparado com ASICs e uma performance em geral maior que CPUs.

Um exemplo de arquitetura voltada ao emprego de FPGAs para NFV é dado pelo trabalho de (PAOLINO; PINNETERRE; RAHO, 2017), o qual propõe uma arquitetura chamada VirtManager para a gerência dos dispositivos FPGAs e a orquestração das funções de rede conforme a demanda. A Figura 2.6 ilustra essa arquitetura.

Outro exemplo de uso de FPGAs no contexto de NFV é encontrado no trabalho proposto por (ALMEIDA et al., 2023), exemplificado pela Figura 2.7. Nele se propõe uma arquitetura voltada para a implementação de NFV em redes móveis 5G e em redes integradas com satélites. Entre os pontos críticos apresentado pelos autores para o desenvolvimento da arquitetura, e escolha da tecnologia FPGA, estão o desempenho exigido pelas velocidades 5G e a flexibilidade e reconfigurabilidades envolvidas com o uso de redes de satélites.

Figura 2.6: Arquitetura VirtManager.(PAOLINO; PINNETERRE; RAHO, 2017)

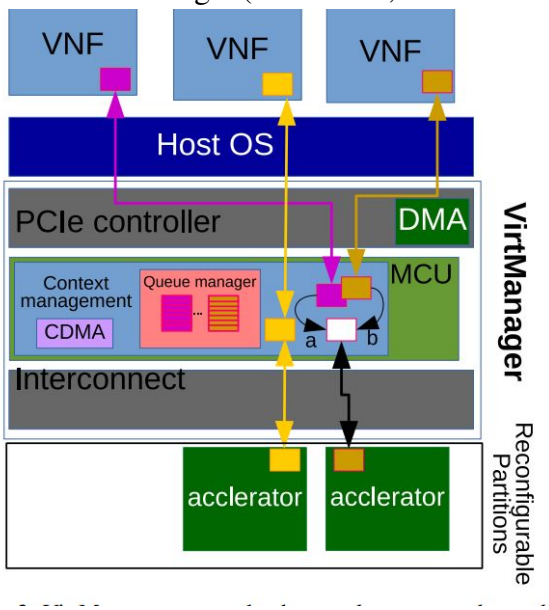
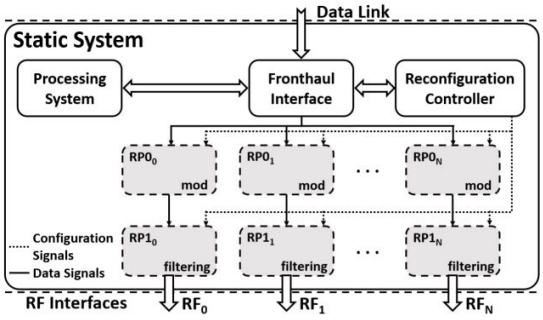


Figura 2.7: Arquitetura proposta por.(ALMEIDA et al., 2023)



2.3 Algoritmo genético

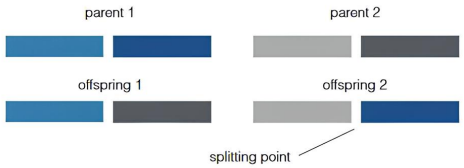
Conforme (SRINIVAS; PATNAIK, 1994), algoritmos genéticos são definidos como um conjunto de técnicas de busca e otimização com inspiração nos mecanismos de evolução, seleção natural e genética encontrados na natureza. Interesse nesse tipo de abordagem existe desde a década de 70, com a proposta de um algoritmo feito por (HOLLAND, 1975). Desde então, a área recebeu enorme atenção e pesquisa para desenvolvimento de algoritmos para aplicação em aprendizado de máquina, tecnologia VLSI, processamento de imagens, roteamento de redes de computadores, mineração de dados, além de quaisquer áreas nas quais há interesse em otimização de problemas.

A principal motivação por trás da emulação dos princípios de evolução natural está no fato de muitos problemas de otimização serem demasiadamente complexos, com inúmeros parâmetros e condições para serem analisados e uma grande variedade de soluções possíveis, a qual nem sempre pode ser verificada em sua totalidade. Outra dificuldade

está no fato de somente um aspecto específico do problema precisar ser otimizado, o que exige a busca de quais parâmetros de configuração são realmente relevantes à solução ótima. Há de se mencionar, também, que em alguns casos as condições do problema e as características do ambiente estão em constante mudança. Isso leva a solução ótima encontrada num primeiro momento a não ser mais tal num outro momento. Aspectos como esses levam à necessidade de um tipo de algoritmo que consegue por conta própria definir quais os parâmetros mais importantes e selecionar os melhores valores conforme um ambiente que muitas vezes está em constante mudança.

Na natureza, indivíduos de uma espécie qualquer disputam entre si e com demais espécies pelos recursos disponíveis para a sua sobrevivência e eventual procriação, a fim de perpetuar a sua espécie. Além dessa competição, há também as condições ambientes em que o indivíduo se encontra, as quais estão em constante mudança. Adaptar-se a esses desafios à sobrevivência é o ponto chave para a contínua existência da espécie. E o que define o quão capaz cada indivíduo é para sobreviver é a manifestação de suas características genéticas. Cada característica é geralmente controlada por um ou mais genes. Tais genes dependem do parentesco desse indivíduo, isto é, quem são os seus pais, e mudanças ocorridas durante a sua vida, as quais chamamos de mutações. O parentesco é um aspecto extremamente importante para a adaptabilidade não só do indivíduo, mas de sua espécie inteira. Isso se deve ao fato que o processo de reprodução está diretamente relacionado à diversidade de material genético da espécie. Durante a reprodução, num processo chamado de crossover, os cromossomos de cada um dos pais recombinam-se para criar os genes do filho, os quais podem ou não ter as mesmas combinações de cromossomos. E é essa variedade de combinações que define a diversidade genética e consequentemente a possibilidade da espécie se adaptar a uma maior gama de mudanças ambientais.

Figura 2.8: Exemplo de crossover em que o filho herda 1 cromossomo de cada pai. (KRAMER, 2017)



Nessa competição pela sobrevivência, os indivíduos mais aptos possuem maiores chances de sobrevivência e reprodução, enquanto que os que menos aptos possuem maior risco de serem eliminados. Dessa forma, contínua e sistematicamente os indivíduos de uma espécie são selecionados conforme um conjunto de características que melhor se

adapta ao ambiente em que se encontram, ou, caso isso não seja possível, são levados a extinção. No caso dos algoritmos genéticos, essa aptidão se dá por um valor numérico calculado pela função de *fitness*. Essa função é específica ao problema em que se está trabalhando e quais características são consideradas importantes, sendo um dos aspectos mais importantes na construção de um algoritmo genético.

São essas características de constantemente selecionar os indivíduos mais aptos, em um ambiente no qual a definição de mais apto também está sempre em mudança, que inspirou a criação e uso extensivo de algoritmos genéticos. Isso se dá ao fato que o processo de seleção natural de indivíduos numa espécie pode ser comparado a um processo de otimização das características dessa população

Em (LAMBORA; GUPTA; CHOPRA, 2019) define-se como algoritmo genético a seguinte sequência de passos:

1. Inicialização da população
2. Avaliação dos indivíduos (cálculo de *fitness*)
3. Seleção de indivíduos para reprodução conforme aptidão
4. Crossover de cromossomos
5. Mutações dos filhos
6. Checagem do critério de parada. Caso falso, volta para o passo 2
7. Fim do algoritmo

No caso desse trabalho, o algoritmo genético implementado é aplicado na otimização do particionamento dos FPGAs presentes na rede tendo em vista que a carga de trabalho futura, isto é, os designs que serão implementados nesses FPGAs, é arbitrária e não pode ser antecipada de acordo com quaisquer padrões. Os detalhes dessa implementação podem ser encontrados no capítulo 4.

2.4 Trabalhos Relacionados

Propostas de metodologias e frameworks para floorplanning de designs modularizados para FPGAs envolvendo PR podem ser encontradas na literatura atual com grande variedade de abordagens.

Autores como (WANG et al., 2021), (RABOZZI; LILLIS; SANTAMBROGIO, 2014), (GALEA; CARPOV; ZAOURAR, 2018) e (MURRAY; BETZ, 2015) exploraram variadas técnicas para automatizar o processo de *floorplanning* de regiões reconfiguráveis e como produzir os resultados mais eficientes dada uma série de parâmetros fornecidos pelo designer. No trabalho desenvolvido em (WANG et al., 2021), os autores utilizam uma abordagem de modelagem por descrição dinâmica de regiões reconfiguráveis e Mixed-Integer Linear Programming (MILP) com o objetivo principal de reduzir a fragmentação interna de partições e aumentar a utilização da malha do FPGA. Em comparação, em (RABOZZI; LILLIS; SANTAMBROGIO, 2014) os autores fazem uso de MILP para o particionamento e geração de floorplans envolvendo PR para FPGAs. Em (MURRAY; BETZ, 2015) elaborou-se um floorplanner baseado em simulated annealing e árvores binárias que também incorpora Irreducible Realization Lists (IRLs) durante o processo de geração do floorplan. A abordagem de simulated annealing também foi empregada em (GALEA; CARPOV; ZAOURAR, 2018) para elaborar possíveis floorplans e escolher os melhores candidatos.

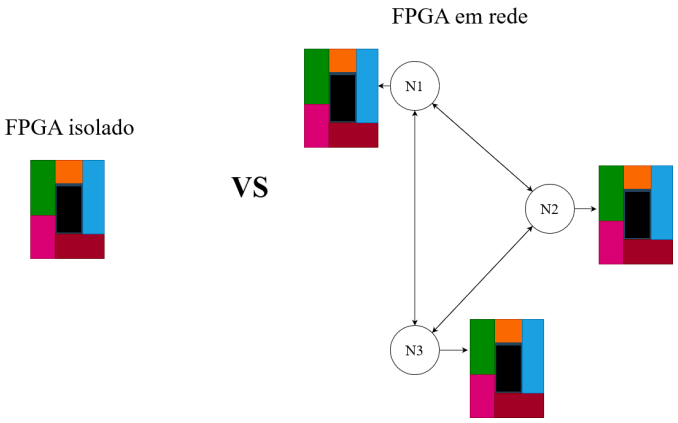
Contudo, tais metodologias de *floorplanning* têm como foco a otimização da alocação de designs completos para regiões reconfiguráveis. Ou seja, já se conhece a priori qual o circuito a ser implementado junto de suas características, demanda de recursos, restrições e outros detalhes importantes para a implementação. Para casos em que não se sabe ainda quais designs se deseja implementar, o que é comum em ambientes compartilhados por múltiplas aplicações e usuários e em sistemas de escopo dinâmico, as otimizações trazidas por esses métodos não necessariamente se aplicam ou se aplicam apenas parcialmente, o que exige uma mudança fundamental na abordagem do problema para conseguir os resultados desejados. Além disso, tais abordagens não levam em consideração a presença desses FPGAS em uma rede na qual estão conectados a outros dispositivos e portanto, não fazem uso dessa característica no momento do particionamento.

Trabalhos como os elaborados por (DHAR et al., 2022), (JORDAN et al., 2021) e (RINGLEIN et al., 2019) são exemplos de soluções voltadas a esse cenário de precisar alocar designs de caráter diverso em uma malha FPGA compartilhada sem o conhecimento a priori de suas características e funcionalidades. (JORDAN et al., 2021) propõe uma framework voltada a ambientes CPU-FPGA compartilhados por múltiplos usuários e aplicações. O framework lida tanto com a alocação de processos às CPUs quanto das aplicações FPGAs às regiões reconfiguráveis dos dispositivos. No que tange o particionamento dos FPGAs disponibilizados no servidor, o trabalho assume um cenário em que o

administrador do sistema previamente fez o particionamento e não lida com a configuração inicial das regiões reconfiguráveis. (DHAR et al., 2022) e (RINGLEIN et al., 2019) também propõem frameworks para gerência de dispositivos FPGAs em ambientes compartilhados na nuvem, contudo a principal diferença dos dois para o framework MUTECO está em assumir a tarefa de configurar as regiões reconfiguráveis dos FPGAs. (DHAR et al., 2022) propõe um particionamento uniforme e regular da malha FPGA em slots de tamanho iguais entre si. Para designs que excedem a quantidade de recursos de um slot, aloca-se um múltiplo inteiro de slots até que a demanda seja satisfeita. Já (RINGLEIN et al., 2019) assume um cenário em que cada FPGA é utilizado por apenas um usuário, sendo feita a alocação de aplicações de diferentes usuários para diferentes dispositivos FPGAs.

Apesar desses trabalhos conseguirem lidar com a alocação dinâmica de designs aos quais não se pôde preparar previamente, nenhum deles considerou uma metodologia de particionamento das regiões reconfiguráveis que buscasse otimizar a alocação dessas regiões para diferentes perfis de aplicações, como por exemplo funções virtualizadas de rede. Muito menos que os dispositivos FPGAs a serem particionados estariam inseridos numa rede e que o conjunto de aplicações a serem implementadas seriam distribuídas nessa rede. Há de se considerar a possibilidade que, assumindo que as aplicações a serem implementadas no FPGA tenham certas características em comum, os particionamentos que melhor utilizem a malha do dispositivo sejam dependentes de um conhecimento global do perfil dessas aplicações. Também não se pode ignorar que, no cenário em que os dispositivos FPGAs estão conectados por uma rede a outros dispositivos, particionar cada FPGA individualmente de forma agnóstica a topologia da rede pode levar a situações em que não se consegue implementar todas as aplicações desejadas embora o conjunto de FPGAs presentes na rede tenham capacidade para tal. A figura 2.9 ilustra a diferença da abordagem atual encontrada na literatura comparada com uma ciente de topologia.

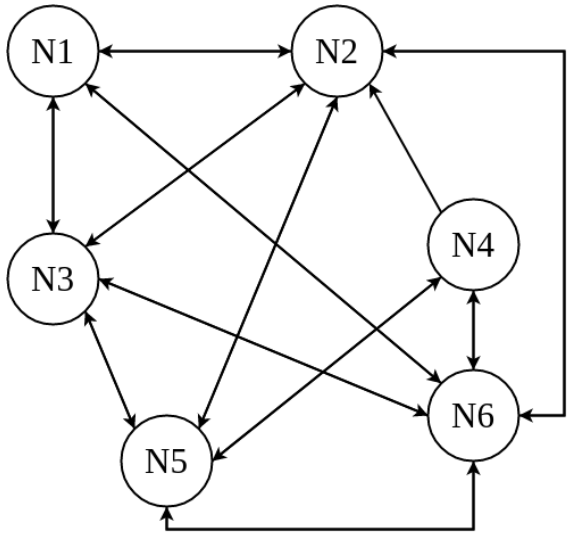
Figura 2.9: Abordagem atual agnóstica a rede comparada com uma abordagem ciente de topologia



3 DEFINIÇÃO E MODELAGEM DO PROBLEMA

Considera-se uma rede de tamanho e topologia arbitrários, representada por um grafo como o da Figura 3.1. Cada nodo dessa rede representa um conjunto de dispositivos de rede (roteadores, switches) com uma certa capacidade computacional para implementação de VNFs. Essa capacidade computacional é dada por um conjunto de FPGAs, podendo ir de zero (nenhum FPGA, e portanto, nenhuma capacidade de executar VNFs) até três dispositivos FPGAs, como representado na Figura 3.2. Cada dispositivo FPGA possui a capacidade de ser parcialmente reconfigurado e está particionado em um conjunto arbitrário de PRRs. Tais regiões se restringem a formatos retangulares, pois conforme as recomendações dadas por (XILINX, 2023), regiões de formato irregular (formato em T ou L, por exemplo) apresentam um acréscimo significativo na complexidade de roteamento e posicionamento de recursos durante a síntese, e portanto não são recomendadas. As arestas representam os enlaces de rede entre cada nodo, possuindo uma capacidade de vazão e um valor de latência. Considera-se que todos os enlaces são bidirecionais.

Figura 3.1: Representação em grafo da rede

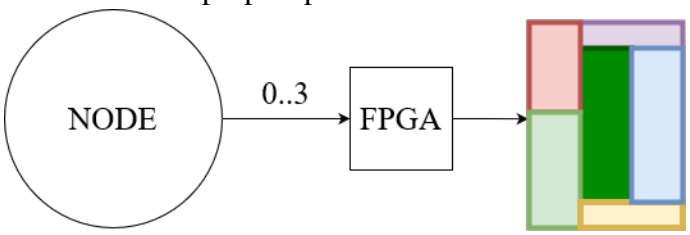


Supondo que para essa rede, cujas características de topologia, composição dos seus nodos e capacidade computacional são conhecidas, deseja-se poder implementar um número arbitrário de VNFs dentro de um conjunto possível de VNFs (por exemplo, firewalls e inspetores de pacotes). Para cada VNF a ser implementada, considera-se um ou mais possíveis designs (que variam em capacidade de processamento e recursos da malha FPGA exigidos pela implementação). Além disso, é necessário que haja a capacidade de excluir ou incluir VNFs dentre as funções implementadas conforme as demandas dos usuários dessa rede. Tais VNFs estariam ainda dentro do conjunto previamente mencio-

nado, e portanto, sabe-se de antemão as suas características. Contudo, não há a possibilidade de prever o conjunto de VNFs dentre as funções a serem implementadas. Ou seja, não há como saber quantas das VNFs a serem futuramente implementadas irão pertencer a cada tipo possível de requisição. Define-se como requisição um pedido de implementação de um conjunto de VNFs (uma SFC), que inclui demandas de vazão mínima, latência máxima, nodo origem e nodo destino.

Com esse cenário estabelecido, define-se como problema alvo desse trabalho a seguinte pergunta: como particionar os FPGAs presentes de uma dada rede, de forma a maximizar a quantidade de requisições atendidas, sabendo que elas pertencem a um conjunto conhecido de VNFs porém sem saber o conjunto de requisições que as incluem? O objetivo do trabalho, dessa forma, é o desenvolvimento de um programa gerador e otimizador de particionamento de FPGAs ciente de topologia.

Figura 3.2: Representação de um nodo. Cada nodo possui de 0 a 3 FPGAs associados, cada um com o seu próprio particionamento



3.1 Definição do ambiente de execução

Tendo em mente a modelagem estabelecida no capítulo 3, define-se como necessário para o programa que buscará os melhores particionamentos para uma dada rede:

1. Arquivo descritor da rede e sua topologia, o qual contém a composição dos nodos em termos de FPGAs, quais os links existentes e suas características no que tange latência e vazão.

2. Arquivo descritor dos modelos de FPGA presentes nessa rede. Esse arquivo deve fornecer as informações sobre a malha FPGA e suas capacidades, isto é, sua quantidade de recursos como CLBs, DSPs, BRAMs e demais tipos e sua disposição espacial no dispositivo. Além disso, é fundamental que esse arquivo delimite a região estática que será alocada no dispositivo a fim de que seja possível fazer o particionamento das PRRs.

3. Arquivo contendo a lista de VNFs e requisições a serem usadas durante a execução do programa como as funções de rede para as quais será otimizado o particionamento.

Define-se como saída final desse programa um arquivo descritor da rede contendo as informações de particionamento de cada FPGA. Esse arquivo representa o melhor particionamento encontrado durante o período de execução do programa para a rede e demais arquivos de entrada. O critério utilizado para definir o melhor particionamento, e portanto, o critério de otimização do particionamento, é o percentual de VNFs que puderam ser implementadas nessa rede.

4 PROPOSTA DE SOLUÇÃO

Para solucionar o problema estabelecido, propõe-se o desenvolvimento de um programa de otimização de particionamento de FPGAs ciente de topologia pelo emprego de um algoritmo genético especificamente criado para esse contexto. A escolha se deu pelo fato de algoritmos genéticos terem a capacidade de buscar soluções a problemas de otimização de forma extremamente flexível, tanto quanto às características do problema (as quais influenciam a definição dos indivíduos da população usada no algoritmo) quanto à função de otimização (a função de *fitness*). Isso é extremamente importante para atacar o problema, pois deseja-se criar um programa que é capaz de particionar qualquer rede indiferentemente de sua topologia, da quantidade e modelos de FPGAs utilizados (assim como sua configuração), e do conjunto de VNFs que seriam implementadas e o conjunto de requisições solicitadas.

4.1 Definição das características do algoritmo genético

O algoritmo genético desenvolvido para esse trabalho tem a seguinte lista de características:

1. Individuo

    A. Descrição geral

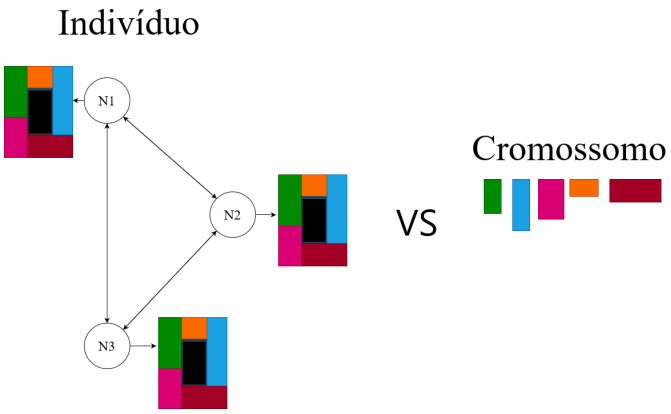
        O indivíduo representado no algoritmo genético é a rede de computadores como um todo. Essa rede segue a mesma composição descrita no capítulo 3 e ilustrada pela Figura 3.1

    B. Cromossomo

        No algoritmo desenvolvido nesse trabalho, os cromossomos representam as partições dos FPGAs de cada nodo da rede. Devido à natureza dessas partições, elas são agrupadas por dispositivo FPGA. Contudo, no cenário de geração de indivíduos filhos e cruzamento de cromossomos, esse agrupamento não é considerado e os cromossomos de cada pai são cruzados independentemente de qual FPGA ou nodo da rede eles se encontram

A figura 4.1 ilustra a diferença entre um indivíduo e seus cromossomos.

Figura 4.1: Definição de indivíduo e cromossomo



2. População inicial

Dada uma população inicial de N indivíduos, cada indivíduo é inicializado iterativamente nodo por nodo. Emprega-se um algoritmo particionador aleatório que para cada FPGA de um nodo gera um conjunto de partições de tamanho e posição aleatórios. Esse conjunto consiste de partições de três tamanhos pré-estabelecidos (grande,médio e pequeno), definidos a partir das VNFs utilizadas. É esse particionamento inicial que provê a diversidade genética inicial da população em questão.

3. Mutações

No algoritmo genético desenvolvido para esse trabalho, decidiu-se implementar dois tipos de mutações sob a hipótese que cada tipo teria efeitos diferentes no tempo de convergência e *fitness* máximo obtidos. As Figuras 4.2 e 4.3 ilustram o seu funcionamento.

A. Mutação tipo 1

O primeiro tipo de mutação, também chamado de mutação por realocação, consiste de executar novamente o particionador aleatório empregado na geração inicial da população sob a respectiva rede. A hipótese na qual se baseou a decisão de implementar essa mutação foi a de que, após o processo de crossover e combinação de diferentes partições de cada rede pai, seriam formadas lacunas na malha dos dispositivos FPGAs da rede filha grandes o suficiente para comportar uma nova partição de pelo menos de um dos três tamanhos mencionados anteriormente.

B. Mutação tipo 2

Já o segundo tipo de mutação, também chamado de mutação por redimensionamento, não faz nenhuma tentativa de alocar novas partições, mas sim de aumentar as partições existentes sob a hipótese de que a malha FPGA dos dispositivos da rede filha possui áreas não ocupadas entre as diferentes regiões alocadas, para as quais pode-se expandir cada partição a fim de aumentar a ocupação total da malha FPGA.

Figura 4.2: Mutação Tipo 1 : realocação

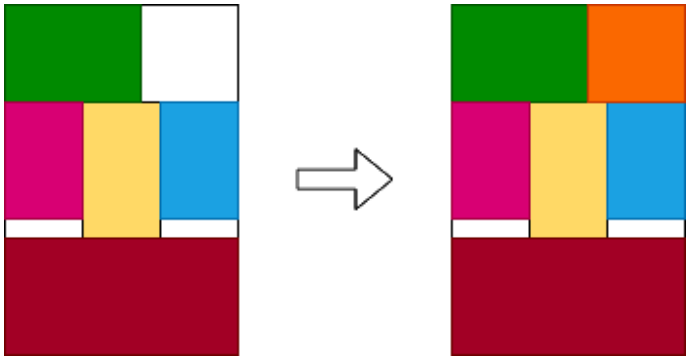
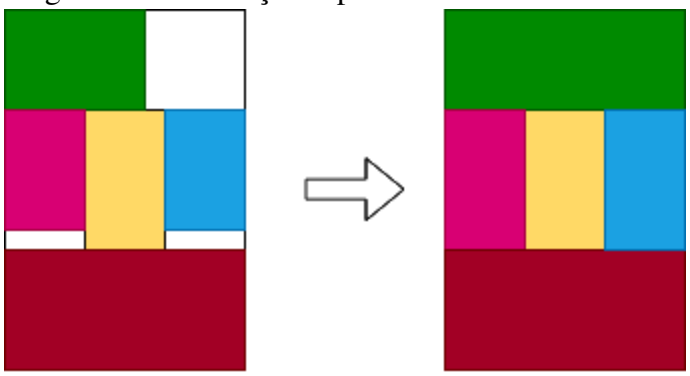


Figura 4.3: Mutação Tipo 2 : redimensionamento



4. Crossover

Assim como introduzido brevemente nas seções anteriores, o crossover nessa implementação consiste de coletar todas as partições alocadas dos FPGAs da rede pai P1 e da rede pai P2 e alocar, FPGA por FPGA da rede filha as partições de forma alternada. Isto é, aloca-se uma partição aleatória de P1 seguida de uma partição aleatória de P2 até que se tenha tentado alocar todas as partições dos pais. Durante essa etapa do crossover, partições alocadas são excluídas da lista de partições disponíveis e portanto não há a possibilidade da mesma partição ser alocada mais que uma vez. Após exaurir as opções de partições a serem alocadas no respectivo FPGA do indíviduo filho, itera-se para o próximo FPGA até que todos os dispositivos FPGAs

do filho estejam alocados. É importante frisar, assim como mencionado no capítulo 3.1 que descreve o ambiente de execução, que os indivíduos criados durante a execução do algoritmo genético possuem a mesma topologia de rede, a qual foi definida nos arquivos de entrada do algoritmo.

5. Fitness

   O *Fitness* de cada indivíduo é calculado considerando uma lista alvo de requisições de tamanho arbitrário $TReq$. Cada requisição desse conjunto é formada por uma ou mais VNFs dentro de um conjunto pré-determinado, interligadas em uma SFC. Dada essa lista de requisições, executa-se um alocador de requisições sobre a rede a ser avaliada e obtém-se o número de requisições alocadas com sucesso. A porcentagem de requisições implementadas com sucesso, isto é, $\frac{Req\quad Alocadas}{TReq}$, representa o *fitness* do indivíduo. É importante denotar que , para que o *fitness* de indivíduos gerados durante diferentes execuções possam ser de fato comparados, é de extrema importância que o tamanho da lista de requisições usadas no cálculo do fitness seja o mesma em todas as execuções.

6. Critério de parada

   A. Solução ideal

      A solução ideal, dado o cálculo de fitness descrito anteriormente, seria um indivíduo que possui um fitness de 1, representando 100% de alocação da lista de requisições alvo e portanto um *fitness* perfeito. O algoritmo termina sua execução ao encontrar um indivíduo que se encaixa nessa característica.

   B. Limite de gerações

      Dado um limite de gerações G, o algoritmo genético é executado para o número especificado de gerações, independentemente do tempo necessário para sua execução.

   C. Limite de tempo

      Dado um limite de tempo $TExec$, o algoritmo genético é executado dentro do intervalo de tempo especificado, terminando sua execução após a geração durante a qual excedeu-se o limite de tempo. Tal consideração foi julgada necessária para evitar o comportamento inesperado do algoritmo caso haja o término da execução durante a criação de uma geração.

4.2 Comportamento e funcionalidades do algoritmo genético

Tendo em vista as características definidas na seção anterior, o algoritmo foi desenvolvido em duas partes principais: a geração da população inicial e a aplicação do algoritmo genético sobre essa população gerada. A Figura 4.4 ilustra a relação entre as duas etapas e o funcionamento geral do algoritmo Na primeira etapa, ilustrada pelo Algoritmo 1 e pela Figura 4.5, dada uma população inicial de tamanho N, para a criação de cada indivíduo, pega-se o arquivo descritor de rede fornecido como entrada e gera-se uma nova rede usando esse descritor como template. Nessa rede, os FPGAs seguem o modelo descrito na entrada e não possuem nenhuma partição além da estática configurada. Em seguida, de forma iterativa percorre-se cada nodo dessa rede e para cada FPGA realiza-se um particionamento aleatório seguindo um conjunto de tamanhos pré-definido (grande, médio e pequeno). Com essa etapa finalizada, roda-se o algoritmo avaliador de *fitness* criado com base no algoritmo alocador de requisições de (GUERRA, 2023) mencionado anteriormente.

Algorithm 1: Inicialização da população do algoritmo genético

Input : Network and topology descriptor (NTOP), FPGA model descriptor (FMOD), Partition Size List (PSIZE), Request list (R) and population size (POPSIZE)

Output : A population of network descriptors with all FPGAs partitioned with PRRs

```
NetworkPopulation ← []
for i ← 0 to POPSIZE do
    network← InitializeEmptyNetworkFromTopology(NTOP, FMOD);
    for each node in network.nodes do
        for each fpga in node. fpgas do
            fpga.partitions ← RandomPartitioner(fpga,PSIZE);
    network.fitness ← NetworkEvaluator(network,R);
    NetworkPopulation.append(network)
```

Já no algoritmo genético propriamente dito, representado pelo Algoritmo 2 e pela Figura 4.6 , dada uma população inicial previamente gerada, realizam-se os processos de seleção de pais, crossover e mutação para a criação de novas gerações. O processo

de seleção de pais usa um simples algoritmo de sorteamento de elementos numa lista com a probabilidade baseada em pesos, nesse caso o *fitness* da rede pai. Isso garante a característica importante do algoritmo genético dos indivíduos mais aptos terem maior probabilidade de passarem seus cromossomos para a próxima geração. Além disso, esse algoritmo implementa um grupo de elite, o qual se define como um grupo de N indivíduos de maior *fitness* que garantidamente sobreviverão a iteração atual e serão passados a próxima. Já os processos de crossover e mutação seguem os requisitos definidos no capítulo 4. O crossover ocorre da seguinte forma: pega-se todas as partições do pai 1 (P1) e pai 2 (P2) e iterativamente percorre-se cada FPGA presente na rede filha (a qual foi inicializada, mas não recebeu nenhuma alocação de partição em seus FPGAs), alocando-se alternadamente as partições de P1 e P2 e removendo as partições alocadas da lista de partições disponíveis para os demais FPGAs. Já o processo de mutação manteve em seu projeto os dois tipos de mutação descritos no capítulo 4. Durante a geração dos filhos, é possível que o indivíduo sofra os dois tipos de mutação, sendo a mutação por realocação dada a prioridade nesse caso.

Algorithm 2: Execução do algoritmo genético sobre a população inicial

```
Input  : NetworkPopulation, EliteSize, ReallocationRate, ResizeRate, Stop
         Condition,Stop Value,NTOP and FMOD
Output: The Network descritor with the highest fitness
STOP ← False;
generation ← 0;
while STOP == False do
    if (Stop condition is TIME and ElapsedTime > Stop Value) OR (Stop
     condition is GENERATIONS and generation > Stop Value) then
        End Genetic Algorithm;
    EliteGroup ← EliteSize networks with the highest fitness from
     NetworkPopulation;
    NonEliteGroup ← (NetworkPopulation - EliteGroup);
    ChildPopulation ← [];
    for i ← 0 to Length(NonEliteGroup) do
        P1,P2 ← SelectParents(NetworkPopulation);
        ParentPartitions ← Concatenate(P1.partitions,P2.partitions);
        ParentPartitions ← Shuffle(ParentPartitions);
        child ← InitializeEmptyNetworkFromTopology(NTOP,FMOD);
        for each node in child.nodes do
            for each fpga in node.fpgas do
                for each partition in ParentPartitions do
                    if fpga.isAllocable(partition) == True then
                        fpga.allocate(partition);
                        ParentPartitions.remove(partition);
        child ← ReallocMutation(child) with ReallocationRate% chance;
        child ← ResizeMutation(child) with ResizeRate% chance;
        child.fitness ← NetworkEvaluator(network,R);
        ChildPopulation.append(child)
    NetworkPopulation ← Concatenate(EliteGroup, ChildPopulation);
    generation ← generation+1;
```

Figura 4.4: Fluxograma geral do algoritmo genético

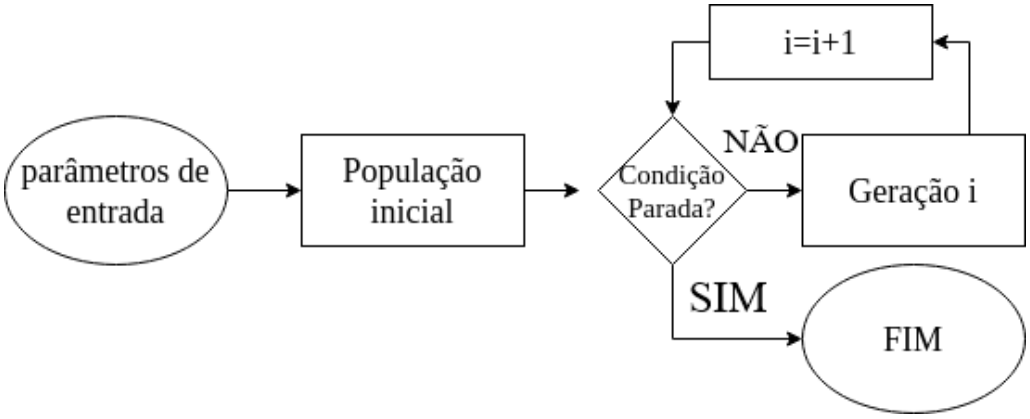


Figura 4.5: Fluxograma da criação da população inicial

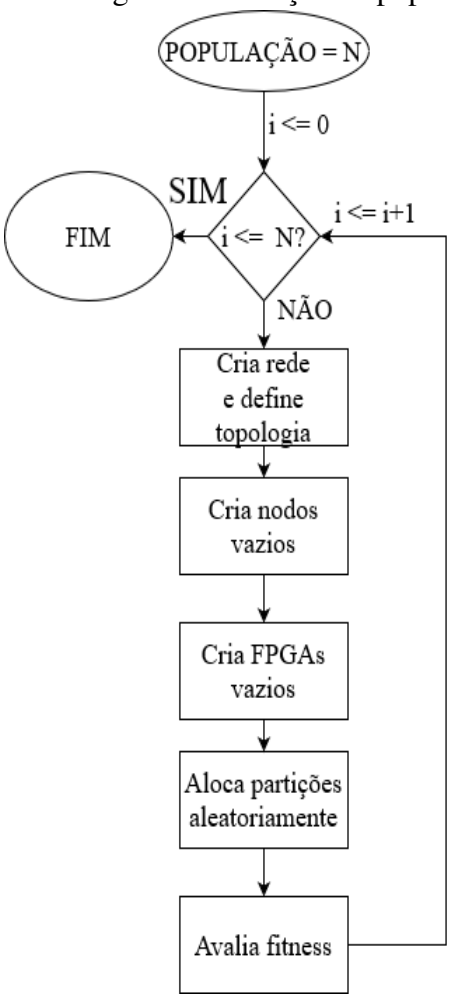
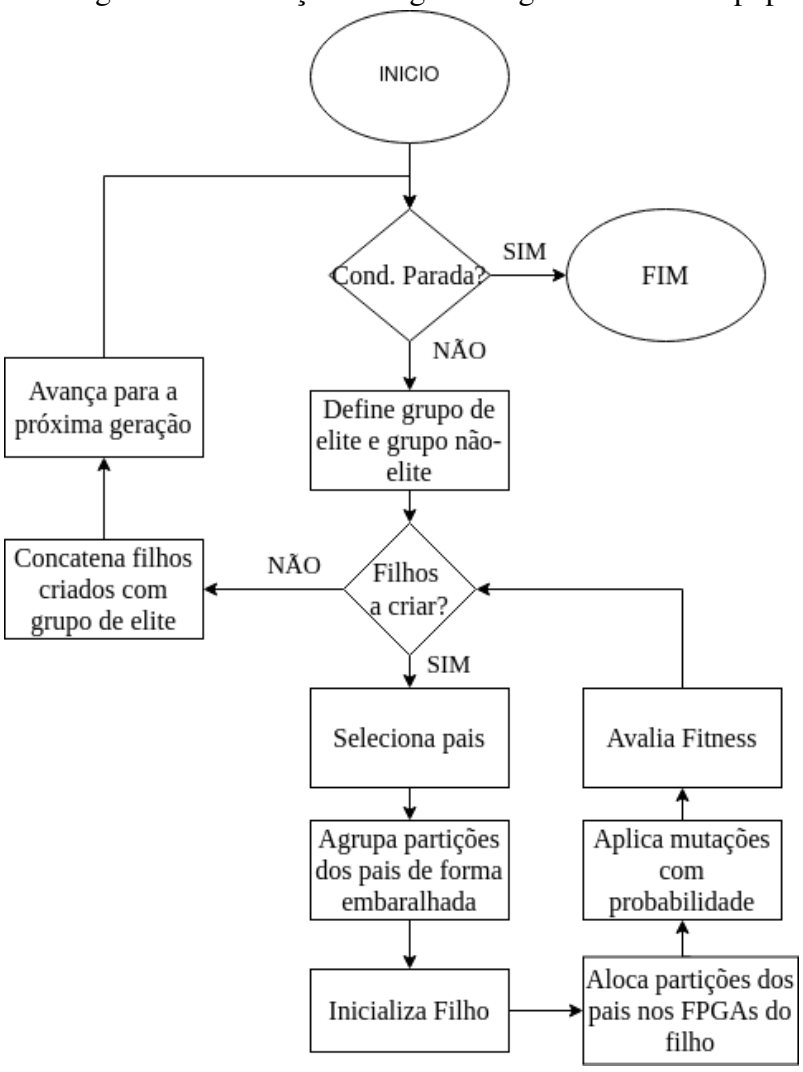


Figura 4.6: Fluxograma da execução do algoritmo genético sobre a população inicial



5 METODOLOGIA

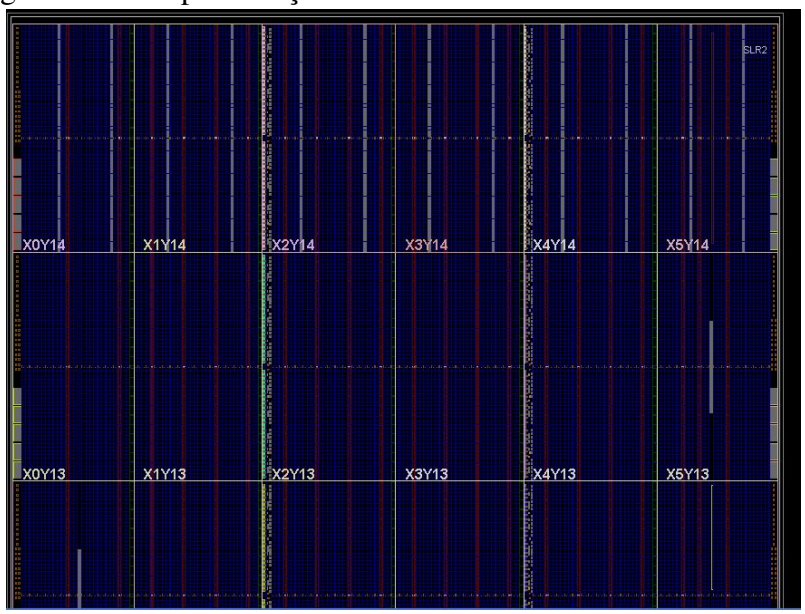
Tendo em vista a modelagem do problema e a definição do ambiente de execução delimitadas no capítulo 3, assim como a especificação proposta do algoritmo genético a ser utilizado no trabalho, realizou-se a implementação desse algoritmo genético na linguagem Python. Os testes e simulações necessários para o desenvolvimento também foram realizados nessa linguagem. Sua escolha ocorreu pela flexibilidade e facilidade no desenvolvimento de software providos pela linguagem, assim como o suporte da comunidade por meio de bibliotecas que oferecem soluções para boa parte das funcionalidades necessárias para o funcionamento e implementação do software. No que se trata dos arquivos de entrada para o algoritmo genético, os arquivos descritores de rede e sua topologia foram gerados previamente a execução dos testes de forma aleatória conforme os seguintes critérios:

1. Número de nodos igual a 10 ou 30
2. Número de links (conexões entre nodos) de 1,2 a 1,3 vezes o número de nodos a fim de garantir um grafo conexo. Isto é, 12 a 13 links para as redes de 10 nodos, e 36 a 39 links para as de 30 nodos.
3. 0 a 3 dispositivos FPGAs por nodo

Sob esses critérios obteve-se dois grupos de redes a serem otimizadas pelo algoritmo genético. A escolha dos tamanhos de redes para a realização do teste se justifica pelo fato que, assim como os trabalhos de (YI et al., 2018), (MOENS; TURCK, 2014), (BENSON; AKELLA; MALTZ, 2010) e (LUIZELLI et al., 2015) descrevem tamanhos realistas para redes de universidades de pequeno porte, assim como os backbones de pequenas ISPs. A geração dos arquivos descritores de rede, assim como as listas de requisições e VNFs e a avaliação de *fitness* das redes foram feitas por meio de um conjunto de scripts escritos em Python providos por (GUERRA, 2023). Já o arquivo descritor do modelo de FPGA utilizado, o VU190 da Xilinx, foi obtido através da combinação das informações do dispositivo providas em (XILINX, 2022) listadas na Tabela 5.1 e da malha FPGA provida pelo software Vivado, o qual contém entre suas funcionalidades a exibição da distribuição e do posicionamento dos recursos do dispositivo, ilustrada na Figura 5.1. Esse arquivo contém também as informações da região estática que será utilizada para todos os dispositivos. A escolha de apenas um modelo de FPGA ocorreu para fins de simplificação do desenvolvimento inicial do algoritmo, cujo foco esteve na implementação de uma versão

inicial funcional do programa a fim de provar a hipótese sugerida na concepção desse trabalho, relegando a trabalhos futuros a expansão da lista de dispositivos FPGA suportados. Outra possibilidade de se explorar em próximos trabalhos é o emprego de mais de uma variante de região estática, a qual pode inclusive ser outro parâmetro a ser otimizado no algoritmo genético junto do particionamento das regiões reconfiguráveis.

Figura 5.1: Representação da malha FPGA no software VIVADO



A fim de determinar a eficiência e capacidade desse algoritmo particionador ciente de topologia, realizou-se dois tipos de testes. A primeira etapa de testes consistiu de dois testes preliminares com o objetivo de reduzir o escopo dos parâmetros de controle do algoritmo genético. O primeiro desses testes almejou comparar o impacto de cada tipo de mutação. Utilizando como base redes de 10 nodos, executou-se o particionador ciente de topologia 3 mil vezes com diferentes combinações de mutação por realocação e por redimensionamento de forma a ter uma cobertura ampla do espaço de solução. A motivação para esses testes preliminares foi a de reduzir o tamanho do vetor de entrada a ser explorado na segunda bateria de testes, pois se uma determinada combinação resulta em *fitness* comparativamente pior, não seria necessário testá-la novamente no futuro.

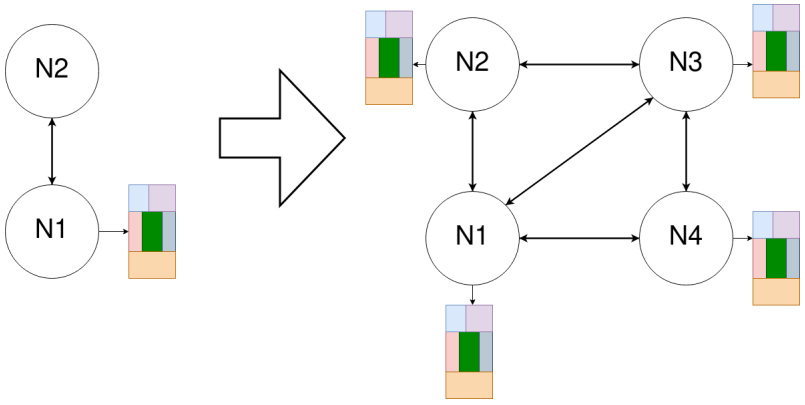
Tendo feito esse teste, o desenvolvimento do otimizador de particionamento passou a focar em avaliar o possível aumento de desempenho trazido por incorporar a ciência de topologia na otimização, isto é, comparar o desempenho de uma versão agnóstica a topologia (que seria a abordagem atual encontrada na literatura) com a versão ciente que está sendo implementada. Para criar a versão agnóstica do particionador criou-se uma rede seguindo uma topologia mínima, isto é, contendo apenas 2 nodos com um único

Tabela 5.1: Recursos e funcionalidades do FPGA

	VU190
Células lógicas do sistema	2.349.900
Flip-Flops do CLB	2.148.480
LUTs do CLB	1.074.240
Máxima RAM distribuída (Mb)	14,5
Blocos de BRAM	3.780
BRAM (Mb)	132,9
CMTs	30
DLLs de E/S	120
Máximo de E/S HP	650
Máximo de E/S HR	52
Slices DSP	1.800
Monitor de Sistema	3
PCIe Gen3 x8	6
150G Interlaken	9
100G Ethernet	9
Transreceptores GTH 16.3Gb/s	60
Transreceptores GTY 30.5Gb/s	60
PLLs Fracionados dos Transceptores	30

FPGA, e rodou-se o algoritmo genético sobre essa rede mínima. Além disso, para minimizar o impacto da latência e vazão do link (o que é indesejado, pois o objetivo é gerar um particionamento sem nenhuma influência da topologia) Obtendo o particionamento otimizado resultante do único dispositivo FPGA presente, replicou-se esse particionamento na rede alvo de teste para todos os seus FPGAs. Dessa forma, a versão agnóstica gera uma rede cujos FPGAs possuem todos as mesmas partições pois eles espelham um único particionamento, enquanto que a versão ciente realiza a otimização do particionamento em todos os FPGAs de forma conjunta, mas independente. O processo é ilustrado pela Figura 5.2

Figura 5.2: Processo de criação da rede mínima e extrapolação para rede a ser avaliada



Dessa forma, o segundo teste preliminar, já levando em consideração os resultados

do primeiro, focou em escolher um conjunto de parâmetros a ser usado com a versão ciente e outro para a versão agnóstica durante os testes comparativos entre as duas versões. Nesse teste, foram criadas 10 redes de 10 nodos e 10 redes de 30 sobre as quais as duas versões do otimizador de particionamento foram executadas. Além disso, as redes criadas foram parametrizadas para terem o número de links igual a 1,2 a 1,3 vezes o número de nodos (ou seja, as redes de 10 nodos tinham de 12 a 13 links, e as de 30 nodos de 36 a 39 links). Utilizando os 7 conjuntos de parâmetros descritos na Tabela 5.2, executou-se o particionador ciente e agnóstico sobre cada rede, totalizando 70 execuções. Com os dados obtidos durante essas execuções pôde-se escolher um conjunto de parâmetros para a versão ciente e um para a versão agnóstica.

Tabela 5.2: Conjunto de parâmetros de controle do GA. Valores em escala 0 a 1

	POPSIZE	ELITE	RESIZE
C1	300	0,1	0,5
C2	150	0,1	0,5
C3	450	0,1	0,5
C4	300	0,2	0,5
C5	300	0,04	0,5
C6	300	0,1	0,2
C7	300	0,1	0,8

Já a segunda etapa teve como objetivo comparar o particionador ciente de topologia com o particionador que é agnóstico a topologia. A fim de garantir uma comparação justa nesses testes entre as duas versões, estipulou-se uma condição de parada por tempo em vez de por geração ou um *fitness* alvo. As duas versões rodam pelo mesmo intervalo, 1 hora para as redes de 10 nodos e 2 horas para as redes de 30 nodos. Sob essas condições as duas versões dispõe-se da mesma quantidade de recursos computacionais para sua execução.

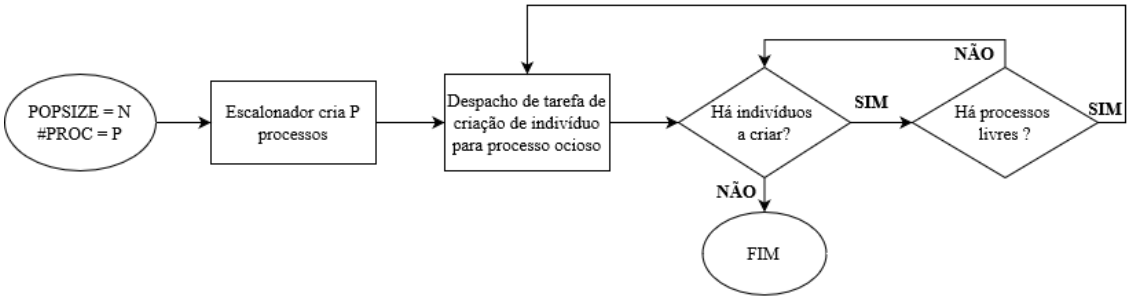
Tabela 5.3: Características do ambiente de teste

Nodos	CPU	RAM	Paralelismo?	#Processos	Tempo(min)
10	Intel i9 12900KF	32GB	Sim	24	60
30	Intel i9 12900KF	32GB	Sim	24	120

A Tabela 5.3 exemplifica as condições do ambiente de teste, incluindo a configuração da máquina na qual rodou-se as simulações, durante essa etapa. O paralelismo mencionado se refere a paralelização da criação de indivíduos na etapa de geração da população inicial e da etapa de formação dos indivíduos filhos para a criação da geração

seguinte durante a execução do algoritmo genético. Considerando uma população de tamanho N e 24 processos, faz-se o despacho da tarefa de criação de indivíduos por uma estrutura de fila da qual cada processo recebe uma tarefa por vez. Ou seja, os processos não são sincronizados entre si após o inicio da execução paralela. O único controle feito é no despacho de tarefas pelo escalonador e no acompanhamento da quantidade de tarefas restantes a serem concluídas (isto é, até que N indivíduos sejam criados). A Figura 5.3 ilustra como funciona a implementação.

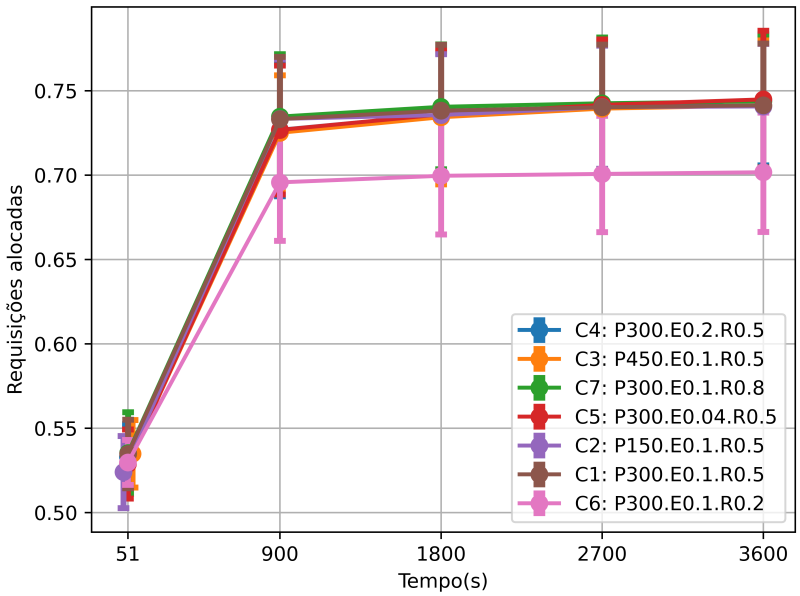
Figura 5.3: Fluxograma da execução paralela para uma geração



6 RESULTADOS

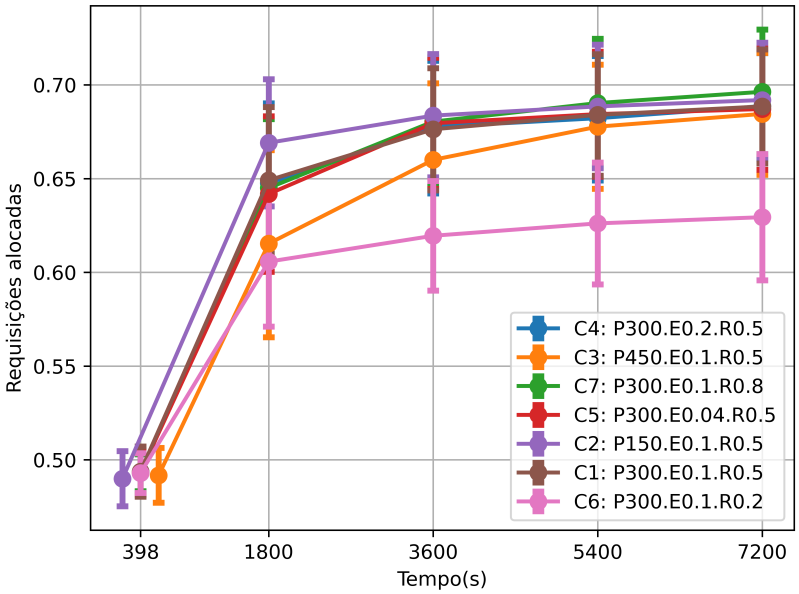
Dos testes preliminares, observou-se que apenas a mutação por redimensionamento teve impacto no *fitness* final na execução do algoritmo genético. Ao contrário das expectativas, a mutação por realocação não teve influência perceptível no algoritmo, independentemente do valor utilizado. Contudo, apesar dos resultados inesperados, isso trouxe uma positiva consequência aos experimentos: das duas mutações, a realocação era a mais intensiva computacionalmente, causando aumento no tempo necessário por geração significativamente. Dessa forma, o descarte dela trouxe simplificações e aumento de desempenho a execução dos testes. O outro teste preliminar, referente a escolha do conjunto de parâmetros para a comparação de versões, observou-se que o conjunto C7 da Tabela 5.2 teve a melhor média de *fitness* para as redes de 10 e 30 nodos da versão ciente de topologia, conforme a Figura 6.1 e a Figura 6.2.A escolha desse conjunto de parâmetros foi feita apesar do empate técnico entre alguns dos conjuntos, com base no valor médio. Já para a versão agnóstica, C7 e C1 tiveram um empate estatístico. Para fins de simplificação, escolheu-se o conjunto C7 para as duas versões.

Figura 6.1: Teste temporizado para escolha de parâmetros N10



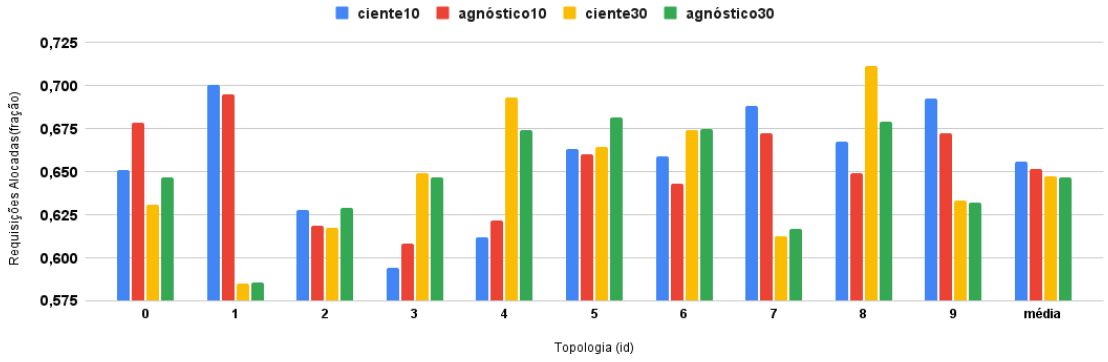
Já dos testes comparativos entre as versões agnóstica e ciente de topologia do algoritmo de otimização de particionamento, pode-se observar que ao mesmo tempo em que a abordagem ciente pode trazer um melhor particionamento que justifica sua maior complexidade e uso de recursos computacionais, não é garantido que ela produza um

Figura 6.2: Teste temporizado para escolha de parâmetros N30



resultado superior à agnóstica. Conforme ilustrado na Figura 6.3, para certas topologias a versão agnóstica gerou particionamentos com maior taxa de alocação de requisições. Enquanto que para as redes de 10 nodos a versão ciente produziu resultados melhores em 7 das 10 redes testadas (e por consequência uma média de alocações nitidamente maior), para as redes de 30 nodos o resultado foi muito mais próximo entre as versões. A versão ciente venceu em apenas 4 de 10 redes, e empatou em duas,o que se reflete na média estatisticamente empatada entre as versões.

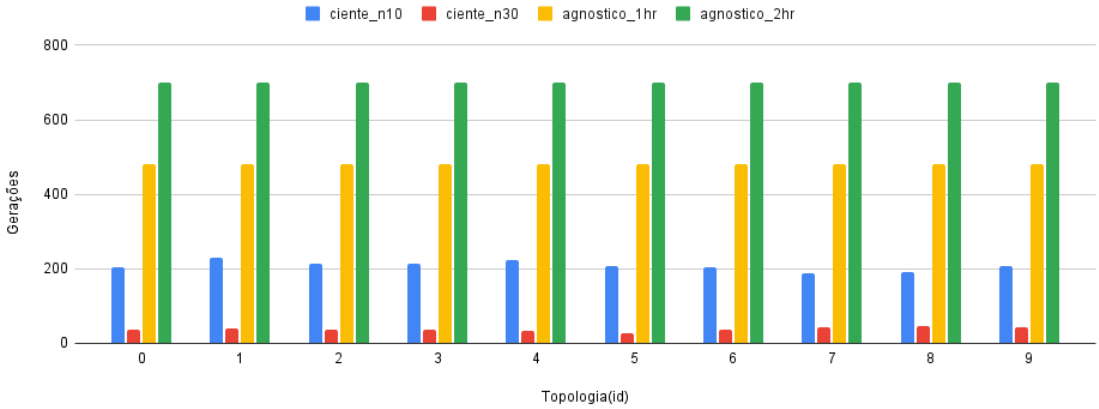
Figura 6.3: Desempenho em alocações para 10 e 30 nodos



Uma das hipóteses levantadas para explicar o porquê da versão ciente não apresentar resultados superiores em todos os casos, além da diferença de desempenho conforme o tamanho das redes testadas, está na quantidade de gerações desenvolvidas durante a execução do algoritmo genético sob cada versão do otimizador de particionamento. Con-

forme pode-se observar na Figura 6.4, a versão ciente produziu um número substancialmente menor de gerações durante a otimização das redes de 30 nodos, o que demonstra a baixa escalabilidade do algoritmo genético desenvolvido durante esse trabalho. Um aumento de 3 vezes no tamanho da rede trouxe uma performance de 5 a 7 vezes pior, comparando o desempenho para 10 e 30 nodos.

Figura 6.4: Quantidade de gerações para 10 e 30 nodos



Ao trazer a versão agnóstica para essa comparação na quantidade de gerações desenvolvidas, o problema de escalabilidade se mostra ainda mais perceptível. Para as redes de 10 nodos, a versão ciente desenvolveu aproximadamente a metade das gerações da versão agnóstica e venceu 7 de 10 comparações. Já para 30 nodos, a ciente criou somente entre 5% 6% da quantidade de gerações observadas na versão agnóstica. É possível que tal diferença de desempenho do algoritmo genético conforme o tamanho da rede sobre a qual executa seja a causa dos resultados aquém do esperado da versão ciente. Contudo, não pode-se descartar a hipótese que outros fatores presentes no algoritmo sejam os culpados. Ou mesmo que a abordagem por algoritmo genético não seja a mais adequada para redes superiores a um certo tamanho de nodos. É, de qualquer forma, um objeto promissor de estudo para trabalhos futuros e uma fonte de motivação para a busca de implementações mais eficientes e de melhor escalabilidade do algoritmo genético desenvolvido neste trabalho.

Além da quantidade de gerações desenvolvidas, outra hipótese inicialmente elaborada para explicar os resultados obtidos foi a de que certas características das redes testadas levaram a discrepâncias nos resultados das duas versões. As duas características investigadas foram a quantidade média de FPGAs por nodo em cada rede e a quantidade de links presentes na rede. No que se refere a quantidade de FPGAs, observou-se que nos casos de teste em que o particionador ciente se saiu melhor, a média de FPGAs por

nodo foi ligeiramente menor. Uma suposição feita em virtude dessa observação foi a de que, havendo um número maior de FPGAs em um nodo para alocar requisições, a versão agnóstica tem maior espaço para desperdiçar. Isto é, num caso de 2 FPGAs por nodo a demanda por um particionamento eficiente é menor caso haja apenas um. Tal suposição, no entanto, implica que o dimensionamento das listas de requisições utilizadas para os testes não foram adequadas e introduziram um certo viés conforme a quantidade de FPGAs por nodo, o que vai contra a metodologia de comparação justa de versões. Contudo, não foi possível investigar mais a fundo essa possível falha metodológica. É, apesar de tudo, uma segunda fonte de estudo para trabalhos futuros.

Tabela 6.1: Média de FPGAs por nodo (N=10)

Ciente melhor	Agnóstico melhor
1,3	1,4

Tabela 6.2: Média de FPGAs por nodo(N=30)

	Ciente melhor	Agnóstico melhor	Empate
Média	1,583	1,8	1,267

Já a quantidade de links não apresentou nenhuma informação relevante que pudesse ser usada como base para sustentar alguma hipótese que explicasse os resultados obtidos. Enquanto que para as redes de 10 nodos a quantidade de links foi maior nos casos em que a versão ciente se mostrou melhor, o oposto se observou para as de 30 nodos. Não foi possível extrair dos descritores de rede outras informações sobre os links que pudessem alterar essa análise. Contudo existe a possibilidade que uma investigação mais minuciosa das topologias de cada rede apresente uma diferente perspectiva quanto ao impacto da quantidade de links.

Tabela 6.3: Média de links por nodo (N=10)

Ciente melhor	Agnóstico melhor
13	12,33

Tabela 6.4: Média de links por nodo (N=30)

Ciente melhor	Agnóstico melhor	Empate
36,5	37,5	37,5

7 CONCLUSÃO

Os resultados obtidos no curso do desenvolvimento desse trabalho demonstram a viabilidade da abordagem ciente de topologia para o particionamento de FPGAs inseridos em rede. Embora sua eficácia não tenha atingido plenamente os objetivos elaborados na proposta do trabalho, a análise dos resultados apontou potenciais pontos de melhoria na implementação do algoritmo genético empregado que podem ser explorados mais a fundo em trabalhos futuros. A performance no que tange o uso de recursos computacionais e a escalabilidade do algoritmo demonstraram ser as principais características negativas da versão atual do código elaborado, e portanto o foco primário para uma próxima etapa de desenvolvimento. Uma das rotinas mais caras computacionalmente na execução do algoritmo foi a função de *fitness*, a qual percorre os diferentes caminhos entre nodos para verificar a possibilidade de alocação das requisições. Uma implementação alternativa que reduziria o seu custo ao passo que sua precisão seja minimamente afetada seria uma função de avaliação aproximada, sendo a função de avaliação completa reservada a ser usada apenas em pontos específicos da execução do algoritmo.

Outro ponto a ser explorado futuramente é a expansão do trabalho para contemplar outros modelos de FPGA, assim como uma variedade maior de tamanhos pré-definidos de partições a ser utilizados e da lista de requisições e VNFs. O uso de diferentes regiões estáticas (a qual foi reduzida para uma única variante durante esse trabalho por questões de simplificação) também deve ser considerado em versões posteriores. Também não pode se descartar a possibilidade de que o emprego de outros tipos algoritmos meta-heurísticos, como simulated annealing, ant colony ou demais algoritmos evolucionários. Tal implementação pode ser feita tanto em substituição do algoritmo genético ou em combinação. Contudo, tal mudança drástica de metodologia necessitaria de um foco exclusivo para sua efetiva implementação.

REFERÊNCIAS

ALMEIDA, L. F. et al. Moving nfv toward the antenna through fpga-based hardware reconfiguration. **IEEE Communications Letters**, v. 27, n. 1, p. 342–346, 2023.

BANERJEE, P.; SANGTANI, M.; SUR-KOLAY, S. Floorplanning for partially reconfigurable fpgas. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 30, n. 1, p. 8–17, 2011.

BENSON, T.; AKELLA, A.; MALTZ, D. Network traffic characteristics of data centers in the wild. In: . [S.l.: s.n.], 2010. p. 267–280.

BOUTROS, A.; BETZ, V. Fpga architecture: Principles and progression. **IEEE Circuits and Systems Magazine**, v. 21, n. 2, p. 4–29, 2021.

DHAR, A. et al. Dml: Dynamic partial reconfiguration with scalable task scheduling for multi-applications on fpgas. **IEEE Transactions on Computers**, v. 71, n. 10, p. 2577–2591, 2022.

FAROOQ, U.; MARRAKCHI, Z.; MEHREZ, H. Fpga architectures: An overview. In: ____. **Tree-based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization**. New York, NY: Springer New York, 2012. p. 7–48. ISBN 978-1-4614-3594-5. Available from Internet: <https://doi.org/10.1007/978-1-4614-3594-5_2>.

GALEA, F.; CARPOV, S.; ZAOURAR, L. Multi-start simulated annealing for partially-reconfigurable fpga floorplanning. In: **2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)**. [S.l.: s.n.], 2018. p. 1335–1338.

GUERRA, V. S. **Alocação Realista de Funções de Rede Virtualizadas em FPGAs Reconfigurados Dinamicamente**. Trabalho de Conclusão em Engenharia de Computação — UFRGS, 2023.

HASSAN, A. et al. Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on fpga. In: **2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)**. [S.l.: s.n.], 2015. p. 183–186.

HERRERA, J. G.; BOTERO, J. F. Resource allocation in nfv: A comprehensive survey. **IEEE Transactions on Network and Service Management**, v. 13, n. 3, p. 518–532, 2016.

HOLLAND, J. **Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**. University of Michigan Press, 1975. ISBN 9780472084609. Available from Internet: <https://books.google.com.br/books?id=YE5RAAAAMAAJ>.

JORDAN, M. G. et al. Muteco: A framework for collaborative allocation in cpu-fpga multi-tenant environments. In: **2021 34th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2021. p. 1–6.

KACHRIS, C.; SIRAKOULIS, G.; SOUDRIS, D. Network function virtualization based on fpgas:a framework for all-programmable network devices. 06 2014.

KRAMER, O. Genetic algorithms. In: ____. **Genetic Algorithm Essentials**. Cham: Springer International Publishing, 2017. p. 11–19. ISBN 978-3-319-52156-5. Available from Internet: <https://doi.org/10.1007/978-3-319-52156-5_2>.

LAMBORA, A.; GUPTA, K.; CHOPRA, K. Genetic algorithm- a literature review. In: **2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)**. [S.l.: s.n.], 2019. p. 380–384.

LUIZELLI, M. C. et al. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In: IEEE. **2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.], 2015. p. 98–106.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys & Tutorials**, v. 18, n. 1, p. 236–262, 2016.

MOENS, H.; TURCK, F. D. Vnf-p: A model for efficient placement of virtualized network functions. In: IEEE. **10th international conference on network and service management (CNSM) and workshop**. [S.l.], 2014. p. 418–423.

MURRAY, K. E.; BETZ, V. Hetris: Adaptive floorplanning for heterogeneous fpgas. In: **2015 International Conference on Field Programmable Technology (FPT)**. [S.l.: s.n.], 2015. p. 88–95.

NIEMIEC, G. S. et al. A survey on fpga support for the feasible execution of virtualized network functions. **IEEE Communications Surveys & Tutorials**, v. 22, n. 1, p. 504–525, 2020.

PAOLINO, M.; PINNETERRE, S.; RAHO, D. Fpga virtualization with accelerators overcommitment for network function virtualization. In: **2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)**. [S.l.: s.n.], 2017. p. 1–6.

RABOZZI, M.; LILLIS, J.; SANTAMBROGIO, M. D. Floorplanning for partially-reconfigurable fpga systems via mixed-integer linear programming. In: **2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines**. [S.l.: s.n.], 2014. p. 186–193.

RABOZZI, M.; MIELE, A.; SANTAMBROGIO, M. D. Floorplanning for partially-reconfigurable fpgas via feasible placements detection. In: **2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines**. [S.l.: s.n.], 2015. p. 252–255.

RINGLEIN, B. et al. System architecture for network-attached fpgas in the cloud using partial reconfiguration. In: **2019 29th International Conference on Field Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2019. p. 293–300.

RUAN, J. et al. Increasing flexibility of cloud fpga virtualization. In: **2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2022. p. 350–357.

SRINIVAS, M.; PATNAIK, L. Adaptive probabilities of crossover and mutation in genetic algorithms. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 24, n. 4, p. 656–667, 1994.

WANG, J. et al. Duprfloor: Dynamic modeling and floorplanning for partially reconfigurable fpgas. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 40, n. 8, p. 1613–1625, 2021.

XILINX. **UltraScale Architecture and Product Data Sheet: Overview**. [S.l.], 2022. Available from Internet: <https://www.mouser.com/datasheet/2/903/ds890_ultrascale_overview-1591529.pdf>.

XILINX. **Vivado Design Suite User Guide: Dynamic Function eXchange**. [S.l.], 2023. Available from Internet: <https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration/Design-Considerations-and-Guidelines-for-7-Series-and-Zynq-Devices>.

YI, B. et al. A comprehensive survey of network function virtualization. **Computer Networks**, v. 133, p. 212–262, 2018. ISSN 1389-1286. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1389128618300306>.