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

VICTOR SBERSE GUERRA

Alocação Realista de Funções de Rede Virtualizadas em FPGAs Reconfigurados Dinamicamente

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

Orientador: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
2023

“You can avoid reality,
but you cannot avoid the consequences of avoiding reality.”
— Ayn Rand

AGRADECIMENTOS

Agradecer primeiramente à minha família que sempre me apoiou de inúmeras formas para que eu chegasse até aqui. Agradecer também meus amigos de longa data que sempre estiveram ao meu lado e a todos os novos que fiz nessa jornada, sou grato a cada um de vocês. Um agradecimento a todos os professores com quem tive o prazer de cruzar o caminho nesses anos, e, em especial, ao orientador desse trabalho, por me guiar nessa etapa final.

RESUMO

Utilizar de Field Programmable Gate Arrays (FPGAs), um tipo de hardware flexível, para soluções de virtualização de funções de rede (NFV), traz uma série de vantagens econômicas e práticas para quem as utiliza. Este trabalho tem por objetivo maximizar o uso de recursos de servidores, buscando aumentar sua eficiência, a fim de alocar o maior número possível de Funções de Rede Virtualizadas (VNFs) executadas em FPGAs, podendo elas serem diferentes ou não, devido à capacidade deste tipo de hardware de realocar recursos dinamicamente de acordo com as necessidades. Assim, é possível desenvolver heurísticas para alocar funções de rede, para que se explore o maior ganho possível dada uma infraestrutura heterogênea para virtualização de funções de rede. Será demonstrado que heurísticas simplificadas existentes na literatura, chegam frequentemente a soluções inválidas por não considerarem a existência de partições. Será desenvolvida uma heurística gulosa para alocar VNFs em infraestrutura com FPGAs de forma mais realista, considerando a existência de partições.

Palavras-chave: FPGA. NFV. VNF. Posicionamento. Alocação. Particionamento.

Realistic Allocation of Virtualized Network Functions in Dynamically Reconfigured FPGAs

ABSTRACT

Using Field Programmable Gate Arrays (FPGAs), a type of flexible hardware, for network functions virtualization (NFV) solutions brings a number of economic and practical advantages to those who use them. This work aims to maximize the use of server resources, seeking to increase their efficiency, in order to allocate the largest possible number of Virtualized Network Functions (VNFs) executed in FPGAs, which may or may not be different, due to the capacity of this type of hardware to dynamically reallocate resources as needed. Thus, it is possible to develop heuristics to allocate network functions, in order to exploit the greatest possible gain given a heterogeneous infrastructure for virtualization of network functions. It will be shown that simplified heuristics found in the literature often arrive at invalid solutions because they do not consider the existence of partitions. A greedy heuristic will be developed to allocate VNFs in infrastructure with FPGAs in a more realistic way, considering the existence of partitions.

Keywords: FPGA, NFV,VNF, Placement, Allocation, Partitioning.

LISTA DE ABREVIATURAS E SIGLAS

FF	Flip-Flop
AES	Advanced Encryption Standard
CLB	Configurable Logic Block
CPU	Central Processor Unit
DFX	Dynamic Funtion Exchange
DNS	Domain Name System
DPI	Deep Packet Inspection
DSP	Digital Signal Processing
GPP	General-Purpose Processor
GPU	Graphic Processor Utit
HDL	Hardware Description Language
ILP	Integer Linear Programming
IOB	Input Output Block
LUT	Lookup Table
MUX	Multiplexador
NAT	Network Adress Translation
NFV	Network Function Virtualization
SFC	Service Function Chaining
VNF	Virtual Network Function
ASIC	Application-Specific Integrated Circuit
BRAM	Block Random Access Memory
COTS	Commercial Off The Shelf
FPGA	Field Programmable Gate Array
OPEX	Operational Expenditure
CAPEX	Capital Expenditure

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este trabalho considera a utilização de Field Programmable Gate Arrays (FPGAs) como hardware generalista principal para processamento das funções de rede, dado que este tipo de dispositivo possui características específicas como sua versatilidade, podendo atender diferentes tipos de funções, dada a sua composição de recursos internos. Possui também a capacidade de ser reprogramável, podendo em caso de necessidade ou demanda, alterar sua funcionalidade. Sua disponibilidade comercial também é relevante, facilitando o acesso a dispositivos desse tipo. Possui, ainda, frequentemente um desempenho superior a processadores de propósito geral, já que a implementação em FPGAs é feita em nível de hardware, enquanto que soluções puramente executadas via software podem apresentar um desempenho inferior quando comparados ambos os métodos. Com isto, esse tipo de dispositivo acaba por ter um grande potencial para aplicação em funções de rede, aliando desempenho com flexibilidade, como abordado por (NIEMIEC et al., 2020).

Dada a sua versatilidade, é possível realocar recursos para implementar funções diversas, que demandam especializações diferentes, como quantidades massivas de dados em paralelo, ou lidar com grande número de acessos a memória, por exemplo. Estas funções de rede consistem em serviços que garantem uma boa utilização de redes de computadores que são utilizadas diariamente por inúmeros usuários, como firewall, Deep Packet Inspection (DPI), Advanced Encryption Standard (AES), load balancing, entre outras.

Tendo os FPGAs como elemento base para a computação de dados, o modelo apresentado, do ponto de vista topológico, consiste em uma rede caracterizada por um grafo. Cada nodo pode conter recursos computacionais para o processamento destas funções, contando com um número finito dos mesmos, representando a quantidade de recursos disponíveis dentro de cada FPGA, respeitando as características físicas e de projeto de cada dispositivo. Os vértices desse grafo são os links que conectam os nodos uns aos outros, servindo de caminho para a transmissão de dados.

A motivação deste trabalho baseia-se em apresentar uma abordagem que leva em consideração as peculiaridades e características intrínsecas do FPGA. O que existe hoje, disponível na literatura por outros autores que também atacam este problema de virtualização de funções de rede, acaba por fazer simplificações técnicas que limitam sua aplicabilidade prática. Como este é um assunto que possui grande interesse econômico, dada

a expansão do uso de redes globalmente, aumentando a necessidade de se comunicar e processar cada vez mais dados, desejamos obter soluções válidas e eficazes.

É de suma importância frisar certos detalhes, visto que existem outras abordagens similares na literatura para este problema, que embora tragam um conceito semelhante, tratam as especifidades do hardware em questão de maneira simplificada. Tais abordagens consideram os recursos computacionais como se fossem acessíveis de qualquer maneira, utilizando-os de acordo com o necessário, o que na prática não é verdade, já que existe uma série de regras para se alocar estes recursos, que precisam respeitar as particularidades do hardware. De maneira simples, algumas dessas abordagens não seguem certas regras de alocação de recursos dentro dos FPGAs. Uma solução realista deve considerar a disposição espacial dos recursos ao longo do dispositivo, e pode não conseguir alocar, muitas vezes, somente o recurso que se necessita, acabando por ter que englobar outros recursos mesmo que não venham a ser utilizados. Essa consciência sobre regras para formações de partições de componentes é o que difere este trabalho dos demais. Com isso são denominadas e separadas as soluções baseadas na Abordagem Ciente de Partições e na Abordagem não-Ciente de Partições.

Assim, este trabalho vem por apresentar e discutir os resultados atingidos através de simulações considerando as duas abordagens. Demostrando que, embora estas abordagens não-cientes disponíveis na literatura agreguem do ponto de vista teórico, elas podem chegar em soluções inválidas do ponto de vista prático, já que por sua vez, não respeitam características físicas dos dispositivos. Acabam por apresentar números maiores de funções alocadas quando comparadas a abordagem deste trabalho, pois devido a simplificações na modelagem dos FPGAs, acabam por alocar uma quantidade de funções maior do que o hardware conseguiria suportar. Com isso, serão demonstrados exemplos válidos, mostrando uma alocação real em certos modelos de FPGAs. Tentando solucionar o problema de uma maneira eficiente, através de uma heurística de alocações de funções de rede em um layout de partições previamente estabelecido. Serão apresentadas comparações de resultados, exemplificando como a solução desse trabalho obtém soluções válidas para o problema de alocação de funções em FPGAs, respeitando as particularidades do hardware escolhido.

Por fim, a estrutura deste trabalho está separada em capítulos para o melhor acompanhamento da resolução do problema abordado. No capítulo 2 é apresentada a fundamentação teórica para o embasamento da solução, além da discussão de trabalhos já publicados na área. No capítulo 3 é explicado como é a linha de pensamento lógico por trás

da solução deste trabalho, abordando detalhes como estruturas, algoritmos e heurísticas de posicionamento utilizadas, debatendo usos e eficiências. No capítulo 4 é apresentada a metodologia utilizada para avaliação deste trabalho, entrando em detalhes e tomadas de decisões, com foco na parte das simulações do problema, exemplificando dados e demais informações que foram empregados para montar o modelo como um todo. No capítulo 5, apresentamos os resultados alcançados através das simulações e discutimos sua relevância. Por último, No capítulo 6, finaliza-se apresentando as conclusões do trabalho, abordando os resultados e as metodologias empregadas, abrindo espaço para a continuação e aperfeiçoamento desta linha de pesquisa.

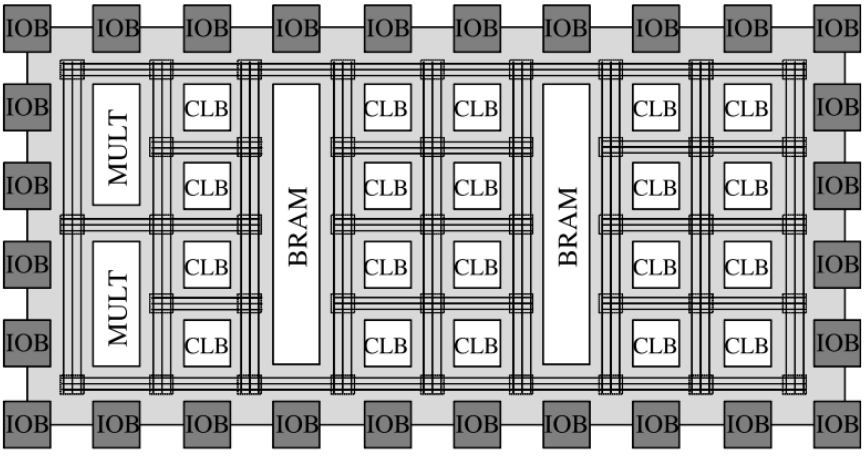
2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Neste capítulo se encontra toda a base de conhecimentos para o entendimento dos tópicos abordados nesse trabalho, como conceitos da área de sistemas embarcados e redes. Na seção 2.1 são apresentado dados sobre FPGAs, o que são, seus benefícios e motivação por trás do emprego deles na solução do problema. Na seção 2.2 são demonstrados conceitos de virtualização de funções de rede, o porquê desse paradigma ser interessante e de constante crescimento. Finalizando na seção 2.3 com os trabalhos relacionados, que serviram de base teórica para fundamentar e avaliar a solução deste trabalho.

2.1 Field Programmable Gate Array

Field Programmable Gate Array ou FPGA, de acordo com (HAUCK; DEHON, 2007) é um tipo de dispositivo semicondutor que se diferencia dos demais similares por ter a capacidade de ser programável para atender necessidades específicas dada a demanda do programador. Isto pode ser melhor compreendido sabendo que os FPGAs são organizados internamente como matrizes de recursos, onde seus componentes se encontram dispostos ao longo dos eixos como mostrado na Figura 2.1.

Figura 2.1: Arquitetura de uma FPGA simplificada. (NIEMIEC et al., 2020)



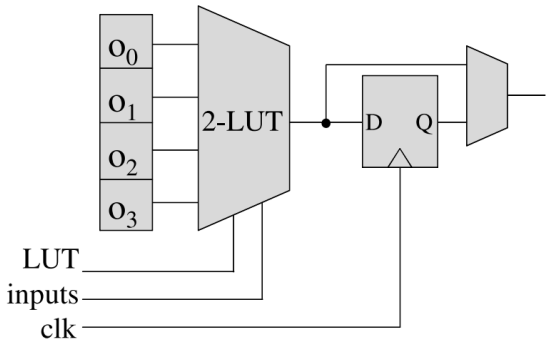
Os principais componentes de um FPGA moderno são:

- CLB: Configurable Logic Blocks (CLBs) compõem a maior parte do dispositivo. Componente lógico principal que é configurado quando feita a programação. Cada CLB é um próprio conjunto de outros componentes lógicos menores como visto na Figura 2.2, onde o número destes podem variar dependendo do modelo e fabricante

escolhido, sendo eles:

- LUT: Lookup Table é a base de um FPGA, pois é o componente físico que implementa portas lógicas através de pequenos arrays de memória que armazenam determinado valor, tendo seu comportamento esperado da mesma natureza que uma tabela verdade, podendo assim, fazendo a combinação de diversas LUTs, estruturar um circuito combinacional complexo. Outra característica das LUTs é que elas são mutáveis, assim sua lógica pode ser totalmente modificada, dando a versatilidade para o dispositivo.
- MUX: Multiplexador é o componente responsável por determinar qual valor lógico, dentre diversos valores de entrada, passará adiante no circuito, dado um valor seletor.
- FF: Flip Flop é um circuito eletrônico utilizado para armazenar o valor de um bit, sendo utilizado para dar continuidade na lógica do circuito dado por estados anteriores em pulsos de clock, permitindo que os dados avancem de forma ordenada e correta.

Figura 2.2: Estrutura base de uma CLB. (NIEMIEC et al., 2020)



- BRAM: Block Random Access Memory (BRAM), parte responsável por armazenar a maior quantidade de dados em memória dentro de um FPGA, sendo um tipo de memória de alta velocidade de acesso.
- IOB: Blocos de entrada e saída (IOB) de dados que permitem o fluxo dos pinos de IO com o resto dos circuitos lógicos e o mundo externo.
- DSP: Bloco responsável por implementar funções aritméticas, como multiplicações e multiplicações com acumulação. Quando disponíveis em grande quantidade no dispositivo, permitem alto paralelismo na execução dessas funções.

Essa diversidade de componentes reconfiguráveis justifica o uso destes dispositivos para virtualização, já que une basicamente o melhor de ambos os lados da genera-

lidade. Apresentando a baixa latência e alta capacidade de processamento comparável a Application-Specific Integrated Circuits (ASICs), ado que sua implementação é feita diretamente em hardware e não em software. Por outro lado demonstra flexibilidade comparável a dos General-Purpose Processors (GPPs), podendo ter sua implementação alterada para atacar os mais diversos problemas.

Justamente por serem um meio do caminho entre as soluções, os FPGAs vêm ganhando papel de destaque, pois além das características abordadas, eles são consideravelmente mais baratos, simples de colocar em operação e de se conseguir acesso que seus concorrentes até então, os ASICs, que são um tipo de hardware que possui desempenho formidável, porém são extremamente caros e demorados para se produzirem, dado que cada um é desenvolvido para resolver um problema específico. Olhando pelo outro lado, apesar de não ser tão flexível e versátil quanto um GPP na troca de funções, dada a implementação feita em hardware nos FPGAs, é possível o processamento de dados de maneira muito mais rápida e eficiente.

Para entender como funciona essa troca, que garante essa flexibilidade de operação para os FPGAs, se necessita voltar ao componentes elementares dele, as LUTs e os CLBs, onde o primeiro é um elemento lógico localizado dentro do segundo. O conceito por trás deles é simples, onde as LUTs são um tipo de memória que guarda determinado valor e passa para a saída dele dependendo o valor recebido como entrada, simulando uma tabela verdade. Os CLBs contém geralmente 4 LUTs, podendo conter de 4 a 6 entradas, assim podendo simular logicamente 4 portas lógicas de até 6 bits de entrada.

Desse modo, encadeando CLBs, é possível montar uma rede de portas lógicas que possui um comportamento de um circuito integrado complexo, já que as LUTs são um tipo de memória, com capacidade de armazenar valores de acordo com o comportamento desejado para elas. Estes são definidos a partir de código em HDL (Hardware Description Language), que consiste em linguagens com a função de descrever o comportamento do hardware. A partir delas, ferramentas de síntese e implementação de cada fabricante interpretam esse código e geram um arquivo de programação tradicionalmente chamado de bitstream. Este tipo de arquivo é interpretado pelo FPGA para que sejam feitas as alterações necessárias, tendo a capacidade de fazer a reconfiguração total ou parcial do dispositivo.

Através dessas ferramentas é possível manipular, respeitando determinadas regras, a combinação de circuitos para que se tenham combinações lógicas de acordo com a necessidade, podendo manipular subconjuntos dos recursos do FPGA. Pode ser citada aqui,

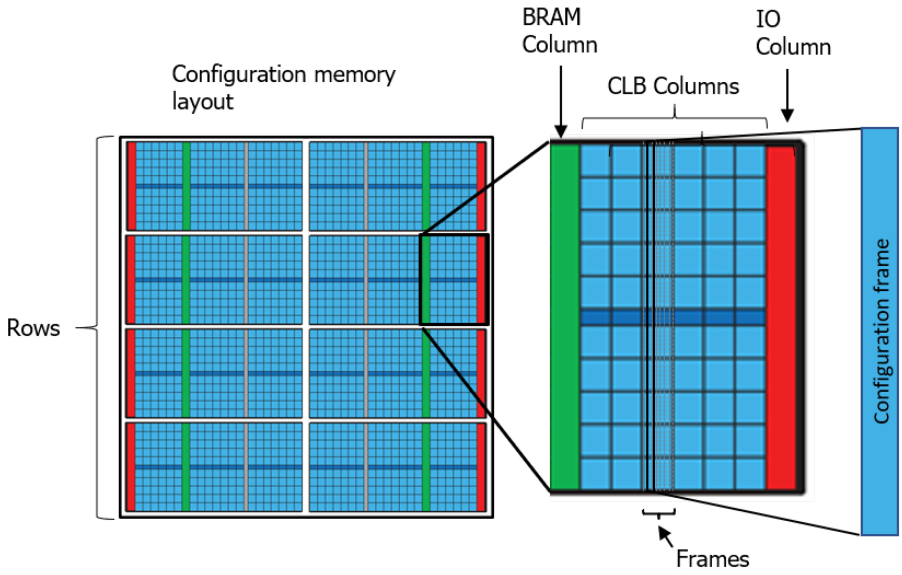
como exemplo, a troca de uma função de rede por outra, ou até a mesma função aumentando ou diminuindo seus recursos, consequentemente sua capacidade de processamento. Esta funcionalidade é o que garante a maleabilidade deste tipo de dispositivo, com a possibilidade de reprogramação, que é mais uma característica que o coloca novamente no meio entre os GPPs e os ASICs. Isso ocorre já que seu hardware é determinado para executar uma função específica, como os ASICs, mas podendo ser reprogramado, alterando seu funcionamento, da mesma maneira que os GPPs.

Um conceito importante que garante a flexibilidade dos FPGAs é o de partições. É a área de recursos delimitada pelas ferramentas de posicionamento e roteamento na hora de passar para o dispositivo o bitstream que vai configurar o hardware para que opere da maneira desejada. Essas áreas podem tomar praticamente todo o dispositivo, dependendo da quantidade de recursos necessários. Algumas regras precisam ser seguidas para a delimitação das partições. Além disso, tipicamente há uma área de configuração estática, responsável por interconectar as partições entre si ou com componentes externos. Neste trabalho foi mantido um valor de 10% dos recursos livres para esse propósito.

A criação de partições precisa respeitar uma altura mínima, consequentemente, uma quantidade mínima de recursos. Isto é um dos principais argumentos bases para este trabalho, dado que esse valor compreende em alocar sempre uma coluna de recursos. Para melhor compreensão, é válido ressaltar a organização dos recursos dentro do FPGA, que são dispostos de forma matricial. Porém existe uma subdivisão para estes recursos, dado por linhas de programação. Esta subdivisão agrupa um numero mínimo de recursos no eixo Y, e esse valor varia entre cada família de FPGA e entre cada fabricante, tornando-se um divisor para o numero total de recursos. A Figura 2.3 demonstra essa estrutura.

Servindo de exemplo, um dos modelos de FPGAs selecionados para este trabalho, o KU040, da família UltraScale da Xilinx, é constituido por 300 CLBs em cada coluna, porém de acordo com o fabricante, a altura da região de configuração compreende a quantidade de 60 CLBs, ou 12 BRAMs. Cada tipo de recurso possui um valor, logo o FPGA é dividido em 5 linhas, onde não é fisicamente possível alocar menos do que essa parcela da coluna de recursos, ou seja, uma alocação que requeira somente 5 CLBs é inválida, visto que ela precisa reservar um valor que seja múltiplo de 60, mesmo que não utilize todos. Vale frisar que esta regra de respeitar os limites da região de reconfiguração, com valores mínimo de recursos a serem reservados, vale somente para o eixo Y, dado que existem também estas mesmas divisões que cortam eixo X, porém não se aplica a mesma restrição. Outra restrição para a determinação das partições é respeitar que dado determi-

Figura 2.3: Representação de uma região de reconfiguração. (GIORDANO et al., 2019)



nado valor de CLBs para uma função, é reservado uma quantia 25% maior, dado que esse número extra vem por facilitar o roteamento e conexão dos elementos internos. Nota-se que esse valor extra é somente para CLBs, não se estendendo a outros tipos de recursos.

2.1.1 Reconfiguração Parcial

Entender como funcionam as partições, sabendo que podem existir diversas delas dentro de um único FPGA, é importante para que se entenda sobre a reconfiguração parcial ou total do FPGA. O conceito de reprogramar um FPGA é chave para a utilização dele como peça chave para virtualização, pois garante a dinamicidade do mundo de redes. Essa reprogramação pode ser total ou parcial, tudo depende da motivação por trás e da quantidade de recursos necessários e disponíveis para tal operação. Uma reconfiguração total ocorre quando as funções alocadas não possuem mais uso, ou existe uma urgência maior por outra função que necessite uma grande quantidade de recursos. Para isto o fluxo de execução do hardware em questão é paralisado, deixando-o inoperante pelo período de tempo em que é feita a reprogramação dele. A reconfiguração parcial, por outro lado, consiste em substituir a configuração de apenas uma partição, enquanto permite que as demais sigam operando normalmente, permitindo uma gerência melhor das outras atividades que estão em uso no dispositivo. As FPGAs escolhidas para este trabalho são da fabricante Xilinx, que utiliza o nome comercial da reconfiguração parcial de Dynamic Function eXchange(DFX).

2.2 Network Functions Virtualization

Funções de rede garantem o bom e seguro funcionamento de redes de computadores. Elas atuam no roteamento, comutação, balanceamento de cargas, firewall, criptografia, além de diversos outros exemplos. São funções que lidam com quantidades massivas de dados, operando preferencialmente com baixa latência, para que exista o menor atraso possível entre as pontas comunicantes. Para tal operação é exigido um paralelismo e uma lógica de processamento que atenda estes requisitos, sendo obtidos através de hardware construído para resolver especificamente cada um desses problemas.

Dado o grande crescimento do uso da Internet, a infraestrutura por trás, que suporta todo esse tráfego intenso de dados, precisou acompanhar esse movimento. Como o investimento nesta área demanda de muitos recursos, a busca por uma solução que se mostrasse mais efetiva do ponto de vista econômico é incessante.

O conceito de virtualização de hardware não é algo novo, mas que vem crescendo. Os equipamentos que antes eram necessários para executar as determinadas tarefas que foram projetados, vêm sendo substituídos por servidores com hardware de propósito geral, capazes de atender os mais diversos tipos de requisições, embora de maneira mais lenta, visto que os dados são manipulados via software e não mais diretamente por hardware. Essa troca permite uma versatilidade por parte do servidor, mas existe essa troca de resolver problemas muito específicos, aliada a uma alta eficiência de processamento, por poder atender mais casos de uso, possivelmente mais usuários, mas com uma menor eficiência no processamento dos resultados. Nesse contexto que entram os FPGAs, dada sua capacidade ser programado para tarefas específicas, entrando como acelerador de hardware, ainda permitindo a flexibilidade do servidor através de sua realocação dinâmica, mas melhorando muito o desempenho que seria obtido somente por GPP.

A ideia por trás de Network Functions Virtualization (NFV) em específico, como detalhado em (MIJUMBI et al., 2016), consiste em substituir hardware especializado proprietário que seria responsável por disponibilizar serviços de rede, como um firewall, ou um Deep Packet Inspection (DPI), por uma rede de servidores, implementados em hardware mais genérico, como processadores de propósito geral, GPUs, ou como neste trabalho, em FPGAs. Antes da virtualização, certas funções de redes eram executadas a parte em hardware próprio, chamados de *middlebox*, baseados em ASICs, que tendem a ser muito caros e de difícil acesso. Este tipo de equipamento ainda é utilizado para tal função, apesar de vir perdendo espaço de atuação, visto que com NFV, a facilidade

de ter ao hardware necessário de forma simples e rápida, com componentes Commercial Off-The-Shelf (COTS).

As vantagens dessa troca são evidenciadas especialmente devido a esta da facilidade de acesso ao equipamento para este tipo de implementação, que além de aliar a simplicidade de instalação, permitindo colocar em funcionamento de maneira mais simples e rápida, já que a complexidade do hardware em questão é mais baixa, permite uma alta escalabilidade, dado que se for necessário, por demanda, a capacidade desses servidores pode ser aumentada ou diminuída de maneira simples, não deixando hardware ocioso, assim garantindo uma melhor utilização dos recursos financeiros. Embora exista o lado negativo no quesito velocidade de processamento, já que de um lado temos uma solução rodando em um equipamento que foi desenvolvido para resolver tal problema, e de outro a mesma solução tendo que ser processada via software, ela ainda assim pode ser vantajosa, já que os custos de aquisição e manutenção das middleboxes são elevados. Isso possibilita que, com uma quantidade de recursos financeiros inferior, se adquira equipamento suficiente que comporte as mesmas demandas. Outra vantagem da facilidade de acesso a esse tipo de equipamento, é que permite comprar de quem oferecer melhores preços, independente da marca ou vendedor que seja, visto que existe pouca diferença prática entre as fabricantes. Alguns aspectos de segurança se tornam necessários no contexto de NFVs, em especial o monitoramento de rede, porém estes não são modelados neste trabalho.

2.3 Trabalhos Relacionados

A base teórica deste trabalho abrange diversas áreas, o que acaba por ter diversas inspirações de variados autores. Como por exemplo dentro do tópico de NFVs, como (DRAXLER; KARL, 2019), que traz uma visão completa e abrangente, visto que utiliza um sistema híbrido com CPUs e aceleradores de hardware, neste caso, faz uso de GPUs ao invés de FPGAs, além de levar em conta toda a parte de posicionamento, roteamento e escala do sistema, o que se mostra um modelo bastante completo, principalmente do ponto de vista teórico. Os autores tratam de um problema NP-hard, mas, por não lidarem com FPGAs, acabam por não lidar com o conceito de partição, logo não tem a preocupação de como lidar com os recursos disponíveis no sistema modelado.

Em uma outra visão similar, em (SHARMA et al., 2020), os autores focam em uma abordagem econômica. É feita uma comparação entre um modelo de Integer Linear Programming (ILP) e um modelo em que o sistema tem ciência da alocação dos acelera-

dores de hardware, otimizando a alocação dos recursos disponíveis. É feita uma análise que tem como alvo principal, mensurar o impacto no OPEX e CAPEX dessas abordagens e comparar com uma tradicional já implantada, onde o principal objetivo é comparar a alocação de funções de rede virtualizadas em redes que são dotadas de hardwares capazes de fazer um processamento de dados com maior eficiência.

Em (PAGANELLI et al., 2021), os autores trazem uma análise também com viés econômico, abordando modelos com diferentes composições de hardware e fazendo comparações entre eles. Os autores trazem os resultados obtidos com foco no bom uso do posicionamento para maximizar o lucro, diferente dos outros que buscam também o mesmo objetivo, porém só modelam recursos.

Trabalhos como (LI; QIAN, 2016), (LAGHRISSI; TALEB, 2019) e (NIEMIEC et al., 2020) trazem uma abordagem mais geral de NFV como um todo, atacando o assunto de maneira que introduz o conteúdo, mostrando diferentes detalhes do paradigma, detalhando funções de rede e tipos diferentes de hardwares que podem ser empregados para empregar a virtualização. O último artigo em especial, serviu de alicerce para a fundamentação teórica deste trabalho, pois aborda exatamente na viabilidade do uso de FPGAs como hardware COTS para implementação de NFVs, fazendo também uma análise de como isso impactaria o OPEX (Operational Expenditure) e CAPEX (Capital Expenditure).

Uma abordagem com direcionamento para utilização em redes móveis é feita por (CAO et al., 2017) e (FAIRSTEIN et al., 2020), que concordam que o uso de NFV é um passo essencial para o crescimento e barateamento desse tipo de redes, que apesar de não ser o foco desse trabalho, só sustenta a tese de que esse paradigma é algo que só tende a ser mais pesquisado e utilizado.

No que tange ao problema de posicionamento de VNFs, de acordo com (YI et al., 2018), por ser provado NP-Hard, se torna muito difícil de atingir um resultado ótimo, principalmente com tamanhos de redes muito grandes. Assim, uma solução muito comum para resolver esse problema acaba por ser a utilização de ILPs e Mixed ILP, abordagens mais complexas quando comparadas a outras heurísticas como a gulosa, que também é encontrada em diversas soluções.

Com essa base teórica, este trabalho se diferencia dos demais, pois, como os que consideram o posicionamento de funções de rede em sistemas heterogêneos, acabam por modelar FPGAs de maneira superficial e às vezes irreal, tendendo a alcançar até mesmo soluções não realizáveis. Desta maneira, este trabalho vem por agregar nesta linha de

pesquisa, mostrando um modelo mais fiel à realidade do hardware.

3 HEURÍSTICA DE POSICIONAMENTO DE VNFS EM INFRAESTRUTURAS COM FPGA

Neste capítulo é demonstrada a lógica e tomada de decisão por trás da solução do trabalho. Na seção 3.1, definimos e modelamos o problema objetivado neste trabalho. Na seção 3.2, apresentamos a heurística de alocação das funções proposta no trabalho, ilustrando como é aplicada.

3.1 Definição do problema

A modelagem do problema atacado neste trabalho tem início ao situar onde ele se localiza. Como se trata da virtualização de funções de rede, o primeiro passo é destacar que todo o problema se estrutura em um rede de nodos, simbolizada por um grafo $G_{net} = (N, L)$ conectado, não dirigido e ponderado. Cada nodo da rede $n \in N$ dispõe de uma quantidade limitada $r(n)$ de recursos computacionais. Como a abordagem neste trabalho foca na utilização de FPGAs, r é descrito como $r(n) = (CLB, BRAM, DSP)$. Estes recursos são os componentes internos do hardware em questão, onde os CLBs são compostos por uma determinada quantidade de Lookup Tables (LUTs), Flip-Flops (FFs) e multiplexadores. Os valores desses recursos variam de acordo com o FPGA escolhido. Cada link de rede $l \in L$ tem uma vazão de dados máxima representada por $t(l)$ e um determinado atraso dado por $lat(l)$.

A geração dos grafos utilizados nas simulações é feita utilizando a biblioteca $networkX$ em Python, detalhada em (HAGBERG; SCHULT; SWART, 2008). É estruturada para modelar e manipular mais facilmente grafos. Garante-se que o grafo seja conexo, podendo assim todo nodo ser acessível por pelo menos um caminho de qualquer nodo de partida. Foram utilizadas algumas topologias de redes conhecidas como inspiração inicial, como por exemplo a GEANT, rede com suporte a educação e pesquisa da comunidade europeia. Embora nesse trabalho, para que se tentasse encontrar resultados com um certo padrão, foi utilizado um gerador que produz um novo grafo para cada iteração de cada instância, possibilitando que seja checada a metodologia, independente de certas características da topologia escolhida.

Partindo deste modelo de infraestrutura, consideramos agora as diversas funções de rede que podem ser virtualizadas e executadas por esse sistema. O conjunto de funções

de rede é dado por $NF = f_0, f_1, f_2...f_{p-1}$, onde cada f é um tipo diferente de função, como firewall, Deep Packet Inspection (DPI), Network Address Translation (NAT), Domain Name Service (DNS), entre outros. Cada uma dessas funções pode ser implementada de maneiras diferentes, no sentido de existir abordagens que buscam mais performance, alocando mais recursos da rede, ou de maneira mais econômica, alocando menos recursos, buscando por atender requisições menores. Desse modo explicita-se que uma mesma função $f \in NF$ possui um conjunto de implementações $I(f) = i_0, i_1, i_2, ..., i_{q-1}$, onde i é uma forma de implementação, podendo assim, apresentar diversas maneiras de serem executadas, dependendo das características requeridas por quem realiza a requisição. Cada implementação i é definida por $i = (clb_i, bram_i, dsp_i, lat_i, t_i)$, onde clb_i é o número necessário de CLBs, $bram_i$ é o valor necessário de blocos de memória BRAM para a execução, dsp_i é o numero necessário de blocos de DSP, lat_i é a latência máxima dessa implementação e t_i é a vazão de dados mínima atingida pela execução desta implementação.

Tendo a topologia e estrutura modelada, assume-se que o modelo esteja pronto para prover serviços de rede, solicitados através de requisições, que devem conter informações básicas, como nodo inicial N_s, nodo final N_d, latência máxima aceita max_lat, taxa de vazão de rede mínima aceita min_t, quais funções de rede deseja-se implementar f, que implica em escolher o tipo de implementação i para cada f, dado por $R = (N_s, N_d, max\_lat, min\_t, f)$.É importante citar que neste trabalho não é considerado Service Function Chaining (SFC), definido por (HALPERN; PIGNATARO, 2015), dentro do contexto de NFVs. Aqui, cada requisição considera uma única função para alocação dentro da rede.

As regras de construção das partições são uma das bases deste trabalho, que tem por objetivo demonstrar que, para se implementar uma função, tende-se a alocar mais recursos do que somente os requisitados para o correto funcionamento da mesma. As partições tem, por recomendação da fabricante, formatos retangulares, como apresentado no Dynamic Function eXchange User Guide (XILINX, 2021). O formato retangular se dá por características físicas do equipamento.

O mínimo de recursos possível que pode ser alocado para montagem de uma partição é uma coluna de recursos, se tornando a unidade básica de alocação. Estas que por sua vez, são delimitadas pelas linhas de configuração ao longo do dispositivo, podendo apresentar valores diferentes para cada modelo de FPGA. Essa característica faz com que, para um melhor aproveitamento da área do dispositivo pela partição, ela tenda ser maior

no eixo Y do que no X, visto que isso garante a alocação de mais recursos de um único tipo. Isto se deve a distribuição dos mesmos, já que verticalmente as colunas são sempre do mesmo componente, enquanto que horizontalmente a distribuição não segue um padrão. Embora seja necessário levar em conta outros fatores para a determinação da partição, como considerar que provavelmente haverá outras a serem definidas. Isso mostra que encontrar uma solução de melhor posicionamento de partições é uma tarefa muito difícil, o que não entra no escopo deste trabalho.

Vale destacar que esta regra de limitação é válida para a altura, não vale o mesmo para a largura, essa tendo flexibilidade de poder ser delimitada da maneira que for necessário. Assim acaba que a quantidade de recursos dentro de uma partição será um múltiplo do numero de recursos presente nesta coluna. Isso implica em funções que necessitam de uma quantidade de recursos que não seja múltiplo do tamanho da coluna, acabam por alocar mais do que de fato precisaria para a execução.

Outra característica física do dispositivo que precisa ser respeitada na hora da definição da partição, é que essas colunas de recursos estão dispostas de forma não uniforme, não respeitando um padrão. Isso implica que se for preciso alocar uma quantidade maior de determinado recurso que não esteja de forma contígua, será necessário agrupar para esta partição recursos que não serão utilizados. Outras regras referentes a alocação de recursos têm ligação não mais com o processamento das funções, mas sim com a respeito da conexão e transmissão de dados interna e externa. Para evitar que ocorram problemas no roteamento, são alocados 25% a mais de CLBs do que o informado pela ferramenta de síntese, já que dependendo de alguns fatores, em caso de necessidade eles podem servir para auxiliar na passagem de sinais. Considerando a área total de recursos do dispositivo, é respeitado que pelo menos 10% da área do dispositivo seja reservada para a área estática, esta que é responsável pela comunicação das partições com os componentes externos, incluindo buffering dos dados de entrada e saída.

3.2 Heurística Gulosa

Para a alocação das funções dentro das partições já previamente posicionadas e dimensionadas, foi utilizada o algoritmo da heurística gulosa. Uma heurística simples que resolve os problemas de forma rápida, buscando o melhor resultado local, embora sem garantir um resultado ótimo resultado global. Como demonstrado no Algoritmo 1, recebendo como parâmetros de entrada a topologia do problema, que consiste na lista de

requisições, na lista de todos caminhos possíveis do grafo e a lista de nodos que contém todas as informações dos link e nodos, com seus respectivos recursos. Na sua saída esperado a lista de requisições alocadas dada a capacidade da topologia.

O algoritmo percorre a lista de requisições em ordem, para cada requisição é pego o nodo inicial e final com intuito de montar uma lista de caminhos, apenas com os que contém o nodo inicial como origem e o final como destino. Essa lista é ordenada a partir do número de nodos no caminho, do menor para o maior. Com esses dados, é percorrido em ordem essa nova lista, onde é checado nodo após nodo se em algum deles existe uma partição que comporte a requisição. Se em um nodo houver mais de uma capaz de alocar esta requisição, é feito um cálculo com pesos ponderados por tipo de recurso, onde é escolhida a que menos permanece com recursos excedentes quando subtraído os necessários para a alocação. Se dentro desse caminho é encontrado uma partição que atenda aos requisitos, então a taxa de transferência desse caminho, do início ao fim, é checada afim de garantir que atenda aos valores demandados. Se ambas capacidades de link e de recursos são validadas, essa requisição é considerada válida e então alocada na lista de requisições alocadas válidas.

Algorithm 1 Heurística gulosa para a abordagem ciente de partições

Input: A list of requisitions R, a list of links L, and a list of nodes N

Output: List of allocated requisitions A

```
1  A ← ∅ ;                                  // Start with an empty allocation
2  for each requisition r in R do
3      routes ← all paths from r.init_node to r.out_node
4      order routes by number of nodes
5      for each path p in routes do
6          node = False
7          link = False
8          for each node n in p do
9              if n.fpga == ∅ then
10                 continue
11             end
12             part ← smallestpartition in n.fpga with enough resources
13             if part! = ∅ then
14                 node = True
15                 break
16             end
17         end
18         if p has high enough throughput and low enough latency then
19             link = True
20             break
21         end
22     end
23     if link == True and node == True then
24         A ← r
25         update the throughput of the path in use
26     end
27 end
```

4 METODOLOGIA EXPERIMENTAL

Todas as simulações foram feitas utilizando a linguagem Python, escolhida por permitir desenvolvimento rápido e dispor de bibliotecas prontas que auxiliaram em alguns pontos da implementação, como a biblioteca $networkX$, que tem como objetivo facilitar a manipulação de estruturas que envolvam redes e grafos num geral. O programa gera uma topologia de rede para cada instância, onde o tamanho dela é dado pelo número de nodos da rede, sendo incrementado de 5 em 5 com limite em 40. Para cada um dos tamanhos são executadas diversas instâncias distintas, com topologias completamente novas, respeitando seu número de nodos, alcançando assim uma média desses valores, suavizando resultados excepcionais. Todos os dados gerados para a execução do programa são salvos em arquivos do tipo JSON e lidos na sequência, para que, caso seja de interesse, se possa utilizar outras topologias quaisquer, desde que respeitando a estrutura de dados empregada.

Quanto a escolha dos modelos dos FPGAs, selecionamos três, cada uma com quantidades de recursos diferentes, simbolizando uma pequena, a KU040, uma média, a KU095, e uma grande, a VU190. Os valores para comparação dos dispositivos escolhidos estão na Tabela 4.1, retirados de (XILINX, 2022). Dentro de cada nodo é sorteada aleatoriamente a existência de FPGAs e seus respectivos modelos, podendo conter nenhuma ou no máximo 3 FPGAs. Após o sorteio da quantidade de FPGAs, são sorteados o modelo de cada dispositivo.

Na Tabela 4.2 são apresentadas as funções de redes consideradas para as simulações, especificando a quantidade de recursos necessários para cada implementação. Nota-se que existem tipos de funções com quantidades diversas de número de recursos, considerando assim as diversas possibilidades que podem ser alcançadas nos resultados. Vale ressaltar que os valores que ali constam são os necessários para a implementação, o que não representa o tamanho final da partição em que essas funções podem ser alocadas, dado que elas precisam seguir regras de construção. Alguns valores de latência se encontram ausentes pois não foram especificados pelos autores. Para esses casos, consideramos que a latência da implementação é um número sorteado de maneira aleatória entre a faixa de valores das latências das funções de mesmo tipo.

Uma dificuldade encontrada na estruturação dessa solução é o posicionamento das partições dentro do FPGA, já que a resolução desse problema é algo complexo e que precisaria de uma heurística própria, fora do escopo deste trabalho, para achar uma solução

Tabela 4.1: Estatísticas do Desempenho do Produto

	KU040	**KU095**	**VU190**
Células Lógicas do Sistema	530.250	1.176.000	2.349.900
Flip-Flops do CLB	484.800	1.075.200	2.148.480
LUTs do CLB	242.400	537.600	1.074.240
RAM Distribuída Máxima (Mb)	7,0	4,7	14,5
Blocos de RAM	600	1.680	3.780
Bloco de RAM (Mb)	21,1	59,1	132,9
CMTs (1 MMCM, 2 PLLs)	10	16	30
DLLs de E/S	40	64	120
Máximo de I/Os HP	416	650	650
Máximo de I/Os HR	104	52	52
Slices DSP	1.920	768	1.800
Monitor do Sistema	1	1	3
PCIe Gen3 x8	3	4	6
Interlaken 150G	0	2	9
Ethernet 100G	0	2	9
Transceptores GTH 16.3Gb/s	20	32	60
Transceptores GTY 30.5Gb/s	0	0	60
PLLs Fracionados dos Transceptores	0	16	30

Tabela 4.2: Funções de Rede Selecionadas

Tipo Função	CLB	BRAM	DSP	Throughput (Gbps)	Latência (μ)	Referência
Firewall	1150	5	0	2.9 - 6.6	1.84 - 4.2	(AJAMI; DINH, 2011)
Firewall	8537	1	0	2	23 - 212	(LOPES; NAZAR; SCHAEFFER-FILHO, 2021)
Firewall	8123	241	0	92.16	73	(FIESSLER et al., 2017)
DPI	8377	37	0	0.8	278	(LOPES; NAZAR; SCHAEFFER-FILHO, 2021)
DPI	8612	438	0	0.8	2778	(LOPES; NAZAR; SCHAEFFER-FILHO, 2021)
DPI	15206	36	0	14.4	-	(YANG; JIANG; PRASANNA, 2008)
DPI	713	96	0	90	-	(JIANG; PRASANNA, 2009b)
DPI	5154	407	0	80	-	(JIANG; PRASANNA, 2009a)
DPI	6048	399	0	102.6	-	(FONG et al., 2012)
AES	2532	0	0	49.38	-	(SMEKAL; HAJNY; MARTINASEK, 2018)
AES	4095	0	0	59.3	2	(ZODPE; SAPKAL, 2020)
AES	2304	0	0	45	-	(SOLIMAN; ABOZAID, 2011)
AES	9561	450	0	119.3	-	(HENZEN; FICHTNER, 2010)
AES	486	0	0	3.44	-	(LIU; XU; YUAN, 2015)

boa. Para contornar esse problema foi determinada, de maneira empírica, a colocação de partições dentro de cada um dos modelos de FPGA. Somente esse posicionamento foi utilizado. As partições foram definidas em 3 tamanhos para que comportassem na média as diferentes funções de rede escolhidas, onde elam acabam por vezes serem levemente maiores dada sua geometria e a posição que se encontram. O FPGA de menor tamanho é o único que possui dois layouts de posicionamento visto que se for definido uma partição de tamanho grande não haveria espaço para outras, logo ele possui uma versão com somente uma partição e outra com mais, porém com tamanhos menores. Segue de exemplo na Figura 4.1, imagens retiradas do software Vivado da Xilinx, onde é possível visualizar o floorplan dos dispositivos, exemplificando a distribuição das partições escolhidas em

Tabela 4.3: Tamanhos de partições em cada modelo de FPGA

Model	Part	CLBs	BRAM	DSP
KU040	0	22,200	480	1,560
	0	10,800	300	600
	1	10,800	180	600
	2	2,958	40	240
KU095	0	19,200	480	192
	1	20,160	480	192
	2	10,440	288	144
	3	3,060	108	0
	4	3,060	144	72
	5	3,060	72	72
VU190	0	19,800	504	288
	1	19,080	576	288
	2	22,140	540	216
	3	19,440	540	216
	4	10,980	288	144
	5	10,800	360	144
	6	2,940	72	0
	7	2,940	72	0
	8	2,940	84	24

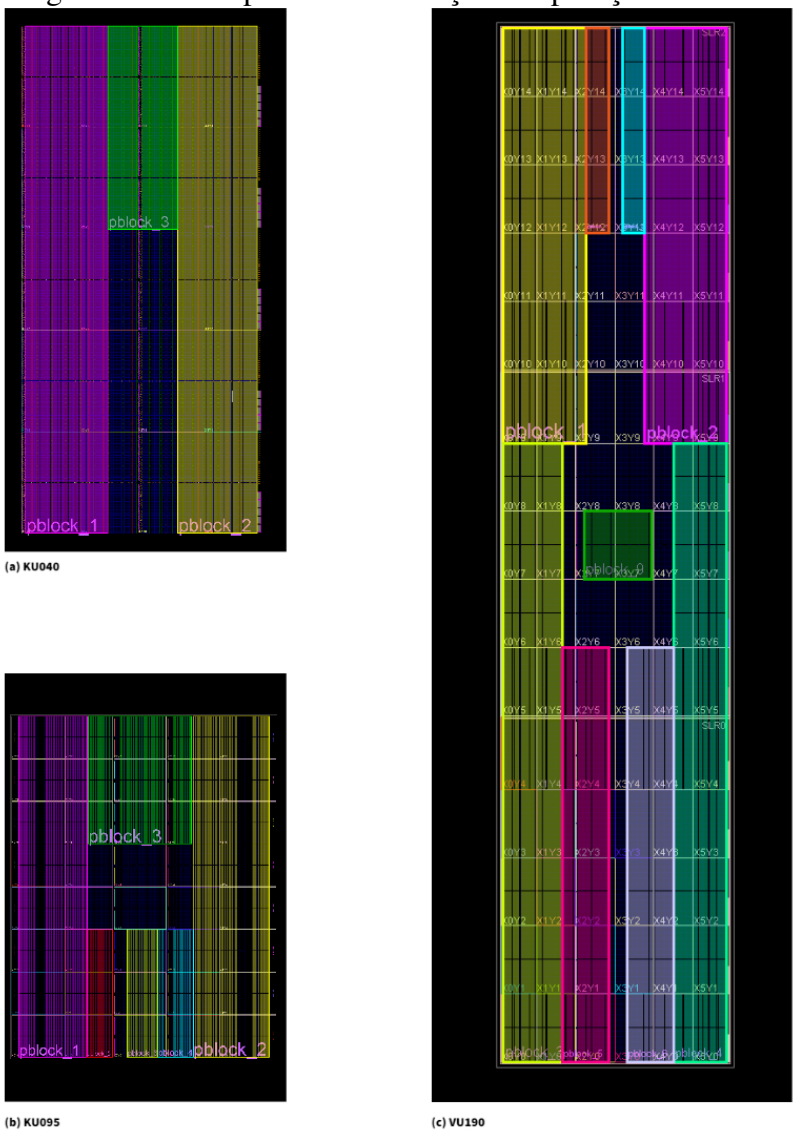
cada dispositivo, indicados pelos valores mínimos listados na tabela 4.3.

Para geração dos valores para criação das topologias, são utilizadas faixas de valores para que se obtenha uma certa variação. Por exemplo, para definição de latência, os valores variam de 20μs até 200μs, ou para vazão de dados que são sorteados dentre quatro valores base, 40$Gbps$, 100$Gbps$, 200$Gbps$ e 400$Gbps$.

Tendo modelado toda estrutura e lógica dessa solução, é feita então, dentro do mesmo programa e utilizando os mesmos dados, a execução do mesmo algoritmo guloso, porém para isto, ele não leva em consideração o conceito de partições, tendo uma liberdade que nem sempre será considerada válida, pois aloca apenas a quantidade de recursos necessários para a execução da função, desconsiderando toda a geometria e regras de construção, similar à abordagem apresentada em (SHARMA et al., 2020).

Este tipo de abordagem tende a ter resultados com quantidades maiores de funções alocadas, visto que por vezes consegue utilizar de recursos que seriam alocados em partições que não os utilizariam completamente. Para realizar a checagem desta abordagem e determinar quantas soluções são de fato válidas e respeitam todas as regras mencionadas para atenderem os requisitos de se adequar em uma partição, é executado um outro trecho de código que analisa todos estas informações e checa se encaixam-se nas regras. Este algoritmo de validação executa de forma otimista, ou seja, se este algoritmo determina

Figura 4.1: Exemplos de distribuições de partições de FPGAs.

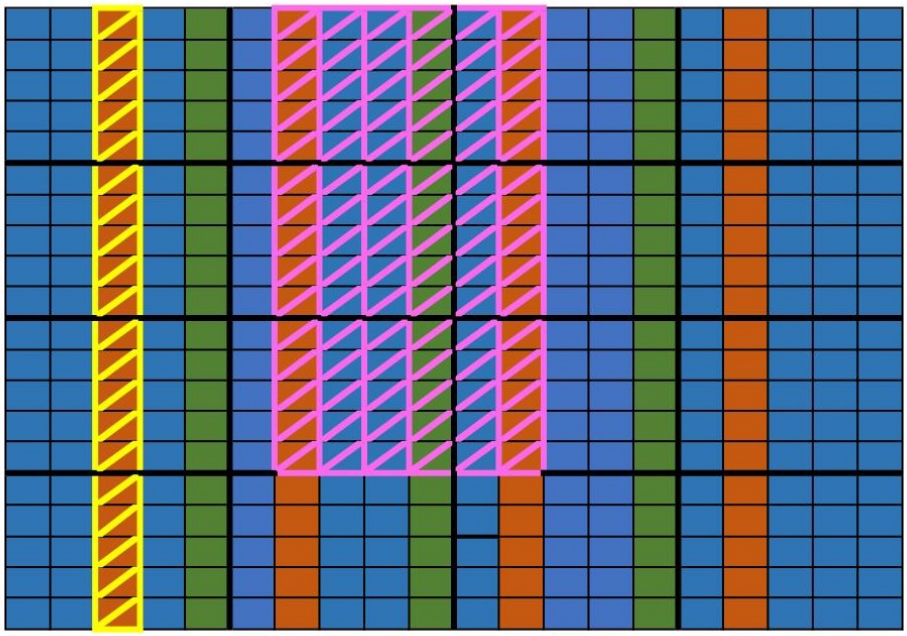


que uma alocação é inválida, ela certamente o é. Quando uma alocação é dita válida, ainda há a possibilidade de que ela seja inválida.

Para checar se a alocação de uma requisição é válida, é considerada a quantidade de recursos e a partir disso é gerada uma partição ótima para aquele determinado valor, tendo como o único fator limitante, os tamanhos absolutos do FPGA, no que diz respeito a distribuição dos recursos. Esse cálculo para a criação das partições tende a sempre priorizar partições maiores no eixo Y, devido aos componentes estarem dispostos em colunas, facilitando o agrupamento de maiores quantidades dos mesmos desta maneira, podendo ver isto na Figura 4.2 e explicado no Algoritmo 2. O algoritmo consiste em verificar todas as requisições recebidas pela execução da heurística gulosa, pela abordagem não ciente de partições. Essa verificação consiste em achar uma área mínima para esta parti-

ção que disponha da quantidade de recursos para ser uma alocação válida. Para isso são checadas em ordem todas as FPGAs dos caminhos possíveis dessa requisição. A partir da identificação do modelo de FPGA, se identifica a quantidade de linhas de configuração e disposição dos recursos nas colunas. Com a quantidade de linhas faz-se uma série de testes, indo do numero total de linhas, decrementando de uma unidade a cada teste, até 1, usando esse valor como denominador num cálculo análogo ao de área, em busca da menor razão de número de linhas por número de trechos de coluna. Tendo determinado essa área a partir da razão, calcula-se a quantidade de recursos dentro dela. Esses testes de encontrar a melhor razão são feitos pelo ponto de vista considerando CLBs e após BRAM, pois existe uma diferença de quantidades mínimas de recursos entre as colunas de cada tipo específico de recurso. Com isso se utiliza da menor partição em relação a área, mas que atendam a essas quantidades mínimas de recursos, e se a quantidade de recursos total dessa partição é menor que a quantidade disponível na FPGA, a requisição é considerada válida. Quando não há um FPGA capaz de receber a implementação requisitada, devido ao esgotamento de recursos, a alocação é considerada inválida.

Figura 4.2: Exemplos de alocações de partições.



A Figura 4.2 é um exemplo um pouco simplificado para a tomada de decisão da determinação do tamanho das partições. Em ambos os casos são criadas partições que buscam satisfazer a alocação, dada quantidade mínima necessária de recursos. Na partição amarela é exemplificado o caso em que se precisa alocar 4 trechos de coluna (linhas do

FPGA) de BRAM, representada pela coluna de cor vermelha. Para se atingir um tamanho ideal da partição, é preferível que se aloque o conjunto de colunas na vertical. Assim foi determinada a partição, dado que existia a capacidade de se alocar toda a quantidade em uma única coluna contígua de recursos, evitando alocação de recursos que não seriam utilizados. Na partição rosa demonstra-se o exemplo de alocação de uma partição quando a quantidade de recursos necessários excede o total disponível em uma coluna completa. Para achar uma solução então é feito um cálculo para encontrar a melhor proporção da partição. Assumindo que ela precise alocar todas as seis linhas de BRAM, a proporção engloba 3 trechos em duas colunas, em uma proporção 3 x 2, alocando menos recursos do que caso fosse optado por 2 x 3 colunas, minimizando assim a quantidade de outros recursos que estão dispostos entre as BRAM.

Apesar de uma solução ser válida e alcançar resultados com maior número de alocações do que utilizando a abordagem ciente de partições, não significa dizer que aquela seria a melhor maneira a se alocar aquelas partições visto que podem acabar sendo geradas soluções inválidas, ou seja, que não podem ser comportadas nos requisitos do FPGA, como veremos no Capítulo 5.

Algorithm 2 Algoritmo otimista para validação de alocação de funções em FPGA

Input: A list of requisitions R, a list of links L and a list of FPGA F

Output: A list of requisitions that were not allocated I

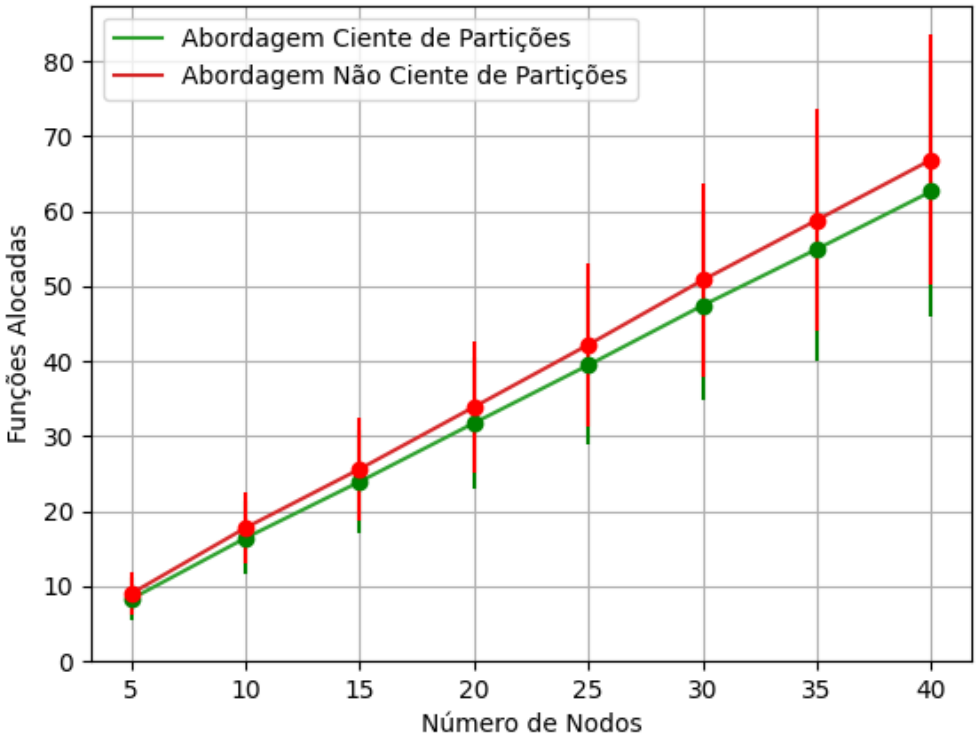
$I \leftarrow \emptyset$
for *each requisition* r *in* R **do**
    $routes \leftarrow$all paths from r.init_node to r.out_node
    order $routes$ by number of nodes
    partition_invalid = *True*
    $min\_Column\_CLB = \lceil r.CLB/60 \rceil$
    $min\_Column\_BRAM = \lceil r.BRAM/12 \rceil$
    for *each path* p *in* $routes$ **do**
        for *each node* n *in* p **do**
            for *each FPGA* f *in* F **do**
                check for the model of f to get *total_lines*
                for $line \leftarrow total\_lines$ *to* 1 *step* -1 **do**
                    if $min\_Column\_CLB\%line == 0$ **then**
                        $BRAM = \frac{min\_Column\_CLB}{min\_Size\_between\_resource}$
                    else
                        $ratio = \frac{min\_Column\_CLB\%line}{line}$
                        if $ratio[line] \leq min\_ratio$ **then**
                            min_ratio = ratio[line] `// Check for the smallest ratio`
                        $BRAM = \frac{\frac{min\_Column\_CLB}{line}}{min\_Size\_between\_resource} * ratio$
                for $line \leftarrow total\_lines$ *to* 1 *step* -1 **do**
                    if $min\_Column\_BRAM\%line == 0$ **then**
                        $CLB = \frac{min\_Column\_BRAM}{min\_Size\_between\_resource}$
                    else
                        $ratio = \frac{min\_Column\_BRAM\%line}{line}$
                        if $ratio[line] \leq min\_ratio$ **then**
                            min_ratio = ratio[line] `// Check for the smallest ratio`
                        $CLB = \frac{\frac{min\_Column\_BRAM}{line}}{min\_Size\_between\_resource} * ratio$
                if *f.CLB* - $min\_Column\_CLB < 0$ *or* $f.BRAM - min\_Column\_BRAM < 0$ **then**
                    continue `// Not enough resources in the FPGA`
                else
                    if $CLB \geq min\_Column\_CLB$ *or* $BRAM \geq min\_Column\_BRAM$ **then**
                        $partition\_invalid = False$
                        update resources values in respective FPGA
                        break `// Go to the next requisition`
    if $partition\_invalid == True$ **then**
        $I \leftarrow r$

5 RESULTADOS EXPERIMENTAIS

Para se atingir os resultados, dentro da execução do programa, cada tamanho de instância, ou seja, quantidade de nodos da topologia, foi repetida múltiplas vezes, extraindo os resultados e a partir deles foi feita uma média para atenuação de resultados excepcionais que poderiam vir a surgir devido a geração de valores aleatórios a cada repetição. Nos gráficos abaixo todos os resultados foram simulados com 1000 repetições por tamanho de instância.

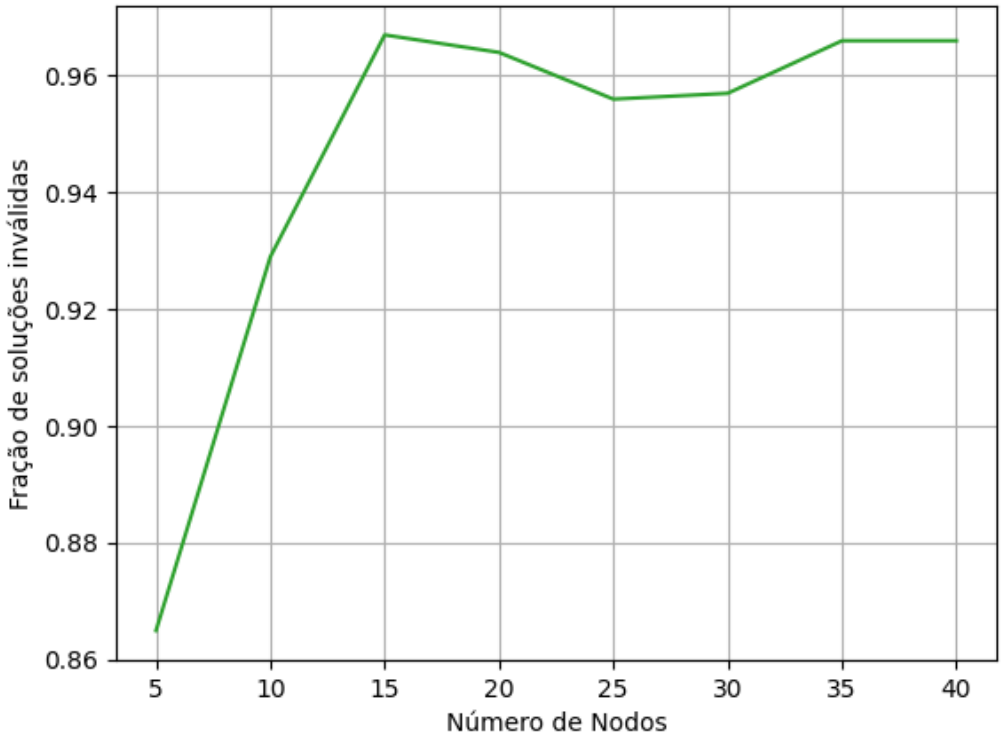
Os resultados demonstrados a seguir foram obtidos através da execução da heurística gulosa em ambas abordagens, para fazer o comparativo de eficiência entre ambas. Demonstrando primeiramente, que a abordagem ciente de partições acaba por alocar uma quantidade menor de funções que a não ciente de partições, como demonstrado na Figura 5.1, onde se nota que a solução deste trabalho aloca ligeiramente um número menor de requisições. Além disto, pode-se verificar que dado o aumento no número de nodos a diferença entre elas vai aumentando de forma proporcional. As linhas verticais no gráfico sinalizam o desvio padrão dos valores.

Figura 5.1: Diferença de alocações entre as abordagens.



Embora a versão não ciente de partições aloque mais funções, nem sempre essas alocações são válidas. Assim, um resultado relevante são os valores apresentados na Figura 5.2, que traz a razão das soluções inválidas pelo número total de soluções para a abordagem não-ciente de partições. Uma solução é considerada inválida quando existe pelo menos uma requisição que foi invalidamente alocada. Observa-se uma crescente de valores da razão em relação ao número de nodos da solução. Dado que topologias maiores chegam a percentuais acima dos 90%, ou seja, praticamente todas soluções simuladas possuem alguma alocação inválida. Isso mostra que as abordagens não cientes de partições, especialmente as de maiores tamanhos de topologias, tendem a serem inválidas do ponto de vista quantitativo de alocações de recursos.

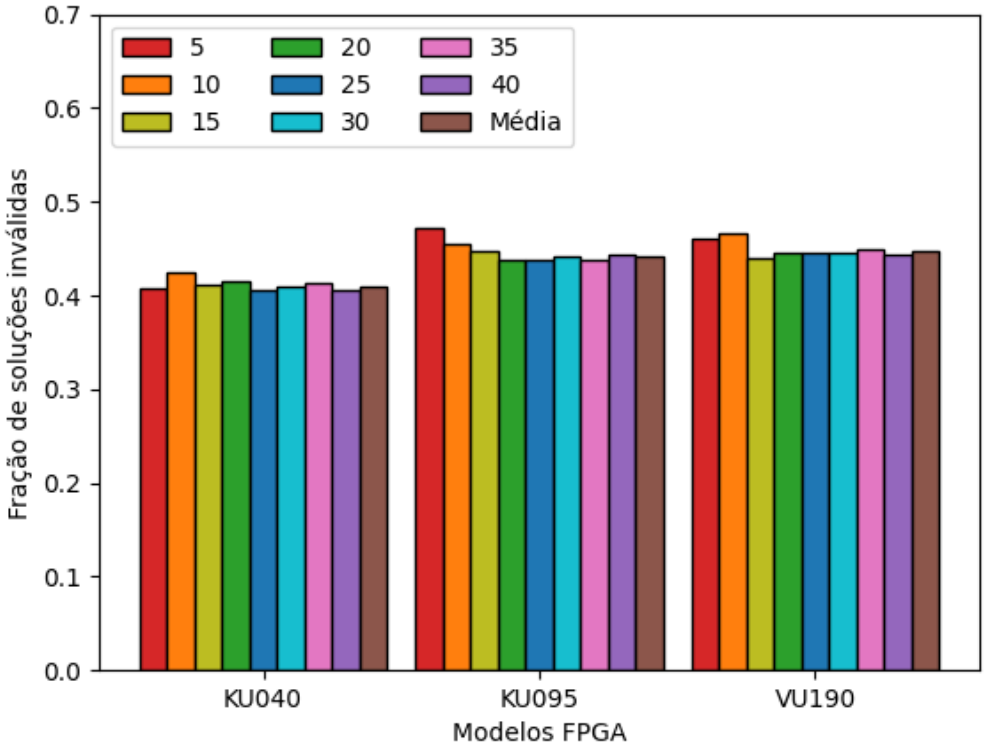
Figura 5.2: Fração de soluções inválidas.



Observando os dados da Figura 5.3, que apresenta a razão de alocação inválidas por FPGA, separando os resultados por modelo e por tamanho de topologia, nota-se que não há uma diferenciação discrepante em relação aos modelos. Realizando a execução de diversas vezes durante uma série de testes, se chega em um valor médio em torno de 45%. Isto implica que, fazendo aproximações, a cada 2 FPGAs numa solução, 1 deles apresenta alguma alocação inválida, corroborando com os valores do gráfico da figura 5.2, já que

a tendencia é que quanto menos dispositivos houver na topologia, maior a chance de ele possuir uma solução completamente válida.

Figura 5.3: Razão de alocações inválidas por modelo de FPGA.



A apresentação desses resultados reforça a teoria de que uma abordagem, na qual não se faz juízo quanto a partições e suas regras de criação, tende a ter um maior número de alocações, embora parte delas se mostrem inválidas, implicando que esta solução como um todo acabar por ser inválida. A partir disso se mostra que esse tipo de abordagem atinge resultados acima do real, levando a conclusões errôneas quanto a decisões a serem tomadas quanto a toda infraestrutura de rede.

6 CONCLUSÃO

Apresentados os resultados obtidos através de uma série de simulações, pode-se concluir que esses foram atingidos de maneira esperada. Demonstrando que a abordagem apresentada neste trabalho aloca uma quantidade menor de funções, dado que ela respeita o conceito de partições, que é uma premissa básica para qualquer implementação em FPGAs baseada em reconfiguração parcial. A quantidade a mais de alocações realizada por uma heurística que não observa esse conceito acaba por ser frequentemente inválida, quando consideramos as restrições físicas do dispositivo, já que ocupam recursos que não estariam disponíveis. Isso implica em tirar conclusões erradas quanto ao real desempenho de determinada infraestrutura de rede.

Esta diferença nos números, embora significativa, precisa ser considerada como uma base, uma referência para futuras avaliações e aprofundamentos de pesquisa, dado que existe toda uma série de variáveis que podem ser ajustadas para poder extrair um resultado mais fiel ao esperado em cada ambiente. Um dos aspectos é posicionamento e dimensionamento das partições dentro dos FPGAs, já que essa problemática envolve todo uma linha de pesquisa de otimização só para ela.

A sequência dessa linha de pesquisa em que este trabalho se baseia, busca por trazer luz neste tópico em específico, para que seja modelado de maneira cada vez mais fiel, já que a virtualização de funções de rede se mostra uma ótima opção para implementação de infraestruturas de rede. Outros trabalhos futuros incluem utilizar de heurísticas mais sofisticadas para a alocação de funções e também de posicionamento de partições dentro do FPGA, com isso otimizando os resultados encontrados. Outra possibilidade para linha de pesquisa é a utilização de dispositivos heterogêneos, como CPUs, GPUs e FPGAs, intercalados ou não, além da utilização de outras funções de rede, que podem ser encadeadas em cadeias de serviço, para uma maior possibilidade de variação dos resultados.

REFERÊNCIAS

AJAMI, R.; DINH, A. Embedded network firewall on fpga. In: **2011 Eighth International Conference on Information Technology: New Generations**. [S.l.: s.n.], 2011. p. 1041–1043.

CAO, J. et al. Vnf-fg design and vnf placement for 5g mobile networks. In: **Science China Information Sciences**. [S.l.]: April 2017, Vol. 60 040302, 2017.

DRAXLER, S.; KARL, H. Spring: Scaling, placement, and routing of heterogeneous services with flexible structures. In: **Network Softwarization (NetSoft)**. [S.l.]: 2019 IEEE Conference, 2019. p. 115–123.

FAIRSTEIN, Y. et al. Nfv placement in resource-scarce edge nodes. In: **Cloud and Internet Computing (CCGRID)**. [S.l.]: 2020 20th IEEE/ACM International Symposium on Cluster, 2020. p. 51–60.

FIESSLER, A. et al. Hypafilter+: Enhanced hybrid packet filtering using hardware assisted classification and header space analysis. **IEEE/ACM Transactions on Networking**, v. 25, n. 6, p. 3655–3669, 2017.

FONG, J. et al. Parasplit: A scalable architecture on fpga for terabit packet classification. In: **2012 IEEE 20th Annual Symposium on High-Performance Interconnects**. [S.l.: s.n.], 2012. p. 1–8.

GIORDANO, R. et al. Custom scrubbing for robust configuration hardening in xilinx fpgas. **Instruments**, v. 3, n. 4, 2019. ISSN 2410-390X. Available from Internet: <https://www.mdpi.com/2410-390X/3/4/56>.

HAGBERG, A. A.; SCHULT, D. A.; SWART, P. J. Exploring network structure, dynamics, and function using networkx. In: VAROQUAUX, G.; VAUGHT, T.; MILLMAN, J. (Ed.). **Proceedings of the 7th Python in Science Conference**. Pasadena, CA USA: [s.n.], 2008. p. 11 – 15.

HALPERN, J. M.; PIGNATARO, C. **Service Function Chaining (SFC) Architecture**. RFC Editor, 2015. RFC 7665. (Request for Comments, 7665). Available from Internet: <https://www.rfc-editor.org/info/rfc7665>.

HAUCK, S.; DEHON, A. **Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation**. [S.l.]: Morgan Kaufmann Publishers Inc., 2007. ISBN 9780080556017.

HENZEN, L.; FICHTNER, W. Fpga parallel-pipelined aes-gcm core for 100g ethernet applications. In: **2010 Proceedings of ESSCIRC**. [S.l.: s.n.], 2010. p. 202–205.

JIANG, W.; PRASANNA, V. Large-scale wire-speed packet classification on fpgas. In: . [S.l.: s.n.], 2009. p. 219–228.

JIANG, W.; PRASANNA, V. K. A fpga-based parallel architecture for scalable high-speed packet classification. In: **2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors**. [S.l.: s.n.], 2009. p. 24–31.

LAGHRISSI, A.; TALEB, T. A survey on the placement of virtual resources and virtual network functions. In: **Communications Surveys and Tutorials**. [S.l.]: IEEE, 2019. p. 1409–1434.

LI, X.; QIAN, C. A survey of network function placement. In: **Consumer Communications & Networking Conference (CCNC)**. [S.l.]: 2016 13th IEEE Annual, 2016.

LIU, Q.; XU, Z.; YUAN, Y. High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion. **IET Computers & Digital Techniques**, v. 9, n. 3, p. 175–184, 2015. Available from Internet: <https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cdt.2014.0101>.

LOPES, F. B.; NAZAR, G. L.; SCHAEFFER-FILHO, A. E. Vnfaccel: An fpga-based platform for modular vnf components acceleration. In: **2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)**. [S.l.: s.n.], 2021. p. 250–258.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. In: **Communications Surveys and Tutorials**. [S.l.]: IEEE, vol. 18, no. 1, 1st Quart., 2016., 2016. p. 236–262.

NIEMIEC, G. S. et al. A survey on fpga support for feasible execution of vnfs. In: **Communications Surveys & Tutorials**. [S.l.]: IEEE, VOL. 22, NO. 1, FIRST QUARTER 2020, 2020. p. 504–525.

PAGANELLI, F. et al. Profit-aware placement of multi-flavoured vnf chains. In: **Cloud Networking (CloudNet)**. [S.l.]: 2021 IEEE 10th International Conference, 2021. p. 48–55.

SHARMA, G. P. et al. Vnf-aapc: Accelerator-aware vnf placement and chaining. In: **Computer Networks**. [S.l.]: Elsevier, Computer Networks 177 (2020), 107329, 2020.

SMEKAL, D.; HAJNY, J.; MARTINASEK, Z. Comparative analysis of different implementations of encryption algorithms on fpga network cards. **IFAC-PapersOnLine**, v. 51, n. 6, p. 312–317, 2018. ISSN 2405-8963. 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S2405896318309182>.

SOLIMAN, M. I.; ABOZAID, G. Y. Fpga implementation and performance evaluation of a high throughput crypto coprocessor. **Journal of Parallel and Distributed Computing**, v. 71, n. 8, p. 1075–1084, 2011. ISSN 0743-7315. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0743731511000815>.

XILINX. **Vivado Design Suite User Guide: Dynamic Function eXchange (UG909)**. 2021. <https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration>. [Online; accessed March 27, 2023].

XILINX. **UltraScale Architecture and Product Data Sheet Overview**. 2022. <https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds890-ultrascale-overview.pdf>. [Online].

YANG, Y.-H. E.; JIANG, W.; PRASANNA, V. K. Compact architecture for high-throughput regular expression matching on fpga. In: . New York, NY, USA: Association for Computing Machinery, 2008. (ANCS '08), p. 30–39. ISBN 9781605583464. Available from Internet: <https://doi.org/10.1145/1477942.1477948>.

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>.

ZODPE, H.; SAPKAL, A. An efficient aes implementation using fpga with enhanced security features. **Journal of King Saud University - Engineering Sciences**, v. 32, n. 2, p. 115–122, 2020. ISSN 1018-3639. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S1018363918302071>.