Entrega 01 - MAC6931

Frederico Meletti Rappa - 15601846

1 Introdução

O presente relatório tem como objetivo detalhar os principais conceitos relacionados à execução de Graph Neural Networks em FPGAs e os desafios relacionados à sua aplicação em aceleradores tradicionais.

Serão discutidos a estrutura e o funcionamento das GNNs, suas principais aplicações e as dificuldades computacionais envolvidas, além de explorar como FPGAs podem ser uma alternativa para otimizar o desempenho desses modelos. Finalmente, como parte de um projeto em andamento, serão apresentados os principais desafios encontrados nas etapas iniciais do projeto.

2 Conceitos fundamentais

2.1 Graph Neural Networks

As Graph Neural Networks (GNNs), propostas em [4], definem um conjunto de modelos de redes neurais em que os dados têm estruturas de grafos. Diferentemente de redes neurais convolucionais (CNNs) e transformadores, que operam sobre imagens e textos respectivamente, as GNNs lidam com estruturas de conectividade complexas, o que introduz desafios específicos para sua computação eficiente.

Estas redes têm aplicações como redes sociais ou de pesquisa, química molecular, tráfego e visão computacional, e podem ser divididas entre aplicações *node-focused*, *edge-focused* e *graph-focused*. Tarefas em nível de nós incluem, por exemplo, a classificação de nós a partir de seus nós vizinhos, como a rotularização de um artigo a partir de os que o citam. Tarefas em nível de arestas incluem a predição de amizade entre usuários em redes sociais, por exemplo, enquanto tarefas em níveis de grafos incluem a classificação de grafos, como moléculas químicas ou sua segmentação.

Nesse sentido, a estrutura de um grafo tem desafios particulares: em primeiro lugar, a conectividade de um grafo pode ser reproduzida de maneiras distintas, o que significa que um mesmo grafo pode ter múltiplas matrizes de adjacências. Desse modo, a ordenação de seus elementos não é trivial, de modo que nós vizinhos podem não aproveitar de localidade espacial na memória. Além disso, diferentemente da estrutura fixa em grade de imagens, grafos possuem regiões com conectividades diferentes [7, 6] e arestas que podem representar informações relevantes. Isso significa que operações eficientes sobre regiões esparsas e densas podem requerir otimizações específicas e diferentes. Outro desafio neste contexto é o fato de quem comumente, nós de grafos não podem ser tratados como amostras independentes, uma vez que suas representações dependem das informações dos nós vizinhos que, como exposto, frequentemente não estão armazenados de forma contígua na memória. Finalmente, a depender do contexto, como redes sociais, o grande número de nós e arestas pode representar um problema crítico, ao introduzir overhead, além de dificultar a interpretação humana [5].

Assim como em redes neurais tradicionais, GNNs são definidas por meios de camadas de modelos com pesos a serem treinados. O mecanismo de *message passing* é amplamente utilizado por GNNs para agregar a um nó informações de seus vizinhos: a cada camada, são feitas operações de agregação e combinação de features [3]. A etapa de agregação produz uma mensagem de uma camada para um nó agregando features de nós vizinhos, como por soma, média ou máximo, por exemplo. A combinação atualiza o vetor de features do

nó com a mensagem produzida anteriormente. Desse modo, enquanto a etapa de agregação tem alto custo de memória por percorrer os vizinhos de cada nó - que podem não estar próximos na memória - a etapa de combinação tem alto custo computacional para atualizar o vetor de features dos nós. Estar operações são, portanto, computacionalmente intensivas no processamento de GNNs.

Da mesma forma que redes neurais tradicionais, há uma distinção entre camadas de redes neurais. Graph Convolutional Networks (GCN) se assemelham a redes neurais convencionais, uma vez que nós agreagam informações dos vizinhos a cada camada. Graph Attention Networks (GAT) estende o conceito de atenção em redes neurais a grafos: o mecanismo de atenção calcula a importância relativa de cada nó em relação a outros

A modelagem de GNNs pode ser feita, como grande parte das redes neurais, através de bibliotecas Python, sendo as principais *Deep Grapho Library* (DGL) e *Pytorch Geometric* (PyG), que extendem Pytorch e, portanto, tem uso similar a outras bibliotecas de aprondizado profundo na linguagem. Ambas as bibliotecas oferecem suporte a execução dos modelos de forma transparente em GPUs. Entretanto, a complexidade dessas classes de modelos de redes neurais exige aceleração eficiente. Métodos convencionais, como CPUs e GPUs, apresentam limitações em eficiência energética e não são otimizados para arquiteturas de modelos de redes neurais especííficas, o que pode resultar em desempenho subótimo. Além disso, aceleradores tradicionais não são capazes de lidar de forma eficiente com as irregularidades inerentes às GNNs pelo acesso irregular a memória e ao uso ineficiente dos recursos computacionais em GPUs [8].

2.2 Field Programmable Gate Arrays (FPGAs)

As Field Programmable Gate Arrays (FPGAs) são dispositivos que podem ser programados para implementar diversos tipos de circuitos ou sistemas digitais. Essa flexibilidade permite que sejam utilizados em aplicações variadas, desde prototipagem rápida até sistemas embarcados e aplicações específicas de hardware.

Uma FPGA é composta essencialmente pelos seguintes blocos:

- Blocos Lógicos Programáveis: responsáveis por implementar as funções lógicas.
- Interconexões Programáveis: permitem a comunicação entre os blocos lógicos.
- Blocos de Entrada e Saída (I/O): conectam os blocos lógicos ao ambiente externo.

Desse modo, a principal vantagem das FPGAs é sua flexibilidade, permitindo reconfiguração mesmo após a fabricação. Isso possibilita atualizações futuras e adaptações do hardware sem a necessidade de um novo ciclo de fabricação. Além disso, a flexibilidade permitem o projeto de hardware especializado mais simples. [1]. Estes dispositivos podem ser programados de acordo com o uso pretendido, seja por meio de linguagens de descrição de hardware como VHDL e Verilog, ou linguagens de alto nivel com High Level Synthesis (HLS).

No contexto de redes neurais, a flexibilidade de FPGAs permite o desenvolvimento de arquiteturas otimizadas para modelos específicos, distribuindo cargas de trabalho críticas e reduzindo gargalos associados ao acesso irregular à memória. Diferentemente de CPUs e GPUs que têm arquiteturas fixas, FPGAs podem ser projetadas para maximizar a eficiência da agregação e combinação de features. Além disso, a possibilidade de pipeline e paralelismo em nível de hardware torna essas plataformas atrativas para tarefas de inferência em tempo real. Além disso, CPUs e GPUs, apresentam limitações em eficiência energética e não são capazes de lidar de forma eficiente com as irregularidades inerentes às GNNs pelo acesso irregular a memória e ao uso ineficiente dos recursos computacionais em GPUs.

3 Bibliografia Consultada

Conforme mencionado, o uso de FPGAs para a aceleração de redes neurais tem sido explorado em devido à sua flexibilidade e eficiência energética. No entanto, a implementação eficiente desses modelos em hardware

ainda enfrenta desafios, especialmente no caso de GNNs, devido à sua natureza irregular de acesso à memória e processamento.

Como exposto na proposta, nesta seção, será apresentada uma breve revisão da literatura consultada nas etapas iniciais do projeto sobre a aplicação de FPGAs em modelos de CNNs e GNNs, destacando as principais abordagens, desafios encontrados e soluções propostas. A seguir, são apresentados alguns trabalhos relevantes encontrados na literatura para a aceleração de GNNs em FPGAs.

3.1 GraphAGILE: An FPGA-based overlay accelerator for low-latency GNN inference

O artigo [8], publicado na IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, descreve uma arquitetura de hardware e compilador teóricos para aceleração de inferência em FPGAs, e serviu como base para os estudos do tema subsequentes no projeto. O artigo propõe GraphAGILE, um *overlay accelerator* composto por que combina (1) um compilador que gera em uma sequência otimizada de instruções a partir da estrutura do grafo e modelo para execução na FPGA; e (2) uma arquitetura de FPGA responsável por executar operações fundamentais de GNNs que otimiza a comunicação entre camadas do modelo. Aceleradores de overlay consistem em uma ISA e compilador desenvolvidos para um domínio específico. Proposto para a FPGA Xilinx Alveo U250, o GraphAGILE suporta modelos comuns de GNNs como *GCN*, *GAT*, *GraphSAGE*. A Figura 1 mostra a proposta de estrutura do acelerador e compilador. O hardware é composto pela FPGA em que o acelerador é executado, memória DDR para armazenamento do grafo, modelo e binários gerados pelo compilador, e um processador responsável pela execução do compulador. O compilador gera uma IR a partir do modelo desenvolvido em Pytorch Geometric, que é otimizada e transformada em um arquivo binário para execução.

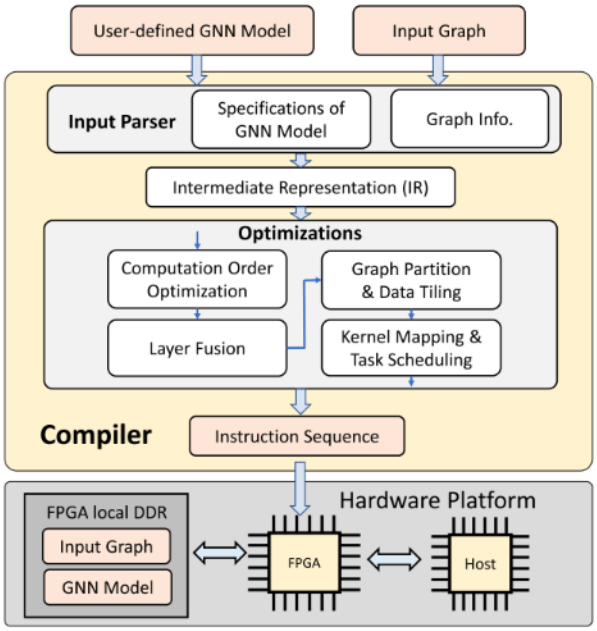


Figure 1: Estrutura de GraphAGILE, retirada de [8]

Conforme mencionado, diferentemente do caráter estruturado de CNNs, grafos têm regiões de diferentes níveis de esparsidade, de modo que implementações do estado-da-arte em CPU e GPU falham em lidar com operações de multiplicações de matrizes densas e esparsas ao mesmo tempo. Além disso, a falta de localidade espacial também torna o useo de aceleradores tradicionais subótimo. Finalmente, ao definir uma ISA gerada

a partir da estrutura do grafo e modelo e uma arquitetura unificada, o artigo propõe eliminar o overhead do desenvolvimento de hardware próprio para modelos e sua reconfiguração.

A arquitetura proposta para a FPGA, disposta na Figura 2 tem como estrutura principal o *Adaptive Computation Kernel* (ACK), unidade de processamento de tamanho configurável para execução de operações como multiplicação de matrizes gerais (GEMM), multiplicação de matrizes esparsas-densas (SpDMM) e multiplicação de matrizes densas amostradas (SDDMM), utilizadas em processamentos em regiões de diferentes esparsidades no grafo. Cada ACK faz parte de um *Processing Element* (PE), de modo que o número de PEs é também parametrizado. PEs operam em paralelo geridas por um scheduler que designa operações a partir das instruções geradas pelo compilador.

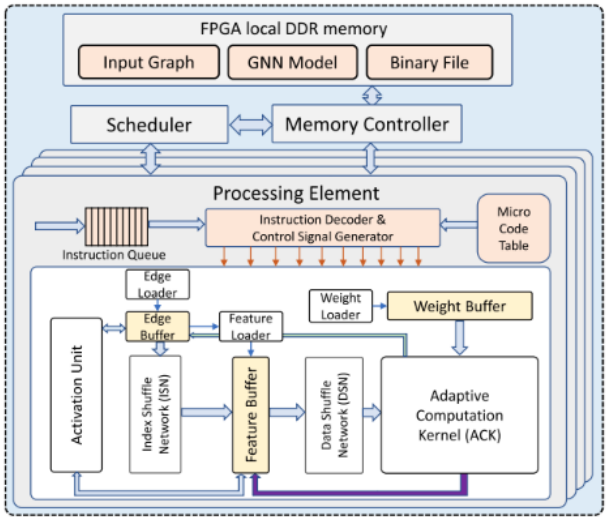


Figure 2: Arquitetura interna da FPGA, retirada de [8]

Conforme mencionado, o compilador proposto gera um conjunto de instruções a partir da definição de alto nível do modelo em Python e do grafo de entrada, e tem como principais operações

1. Particionamento do grafo: O grafo de entrada é dividido em blocos menores para otimizar o acesso à memória e paralelismo de execução.

2. Reordenação da ordem de computação: A sequência de operações é reorganizada para reduzir o número de acessos desnecessários à memória.

3. Fusão de operações: Operações consecutivas são combinadas para minimizar movimentação de dados entre registradores e memória externa.

4. Mapeamento dinâmico de kernels: O compilador escolhe automaticamente as unidades computacionais do ACK mais adequadas para cada operação.

Após a fase de otimização, o compilador geraria código binário que pode ser carregado diretamente no FPGA sem necessidade de reconfiguração de hardware.

O artigo menciona a execução de GrapgAGILE em 8 modelos aplicados sobre 7 datasets comumente utilizados em pesquisas de GNN, e compara os resultados com execuções em CPU AMD Ryzen 3990x, CPU-GPU (AMD Ryzen 3990x + Nvidia RTX3090) e outras abordagens em FPGA. Foi apresentado, portanto, redução de até 47,1x na latência de inferência em relação a implementações em CPU, 3,9x na latência de inferência em relação a CPU-GPU e 2,9x se comparado com os demais aceleradores FPGA.

Conforme mencionado, o acelerador foi base inicial para os estudos de aceleração de GNNs em FPGAs. O artigo descreve que a estrutura de hardware foi desenvolvida em Verilog e apresenta uma estrutura básica da

ALU. Desse modo, tentativas iniciais de execução basearam-se em tentar reproduzir a arquitetura proposta em uma nova FPGA. Entretanto, o artigo carecia de detalhes essenciais para tal. Além disso, a carência de detalhes da estrutura do compilador e o funcionamento a partir do modelo em Python tornou difícil sua reprodução e, apesar de exposta no arquivo a IR gerada, não foi definido exatamente como o scheduler a usa para gerir as PE. Finalmente, após contato com pesquisadores responsáveis pelo artigo, foi exposto que não houve efetivamente implementações em FPGA: execuções em CPU e GPU foram efetivamente feitas, mas os resultados obtidos se basearam em um simulador que calculava o tamanho do binário em bytes e clock cycles necessários para executar os modelos propostos na FPGA real. Desse modo, não havia efetivamente uma compilação e execução do modelo, mas simulações de seu comportamento. Os estudos do artigo, portanto, foram essenciais para entendimento dos desafios e complexidade do tema.

3.2 BoostGCN: A Framework for Optimizing GCN Inference on FPGA

O artigo [6] propõe um framework para aceleração de *Graph Convolutional Networks* (GCNs) em FPGAs. Como em GNN convencionais, a inferência de GCNs apresenta desafios em especial pelo acesso irregular à memória e cargas computacionais desbalanceadas pela distribuição irregular de graus dos nós em grafos.

BoostGCN propõe um método para acelerar em especial a etapa de Agregação de GNNs definida em 2.1 através de *Partition-Centric Feature Aggregation (PCFA)*, em que o particionamento do grafo utiliza informações de vértices, arestas e features de vértices de modo que em vez de processar os nós do grafo um por um, o PCFA opera sobre blocos de partições. Cada partição contém um subconjunto do grafo, e a computação é feita em paralelo dentro de cada partição nos *Feature Aggregation Modules* (FAM), de modo a melhorar a reutilização de dados on-chip e reduz significativamente os acessos irregulares à memória. Além disso, a fim de lidar com a disparidade de densidades de features em regiões do grafo, BoostGCN define módulos no hardware de combinação *Feature Update Modules* (FUM) diferentes para regiões densas (Dense-FUM) e esparsas (Sparse-FUM), de modo que é selecionada a variaedade de FUM de acordo com a região do grafo. BoostGCN também define uma arquitetura de acelerador que permite a execução das etapas de agregação e combinação em pipeline. Para equilibrar a carga de trabalho entre os FAM e FUM e minimizar o tempo de inatividade no pipeline, o BoostGCN implementa estratégias de scheduling como a agregação de uma camada e a atualização da camada seguinte simultâneas, de modo a otimizar a utilização dos recursos de hardware e aumentar o throughput da inferência, conforme apresentado em 3.

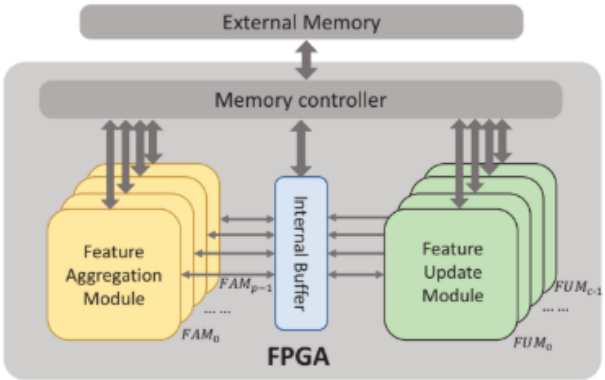


Figure 3: Arquitetura de BoostGCN, retirada de [6]

O funcionamento completo de BoostGCN é feito de acordo com a Figura 4: o framework recebe como entrada um modelo de GCN e um grafo a ser processado e, a partir desses dados, ele extrai informações relevantes para configurar o software e o hardware. Em software, BoostGCN decide o particionamento do grafo a partir de sua estrutura e das características da memória no hardware, além da ordem de execução na

pipeline. Em hardware, o framework decide o tipo de FUM a partir da esparsidade da matriz de features, define os parâmetros de hardware a partir de características da placa e, finalmente, gera o hardware para execução doo código gerado.

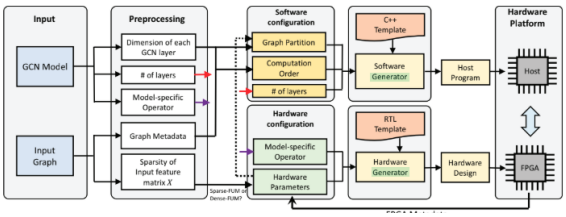


Figure 4: Arquitetura de BoostGCN, retirada de [6]

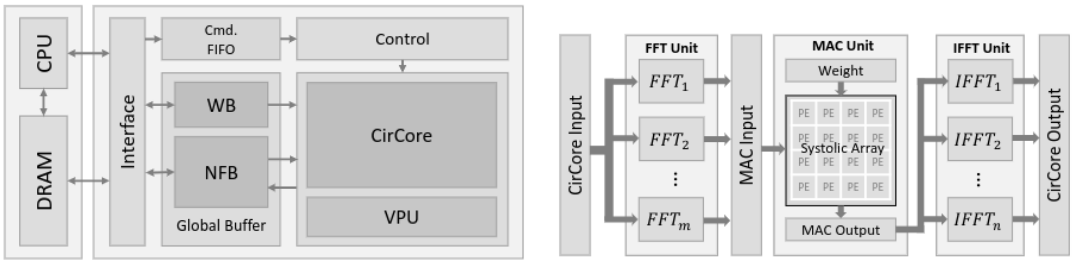
Os experimentos avaliam o desempenho do framework em 4 modelos e datasets a partir da comparação com a execução em CPUs e GPUs a partir das bibliotecas PuG e DGL mencionadas em 2.1, além de outros aceleradores baseados em FPGAs. Os aceleradores gerados pelo BoostGCN demonstraram speedups de cerca de 100x em relação a CPU, 30x em relação a GPU e entre 3 e 45x em relação aos demais em FPGA.

Em termos dos estudos de FPGAs para inferência em GNN, foram feitas tentativas de reproduzir de forma simples a geração de hardware feita por BoostGCN a partir de um modelo e grafo pequeno de GNN. Entretanto, a geração de estruturas equivalentes às FAM e FUM e pipeline de operações mostrou-se complexa, dado que não há descrições de baixo nível no funcionamento de hardware e detalhes dos modelos de GCN utilizados. Além disso, alguns dos modelos e datasets, mesmo simplificados, utilizados mostraram-se extremamente computacionalmente custosos para a CPU e GPU disponíveis, o que também dificultou o prosseguimento dos experimentos.

3.3 BlockGNN: Towards efficient GNN acceleration using block-circulant weight matrices

O artigo [9] propõe uma abordagem de co-design entre software e hardware para acelerar Graph Neural Networks. Diferentemente das abordagens citadas anteriormente, o artigo destaca como principal desafio na aceleração de GNNs a dimensão e complexidade de modelos. Essa complexidade torna a inferência em tempo real desafiadora, especialmente em plataformas de computação com recursos limitados. Para enfrentar esse desafio, os autores introduzem o BlockGNN, que utiliza matrizes de pesos bloco-circulantes para reduzir significativamente a complexidade computacional dos modelos GNN. A técnica já é utilizada na computação de redes neurais profundas, mas não é comumente utilizada em GNNs. Uma matriz bloco-circulante é uma matriz particionada em blocos, onde cada linha de blocos é uma rotação cíclica da linha anterior. Esse uso, em conjunto com transformada rápida de Fourier (FFT), permite a redução de complexidade da inferência. Os autores também propõem uma arquitetura para computação eficiente da inferência *CirCore*, que computa de forma paralela as operações de agregação e combinação, exposto nas Figuras 5.

O CirCore é um núcleo especializado na computação de matrizes bloco-circulantes e opera em três estágios: primeiramente, uma unidade de FFT transforma os vetores de entrada para o domínio espectral; em seguida, uma matriz sistólica realiza as operações de multiplicação e acumulação; por fim, uma unidade de IFFT converte os resultados de volta para o domínio espacial. Para otimizar o desempenho, a FFT e a IFFT são paralelizadas em múltiplos canais, e a matriz sistólica permite a execução eficiente das multiplicações espectrais. A Vector Processing Unit (VPU) é responsável por funções não lineares, como ReLU, exponenciais e sigmoide, além de operações vetoriais adicionais, funcionando como uma unidade SIMD com múltiplos canais de processamento paralelo. O Global Buffer, por sua vez, é dividido em um buffer de pesos e features dos nós, utilizando uma abordagem de prefetching para otimizar a utilização da largura de banda da memória. Diferentemente de arquiteturas que dependem de caches e memória embutida para otimizar o acesso aos



(a) Arquitetura de BlockGNN (b) Arquitetura CirCore

Figure 5: Módulos arquiteturais na FPGA extraídos de [9]

dados, o BlockGNN prioriza o alto desempenho computacional, garantindo que a limitação principal não seja o acesso à memória, mas sim a capacidade de processamento dos cálculos em larga escala.

As análises experimentais foram feitas em quatro modelos de GNN e quatro datasets. Em termos de desempenho, o BlockGNN obteve um speedup médio de 2.3x em relação ao CPU e 4.2x em relação ao HyGCN, referência em aceleração de GCN em arquitetura híbrida. Quanto à eficiência energética, o BlockGNN reduziu o consumo de energia em até 111.9x comparado ao CPU, com uma média de 68.9x. O consumo estimado do CPU foi de 125W, enquanto o BlockGNN operou com apenas 4.6W, tornando-o uma opção viável para aplicações em dispositivos de borda com restrições energéticas.

4 Experimentos iniciais e desafios enfrentados

Conforme mencionado, os artigos citados correspondem a um subconjunto de artigos utilizados nos estudos iniciais de inferência de GNNs em FPGAs. Inicialmente, foram feitos experimentos simples com as bibliotecas PyG e DGL a fim de visualizar os conceitos estudados de forma prática: para tal, foram usados os datasets descritos em [2], que representam dados de redes sociais, compras em e-commerce, citações acadêmicas e biologia molecular, alguns dos quais também foram usados nos experimentos dos artigos descritos. Devido ao suporte nativo e transparente a execução em CPUs e GPUs, os modelos puderam ser executados corretamente em ambas as plataformas com resultados similares nos datasets menores, e apresentaram problemas de performance em volumes maiores de dados.

Experimentos subsequentes basearam-se de forma mais próxima nas arquiteturas descritas nos artigos. Uma vez que inicialmente, o acesso remoto ao cluster de FPGAs A-Machine não estava disponível, tentativas iniciais se basearam em simulação do hardware via Xilinx Vivado, software de projeto de FPGAs. Os experimentos consistiram principalmente em tentativas de reprodução dos hardwares em uma FPGA simulada através de prototipação com VHDL. Inicialmente, portanto, foram simulados hardwares simples de multiplicação sistólica de matrizes. Em sequência, tentativas se basearam na reprodução dos resultados dos artigos, que representou desafios: em primeiro lugar, mesmo que muitas vezes utilizando os mesmos datasets, a heterogeneidade das arquiteturas mostrou-se desafiadora para reproduzir. Além disso, raramente artigos tinham descrições detalhadas de como a arquitetura foi projetada e detalhes de seu projeto, o que dificultava a reprodução. Não foram encontrados exemplos de código aberto e, no caso de GraphAGILE, o projeto se resumia a simuladores em vez de experimentos em hardware como sugerido. Com o acesso remoto ao hardware, outros desafios surgiram: incompatibilidades de versionamento de software impediram a impressão dos hardwares simulados nas placas reais.

5 Conclusão e próximos passos

Este relatório apresentou os principais conceitos de GNNs e FPGAs, além de alguns artigos que compuseram as etapas iniciais da pesquisa e como influenciaram os primeiros experimentos na área. Nota-se a variedade de abordagens para aceleração de GNNs e, em especial, a complexidade das arquiteturas propostas. Como mencionado, a aplicação prática dos estudos mostrou-se particularmente desafiadora, seja por características dos modelos, dados ou plataformas de execução.

Desse modo, tentativas subsequentes se basearam em especial em ferramentas de geração de hardware a partir de modelos em linguagens de alto nível, e serão abordados na Entrega 02.

References

[1] Umer Farooq, Zied Marrakchi, Habib Mehrez, Umer Farooq, Zied Marrakchi, and Habib Mehrez. Fpga architectures: An overview. *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*, pages 7–48, 2012.

[2] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[3] Yingjie Qi, Jianlei Yang, Ao Zhou, Tong Qiao, and Chunming Hu. Architectural implications of gnn aggregation programming abstractions. *IEEE Computer Architecture Letters*, 23(1):125–128, 2023.

[4] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[5] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[6] Bingyi Zhang, Rajgopal Kannan, and Viktor Prasanna. Boostgcn: A framework for optimizing gcn inference on fpga. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 29–39. IEEE, 2021.

[7] Bingyi Zhang, Hanqing Zeng, and Viktor Prasanna. Hardware acceleration of large scale gcn inference. In *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 61–68. IEEE, 2020.

[8] Bingyi Zhang, Hanqing Zeng, and Viktor K Prasanna. Graphagile: An fpga-based overlay accelerator for low-latency gnn inference. *IEEE Transactions on Parallel and Distributed Systems*, 34(9):2580–2597, 2023.

[9] Zhe Zhou, Bizhao Shi, Zhe Zhang, Yijin Guan, Guangyu Sun, and Guojie Luo. Blockgnn: Towards efficient gnn acceleration using block-circulant weight matrices. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1009–1014. IEEE, 2021.