Entrega 02 - MAC6931

Frederico Meletti Rappa - 15601846

1 Introdução

A aceleração de Graph Neural Networks é uma tarefa desafiadora devido a irregularidade dos modelos e complexidade do seu processamento. No relatório anterior, foram expostas tentativas de sua execução em FPGAs a partir de descrições de hardware expostas em artigos, que não tiveram o sucesso esperado. Este relatório descreve as tentativs subsequentes de execução destes modelos no hardware, a partir de ferramentas de High Level Synthesis.

Graph Neural Networks, como outros tipos de redes neurais, são comumente desenvolvidos em Python por meio de bibliotecas que permitem flexibilidade e interpretabilidade de linguagens interpretadas, mas que utilizam código compilado otimizado para melhor performance. No contexto de GNNs, se destacam Pytorch Geometric (PyG) e Deep Graph Library (DGL), ambas baseadas em Pytorch. As ferramentas permitem a descrição de camadas otimizadas específicas de GNNs e têm suporte nativo à execução em GPUs. Desse modo, são frequentemente utilizadas no desenvolvimento dos modelos e como comparativo em projetos de aceleradores. Dadas as dificuldades enfrentadas na reprodução de outros aceleradores, e considerando a existência de conversores de outras redes neurais à descrições de hardware existentes, tentativas foram feitas para a conversão de código em alto nível em hardware otimizado.

2 Experimentos com HLS e modelos Python

2.1 Apache TVM

Apache TVM é um compilador open source para aprendizado de máquina [2]. É otimizado para ter flexibilidade para executar os modelos em CPU, GPU e outras plataformas. Dado que grande parte dos modelos de aprendizado profundo são desenvolvidos a partir de frameworks em Python, a ferramenta objetiva permitir execução de modelos em hardware otimizado para performance sem que haja perda de produtividade ao simplificar ao desenvolvedor o desenvolvimento. TVM tem suporte a frameworks como Pytorch, Tensorflow e Keras, e objetiva apresentar uma solução unificada para diferentes plataformas, portanto sendo agnósticas a hardware. Além disso, TVM permite estender as otimizações geradas pelo compilador e reduz dependências a ferramentas específicas de hardwares.

O funcionamento do compilador pode ser descrito em três etapas: front-end, middle-end e back-end. O frontend é responsável pelo parsing do modelo de entrada e geração de uma representação intermediária *Relay IR*, que permite a definição dos grafos de dados e computações do modelo. O middle-end é responsável por otimizar a representação dem Relay do modelo, fundindo operações e removendo operações custosas. Finalmente, o back-end faz otimizações específicas do hardware alvo, e gera código em baixo nível para ser executado. Desse modo, TVM tem partes independentes que permitem maior flexibilidade nas otimizações.

No que se refere a Graph Neural Networks (GNNs), TVM tem suporte limitado aos modelos. Em especial, a compatibilidade do compilador com as bibliotecas Python de GNN PyG e DGL era limitada se comparada a demais ferramentas de redes de aprendizado profundo e, especialmente tem documentação escassa.

No que se refere a FPGSs, TVM é compatível com Vitis AI. Vitis AI é um ambiente de desenvolvimento projetado pela Xilinx para simplificar implementação e inferência de modelos de redes neurais em FPGAs. Vitis AI inclui IPs otimizados, ferramentas de compilação e otimização de modelos, bibliotecas de software, modelos pré-treinados, tutoriais e exemplos de design. Entretanto, a integração das ferramentas também é limitada: os hardwares suportados pela integração são limitados e a documentação e guias de implementação com TVM não podem ser mais encontrados.

Em termos dos experimentos realizados, conforme mencionado no relatório anterior, tentativas de aceleração iniciais consistiram na reprodução a partir da descrição de modelos e aceleradores na literatura. Conforme mencionado, essa abordagem apresentou diversos desafios no que se refere a integração dos modelos a FPGAs. Por outro lado, a inferência de GNNs utilizando as bibliotecas PyG e DGL em GPU apresentou-se como uma tarefa mais simples, devido à integração nativa dessas ferramentas com otimizações em GPU. Desse modo, dado os sucessos em execução de inferências de GNNs em CPUs e GPUs a partir de bibliotecas em Python, e por se tratar de uma ferramenta open-source robusta com suporte a modelos em alto nível, Apache-TVM foi optada como alternativa para o projeto. Além disso, a aparente compatibilidade com GNNs e FPGAs significaria relativamente baixo esforço de implementação, o que era esperado dado que se objetivava a obtenção de uma baseline funcional. Finalmente, por ter uma IR agnóstica da entrada gerada pelo frontent, e um backend otimizado a partir de Relay para FPGAs, foi optada por ter uma abordagem agnóstica.

Entretanto, como as abordagens anteriores, o uso de Apache TVM mostrou-se desafiador. Em primeiro lugar, não havia documentação clara da compatibilidade da ferramento com a FPGA utilizada, de modo que possíveis incompatibilidades possam ter comprometido seu uso. Além disso, não foi possível gerar de forma consistente IR pelo frontend da ferramenta em modelos de GNN para teste, devido a incompatibilidades entre as bibliotecas Python de GNN e TVM. A depuração do processo de compilação também se mostrou limitada, dificultando a identificação dos pontos de falha. Finalmente, embora TVM seja uma ferramenta open-source ativa, a manutenção, desenvolvimento e suporte a certas funcionalidades – como integração com GNNs – não parecem ocorrer com a mesma frequência ou prioridade que outras áreas da ferramenta. Essa limitação impacta diretamente o suporte oferecido pela comunidade, bem como a disponibilidade de soluções e documentação atualizada para problemas específicos relacionados a GNNs.

2.2 ONNX Runtime

Após as tentativas com o compilador, novas abordagens foram exploradas com ONNX Runtime. Open Neural Network Exchange (ONNX) é um padrão de representação de modelos de aprendizado de máquina que permite a interoperabilidade entre frameworks como PyTorch, TensorFlow, Keras, Scikit-Learn, entre outros. Com isso, modelos treinados a partir de um conjunto de ferramentas em um ambiente podem ser exportados e executados em diferentes plataformas sem perda de funcionalidade.

ONNX representa modelos de aprendizado de máquina como grafos computacionais. Nesse formato, cada nó do grafo corresponde a uma operação ou operador específicos, e as arestas indicam o fluxo de dados entre essas operações. Além disso, o ONNX define um conjunto de operadores que devem estar disponíveis em todos os frameworks que suportam o formato, o que garante a portabilidade dos modelos. O ONNX também especifica tipos de dados padrão, assegurando consistência na interpretação dos dados entre diferentes plataformas. Desse modo, ONNX pode ser comparado a uma linguagem que implementa funcionalidades matemáticas e, como mencionado, é suportada de forma nativa por diversos frameworks.

ONNX Runtime (ORT), por sua vez, é é um motor de execução otimizado para modelos descritos no formato ONNX. ORT é compatível com diferentes hardwares, linguagens como C++, Python, C#, Java e JavaScript e sistemas operacionais. A partir de um modelo pré-treinado e convertido no formato ONNX, ORT realiza diversas otimizações no grafo do modelo, e depois o divide em subgrafos com base nos aceleradores

específicos. Os kernels otimizados presentes no núcleo do ONNX Runtime oferecem melhorias de desempenho, e os subgrafos atribuídos a diferentes Execution Providers (EP) - módulos que permitem a execução em diferentes hardwares - recebem aceleração adicionais. ORT é compatível com Vitis AI por meio de Vitis AI Execution Provider, que permite a execução eficiente de modelos ONNX em aceleradores AMD, como FPGAs e CPUs.

ONNX e ORT apresentaram inicialmente ferramentas promissoras na execução da inferência de modelos pré-treinados pela sua simplicidade de uso e compatibilidade com FPGAs AMD. A integração nativa com frameworks como Pytorch e Keras permite a geração de representação ONNX de modelo simples. Entretanto, a integração com GNNs mostrou-se novamente desafiadora: em primeiro lugar, DGL não é compatível com o formato ONNX, o que limita as alternativas de uso. ALém disso, apesar de Pytorch ser compatível, PyG também mostrou-se limitado na conversão dos modelos: enquanto alguns modelos não puderam ser convertidos, outros geraram a representação ONNX, mas que apresentou erros em ser executada pelo Runtime. Além disso, mensagens de erro não podiam ser analisadas via documentação e indicavam incompatibilidade em versões de biblioteca que estava corretas. Finalmente, não havia compatibiliadde clara com as placas disponíveis, de modo que mesmo em modelos de redes neurais que foram corretamente compilados, houve erro na execução do Runtime para geração dos aceleradores em placas FPGA. Esses erros não eram acompanhados de mensagens informativas, dificultando o diagnóstico. Finalmente, muitas documentações específicas da integração com Vitis AI têm *dead links*, o que indicaria possivelmente que a funcionalidade tem atualmente suporte reduzido.

Assim, apesar das vantagens teóricas de interoperabilidade e otimização oferecidas pelo ONNX Runtime, sua aplicação para GNNs em FPGAs mostrou-se limitada e de difícil integração no ambiente experimental do projeto, especialmente considerando o objetivo de estabelecer uma baseline funcional com de forma simples e a simplicidade de alguns modelos testados.

2.3 GNNHLS

GNNHLS é uma ferramenta desenvolvida e descrita em [5] para facilitar a implementação e avaliação de inferência de Graph Neural Networks utilizando High-Level Synthesis (HLS) para FPGAs. A ferramenta visa automatizar o processo de geração de hardware a partir de descrições em alto nível de modelos de GNNs em Python, permitindo maior rapidez e flexibilidade na avaliação dos modelos.

O funcionamento da GNNHLS parte da entrada de um modelo de GNN descrito em alto nível, que é transformado em uma representação que pode ser sintetizada para hardware de forma otimizada. Além disso, ela permite configurar parâmetros como número de unidades de processamento, largura dos barramentos e estratégias de acesso à memória, a fim de lidar com a natureza irregular dos dados de grafos. Após a geração do projeto de hardware, GNNHLS permite a coleta de métricas sobre uso de recursos da FPGA, latência, throughput e consumo energético para comparação entre implementações.

O artigo "GNNHLS: Evaluating Graph Neural Network Inference via High-Level Synthesis" [5] propõe a ferramenta e analisa sua aplicação em diferentes arquiteturas de GNNs, como Graph Convolutional Networks (GCNs) e Graph Attention Networks (GATs), avaliando seu desempenho em termos de latência, uso de recursos e consumo de energia. GNNHLS permite explorar a execução em diferentes configurações de hardware e otimizar os modelos em FPGAs. Os resultados mostram que, embora a complexidade estrutural das GNNs represente um desafio para a síntese de alto nível, é possível obter soluções eficientes com um bom equilíbrio entre desempenho e consumo energético. Mais especificamente, detalha que GNNHLS alcança até 50,8x em termos de speedup e 423x redução de consumo energético se comparado a abordagens em CPU, e 5,16x e 74,5x em relação a GPU.

Em termos dos experimentos, GNNHLS representou uma alternativa promissora em relação às abordagens anteriores por ser ter código aberto, ser desenvolvida especificamente para GNNs e FPGAs, considerando

que abordagens anteriores apresentaram incompatibilidades com o modelo ou o hardware. Além disso, seu uso em artigos como [5] evidencia seu uso prático. No entanto, novamente, a aplicação da ferramenta não teve resultados positivos: um dos grandes desafios em seu uso mostro-se em incompatibilidades internas entre suas dependências: apesar de disponibilizar configurações para execução nativa e em ambiente Docker, as dependências das ferramentas utizadas apontavam erros que não puderam ser contornados. Além disso, mesmo após adaptações manuais no Além disso, mesmo após adaptações manuais no ambiente Docker e código, persistiram falhas na execução dos scripts fornecidos, especialmente nas etapas de síntese e geração de bitstreams. Desse modo, apesar do potencial da ferramenta evidenciado pelos resultados apresentados no artigo, a possível falta de maturidade e suporte da GNNHLS comprometeu sua viabilidade como alternativa prática no projeto.

2.4 hls4ml

O artigo "hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices" [1] apresenta *hls4ml*, uma biblioteca para inferência de modelos de machine learning em FPGAs. A ferramenta compreende um fluxo de trabalho de co-design de software e hardware de código aberto, para facilitar a implementação de algoritmos de aprendizado de máquina em dispositivos de baixa potência, como FPGAs e ASICs, por meio de High Level Synthesis. Desse modo, a ferramenta visa tornar a implementação de hardware de sistemas de machine learning mais acessível, reutilizável e interpretável, de modo a reduzir o *time to science*.

O funcionamento da ferramenta ocorre a partir do treinamento de um modelo de aprendizado de máquina usando uma biblioteca de alto nível, como Keras ou Pytorch. Esse modelo é exportado e fornecido como entrada para o hls4ml. hls4ml interpreta a estrutura da rede neural - como camadas, parâmetros, pesos - e a converte em código C++ estruturado para ser usado com ferramentas HLS. Esse código C++ é otimizado para refletir a arquitetura da rede e sua lógica de inferência. O usuário pode personalizar a implementação definindo parâmetros como precisão numérica, paralelismo e uso de recursos. O código C++ gerado é passado para ferramentas HLS - como Vivado HLS - que sintetizam o projeto para um alvo de hardware específico como um FPGA, que gera uma descrição em HDL - como VHDL ou Verilog - que pode ser integrada a um projeto de hardware. Além disso, hls4ml também oferece ferramentas para simular a execução do modelo no hardware gerado, permitindo comparar a precisão com o modelo original e analisar métricas como latência, throughput, uso de recursos do hardware e consumo energético.

A ferramenta indica em sua documentação suporte a GNNs e o suporte a ferramentas de High Level Synthesis compatíveis com os utilizados nas placas disponíveis para experimentação. Entretanto, os experimentos práticos com GNNs em FPGAs também apresentaram problemas: em primeiro lugar, a compatibilidade da ferramenta com Pytorch - biblioteca na qual ferramentas para descrição de GNNs em Python se baseiam - se mostrou limitada, dado que parte considerável das operações implementadas na biblioteca não tinham equivalentes na ferramente. Além disso, a ferramenta não permite operações de *scatter* e *Message Passing*, importantes na maior parte das GNNs. Finalmente, se contornados os erros pela modificação e simplificação dos modelos, novos erros eram gerados devido ao número de parâmetros dos modelos. Apesar de haver tentativas da ampliação do suporte da ferramenta a GNNs por contribuidores, não foram integradas às releases principais, e tentativas de seu uso não tiveram sucesso.

3 Experimentos com Vitis HLS

Conforme mencionado, após as tentativas iniciais de reprodução de aceleradores a partir de artigos e o uso de ferramentas de geração de hardware a partir de representações de alto nível, tentativas subsequentes foram feitas para a geração do hardware a partir de Vitis HLS. Vitis HLS é um conjunto de ferramentas

desenvolvido para a conversão de código em C ou C++ em uma representação em RTL em FPGAs AMD. Os experimentos, como os anteriores, foram executados na *A-Machine*, cluster de FPGAs Xilinx U55C, a partir da descrição da descrição do hardware e C++ com diretivas de HLS.

A implementação se baseou na implementação descrita no artigo "Accelerating GNN-based SAR Automatic Target Recognition on HBM-enabled FPGA" [3], que desenvolve via Vitis HLS um acelerador para GNN de visão computacional. O artigo propõe o uso de uma GNN para Reconhecimento Automático de Alvos (ATR) em Radares de Abertura Sintética (SAR) e um acelerador em FPGA que a processe. Imagens geradas por radares podem ser consideradas esparsas uma vez que a maioria dos pixels é irrelevante para classificação, de modo que são transformadas em grafos conectando com arestas pixels vizinhos e removendo os nós correspondentes a pixels desnecessários [3, 4]. A arquitetura de hardware proposta implementa módulos para processamento dos kernels de Feature Aggregation - combinação de características de nós vizinhos - e Feature Update - atualização da matriz de *features* de um nó - respectivamente usando paradigma Scatter-Gather e matriz sistólica de ALUs. A implementação em FPGA Xilinx Alveo U280 obteve 5,2x e 1,57x menor latência, 36,2x e 4,9x maior eficiência energética e 10x e 3,3x maior throughput em comparação com implementações em CPU e GPU, respectivamente.

Nesse caso, tentativas foram feitas a partir da implementação do acelerador mencionado, considerando as falhas anteriores com ferramentas de geração de hardware a partir de descrições em alto nível. Entretanto, a execução do novo modelo mostrou-se novamente desafiadora: a replicação do projeto original esbarrou em dificuldades relacionadas à falta de documentação detalhada, ausência de partes do código-fonte disponibilizado, além de incompatibilidades de versão entre o Vitis HLS e as bibliotecas utilizadas no projeto original. Apesar do esforço de adaptação do código às versões mais recentes da ferramenta e da infraestrutura disponível, não foi possível obter uma versão funcional e completa do acelerador. Além disso, a síntese do hardware novamente apresentou erros de difícil identificação, e foram encontrados problemas na compilação do código C++ com diretivas de HLS removidas. Finalmente, não foi possível, também, a simulação do hardware nativa de forma consistente nas ferramentas da fabricante, o que prejudicou o desenvolvimento e adaptação do modelo. Experimentos com novos modelos a partir do hardware de [3] também não tiveram sucesso, uma vez que o projeto original apresenta forte acoplamento com a estrutura da aplicação proposta, dificultando sua generalização para outras arquiteturas de GNNs. Apesar do fato de que Feature Aggregation e Update serem conceitos essenciais em GNNs, a tentativa de adaptar o pipeline de processamento e de seus módulos para diferentes representações de grafo resultou em inconsistências e novos erros de síntese, indicando rigidez na forma como o design foi concebido e implementado. Essas limitações evidenciam um dos principais desafios no uso de Vitis HLS para aplicações baseadas em GNNs: a baixa portabilidade de aceleradores complexos entre diferentes modelos e domínios, mesmo quando se adota uma ferramenta de síntese de alto nível. O alto nível de customização do hardware para um domínio específico, embora traga ganhos significativos de desempenho, torna o reaproveitamento da implementação difícil sem uma reengenharia extensa, em especial em áreas com literatura limitada, se comparada com outros modelos e aceleradores.

4 Conclusão e próximos passos

Neste relatório, foram exploradas ferramentas e abordagens com o objetivo de acelerar a inferência de Graph Neural Networks (GNNs) em FPGAs, a partir de descrições em alto nível e implementações mais próximas ao hardware. Apesar dos benefícios de desempenho e flexibilidade oferecidas por frameworks como Apache TVM, ONNX Runtime, GNNHLS e hls4ml, os experimentos revelaram limitações significativas no suporte a GNNs, na compatibilidade com os dispositivos disponíveis e na maturidade das ferramentas para esse domínio específico. Ferramentas generalistas de compilação como TVM e ONNX Runtime, embora consolidadas em outras áreas do aprendizado de máquina, ainda apresentam suporte incipiente para GNNs e integração com

FPGAs, especialmente no contexto de bibliotecas como PyG e DGL. Já soluções específicas como GNNHLS e hls4ml, mesmo com propostas mais direcionadas, ainda enfrentam desafios relacionados à complexidade estrutural dos modelos e à manutenção do ecossistema de software.

A tentativa de replicação e adaptação de aceleradores propostos na literatura com Vitis HLS também apresentou desafios, como documentação incompleta, dependências desatualizadas e dificuldades na adaptação dos projetos às ferramentas e infraestrutura atuais. Esses obstáculos, como os descritos no relatório anterior, revelam uma lacuna entre os resultados reportados em publicações e a reprodutibilidade em ambientes reais de desenvolvimento.

Dessa forma, embora existam caminhos promissores para a aceleração de GNNs em FPGAs, o estado atual de ferramentas e frameworks ainda impõe barreiras significativas. Desse modo, propõe-se que seja analisada a aceleração de Redes Neurais Convolucionais em FPGAs, considerando sua abudância de literatura e estrutura regular. Em especial, conforme mencionado na Proposta de Projeto, objetiva-se utilizar o cluster A-Machine para estender a aceleração em um cluster Multi-FPGA, conforme será discutido nas entregas ao longo do semestre.

References

[1] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*, 2021.

[2] Kausthub Thekke Madathil, Abhinav Dugar, Nagamma Patil, and Unnikrishnan Cheramangalath. Optimizing machine learning operators and models for specific hardware using apache-tvm. In *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2023.

[3] Bingyi Zhang, Rajgopal Kannan, Viktor Prasanna, and Carl Busart. Accelerating gnn-based sar automatic target recognition on hbm-enabled fpga. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2023.

[4] Bingyi Zhang, Sasindu Wijeratne, Rajgopal Kannan, Viktor Prasanna, and Carl Busart. Graph neural network for accurate and low-complexity sar atr. *arXiv preprint arXiv:2305.07119*, 2023.

[5] Chenfeng Zhao, Zehao Dong, Yixin Chen, Xuan Zhang, and Roger D Chamberlain. Gnnhls: Evaluating graph neural network inference via high-level synthesis. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*, pages 574–577. IEEE, 2023.