Unidade de Processamento Neural Extensível a Partir de um Dispositivo Lógico Programável

Thiago Cruz, Jemerson Damasio, Danilo Santos, Danyllo Albuquerque,
Mirko Perkusich, and Hyggo Almeida
VIRTUS - Núcleo de Pesquisa, Desenvolvimento e Inovação
Universidade Federal de Campina Grande (UFCG)
Campina Grande, Paraíba - Brazil
(thiago.cruz,jemerson.damasio,danilo.santos)@virtus.ufcg.edu.br
(danyllo.albuquerque,mirko,hyggo)@virtus.ufcg.edu.br

RESUMO

O aprendizado profundo (do inglês, *Deep Learning*) representa uma técnica poderosa para resolver problemas complexos de aprendizado. Com o crescimento dos dispositivos integrados combinado com uma demanda por baixa latência e melhoramento continuo, os modelos treinados precisam cada vez mais ser executados de forma eficiente. A fim de atender essas demandadas, bem como manter o baixo custo de energia, neste artigo é apresentada a experiência do desenvolvimento de uma Unidade de Processamento Neural baseado em uma arquitetura de acelerador escalável para redes de aprendizado profundo em larga escala usando o *Field-Programmable Gate Array* (FPGA) como o protótipo de hardware.

KEYWORDS

Inteligência Artificial, Aprendizado Profundo, FPGA, Desempenho, Acelerador de Hardware

1 INTRODUÇÃO

O aprendizado profundo usa um modelo de rede neural multicamadas que possuem seus parâmetros atualizados por meio de uma função de custo que, junto a um conjunto de dados, permite que o sistema aprenda os padrões presentes e assim generalizar previsões para dados não vistos durante o treinamento [4]. Entre os modelos neurais de aprendizado profundo mais usados estão as Redes Neurais Profundas (do inglês, *Dense Neural Network*) (DNNs) [1] e as Redes Neurais Convolucionais (do inglês, *Convolutional Neural Network*)(CNNs)[9], que provaram ter excelente capacidade de prover suporte ao aprendizado de máquina complexo.

No entanto, com o aumento da complexidade das aplicações práticas, os modelos tem se tornado cada vez mais robustos como o *Baidu Brain* e o sistema de reconhecimento de gatos do Google, ambos com mais de 1 bilhão de conexões neuronais [7]. O volume explosivo de dados torna o processamento neural custoso em termos de processamento e consumo de energia. Portanto, existem desafios significativos para implementação de unidades de processamento neural de alto desempenho com baixo custo de energia, especialmente para modelos de redes neurais de aprendizado profundo em larga escala [9].

ISE '22, October 4, 2022, Proceeding-based Workshop, Brazil
2022. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ISE '22: 2nd Brazilian Workshop on Intelligent Software Engineering, October 4, 2022, Virtual, Brazil*, https://doi.org/10.5753/ise.2022.227056.

Até agora, as soluções de última geração para acelerar algoritmos de aprendizado profundo são os Arranjos de Portas Programáveis (do inglês, *Field-Programmable Gate Array*) (FPGA), o Circuito Integrado Específico de Aplicação (do inglês, *Application Specific Integrated Circuit*) (ASIC) e a Unidade de Processamento Gráfico (do inglês, *Graphic Processing Unit*) (GPU)[7]. Em comparação com a aceleração através de GPU, aceleradores de hardware como FPGA e ASIC podem atingir desempenho moderado com menor consumo de energia. Contudo, tanto o FPGA quanto o ASIC têm recursos de computação relativamente limitados, como memória e largura de banda de entrada/saída. Portanto, também é um desafio desenvolver redes neurais profundas complexas e massivas usando esses aceleradores de hardware.

Atualmente, em torno das pesquisas de aceleração de FPGA, Ly e Chow [5] projetaram soluções baseadas em FPGA para acelerar a Máquina Boltzmann Restrita (do inglês, *Restricted Boltzman Machine*) (RBM). De forma semelhante, Kim *et al.* [2] também criaram um acelerador baseado em FPGA para o RBM. Esses estudos usam vários módulos de processamento RBM paralelos, sendo cada módulo responsável por um número relativamente pequeno de nós. Outros estudos semelhantes também apresentam aceleradores de redes profundas baseados em FPGA [6] [8]. Em resumo, os estudos mencionados acima se concentram na implementação eficiente de um algoritmo de aprendizado profundo específico, mas exibiram ou discutiram meios de como suportar um aumento no tamanho da rede neural profunda com uma arquitetura de hardware escalável e flexível.

Com o objetivo de resolver esse problema, o presente artigo apresenta a experiência em desenvolver uma Unidade de Processamento Neural Extensivel (do inglês, *Extensible Neural Processing Unity*) (XNPU) para acelerar as partes computacionais do *kernel* de algoritmos de aprendizado profundo como um acelerador de hardware em FPGA. Esta solução é totalmente flexível e customizável, podendo utilizar diversas topologias de modelos de aprendizado profundo. A seguir apresentaremos as diretrizes utilizadas no processo de implementação desta solução, seus desafios e os próximos passos desta pesquisa.

2 EXEMPLO MOTIVADOR

Entre as unidades de processamento neural (do inglês, *Neural Processing Unity*) (NPU) mais notórias da atualidade estão aquelas pertencentes as empresas Google e à Huawei. Os chips da linha

Ascend[1] da Huawei usam um conjunto de instruções próprias para aprendizado profundo, de modo a processar e simular em circuitos eletrônicos um grande número de nós. O *Ascend* 310 atinge as marcas de 8 Tera-FLOPS para meia precisão de ponto flutuante (FP16) e 16 Tera-FLOPS para precisão de inteiros (INT8), consumindo no máximo 8W. Por outro lado, o *Ascend* 910 chega a 256 Tera-FLOPS para meia precisão e 512 para precisão de inteiro, e um consumo máximo de 350W.

O acelerador baseado em unidade de processamento tensorial (do inglês, *Tensor processing unit*)(TPU) *Edge*[2] da empresa Google permite a implementação de inferências de aprendizado de máquina com alta qualidade, no ambiente de programação de código aberto *TensorFlow Lite*[3]. Esse dispositivo consegue combinar hardware personalizado com software aberto e soluções de inteligência artificial para atuar nas mais diversas aplicações, desde detecção de anomalias e reconhecimento de voz até visão de computacional.

3 ARQUITETURA DA SOLUÇÃO

A arquitetura proposta é composta de dois módulos: (i) *Ferramentas de desenvolvimento em PC* e (ii) *FPGA*. Esses módulos são descritos nos diagramas ilustrados na Figura 1. O primeiro módulos é responsável pela criação e treinamento da rede quantizada, além de envio de dados para o módulos FPGA, sendo composto pelos seguintes blocos:

- *Script em Python*: Responsável pela serialização dos dados e seu posterior envio para o módulo FPGA.
- *Arquivo de dados*: Conjunto de dados usado como entrada tanto da rede neural hospedada na XNPU quanto da rede quantizada que será usada como referência.
- *Script comparador*: Responsável por realizar a comparação entre dois conjuntos de dados da mesma dimensão.
- *Tensorflow script*: Script usando o *framework* de código aberto TensorFlow.
- *Rede neural*: Modelo de rede neural ou rede neural convolucional adptado para um cenário específico de uso.
- *Aprendizado*: Script de treinamento do modelo.
- *Quantização*: Conversão de parâmetros de rede de float32 para int8 usando uma API do TensorFlow.
- *Inferência*: Script de cálculo de inferência de modelo quantizado.

O módulo *FPGA* é responsável pela aceleração de hardware da solução proposta. Os blocos instanciados na FPGA são:

- *UART*: Este é o transmissor/receptor assíncrono universal (do inglês, *Universal Asynchronous Receiver/Transmitter*). É o periférico responsável pela comunicação entre o XNPU e outros dispositivos através de um procolo de comunicação.
- *Controle*: Lida com solicitações de outros dispositivos para orquestrar os blocos de inferência e convolução para executar a função solicitada. Internamente, possui um banco de *flip-flops*, acessados via endereço, que servem para configurar a XNPU e verificar seus passos de operação, além de um sub-bloco interno de quantização para oferecer a opção de conversão dos dados de entrada para int8 caso não estejam nesse formato. Possui interface com o bloco UART e o barramento.
- *Memória*: Conjunto de *flip-flops* que simulam a memória volátil para armazenar os pesos e parâmetros de quantização do modelo treinado, os dados de entrada e os resultados de inferência. Os dados armazenados serão acessados através do bloco Controle.
- *Núcleo de inferência*: Componente desenvolvido para realizar o algoritmo de inferência.
- *Núcleo de convolução*: Componente desenvolvido para realizar o algoritmo de convolução. Possui uma interface com o core de inferência pois depende do mesmo para realizar o algoritmo de CNN.
- *Barramento*: É um barramento de porta lógica OR que tem por objetivo prover a comunicação entre o núcleo de inferência e núcleo de convolução com o bloco de controle. Este tipo de barramento foi escolhido devido à sua simplicidade e baixa inserção de latência.

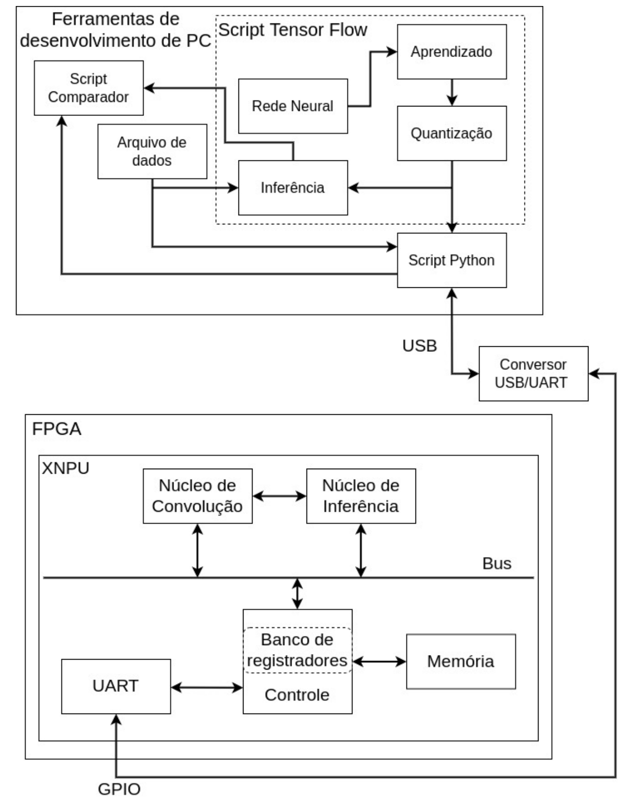

Figura 1: Arquitetura da solução proposta

A comunicação entre o módulo *Ferramentas de desenvolvimento em PC* e o módulo *FPGA* é realizada por um dispositivo capaz de converter o protocolo de comunicação UART e o protocolo de comunicação da porta serial universal (do inglês, *Universal Serial Bus*)(USB). Seus pinos são conectados aos pinos de Entrada/Saída de Uso Geral (do inglês, *General Purpose Input/Output*)(GPIO) da *FPGA* e a uma porta USB.

[1]https://bit.ly/3Ea7ew6
[2]https://cloud.google.com/edge-tpu
[3]https://www.tensorflow.org/

4 DESCRIÇÃO DA SOLUÇÃO

Com base na arquitetura previamente definida, implementou-se uma Unidade de Processamento Neural Extensível a partir de um Dispositivo Lógico Programável de acordo com o que segue.

4.1 Multiplicação em Hardware

A multiplicação em hardware é algo muito custoso em termos de processamento computacional. Para lidar com esse problema e, aproveitando-se do fato do algoritmo implementado na solução lida apenas com inteiros, foi utilizado um hardware que implementa o algoritmo da multiplicação camponesa russa [3]. É uma técnica relevante para multiplicação de dois inteiros que permite realizar o cálculo com apenas com operações de soma, deslocamento à esquerda e deslocamento à direita.

4.2 Hardware da Quantização

A quantização é obtida pela seguinte formula:

$$y_q = \frac{y_r}{s} + z_p \tag{1}$$

- y_q: Representação do número como int8
- y_r: Representação do número como float32
- s: Escala do número que é um float32 parâmetro de quantização
- z_p: Ponto zero que é um int8 parâmetro de quantização

O hardware responsável por realizar esse operação possui a rquitetura descrita através do diagrama apresentado na Figura 2:

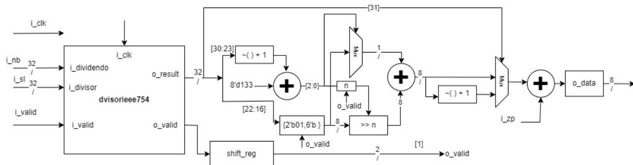

Figura 2: Arquitetura de hardware de quantização

É importante mencionar que o hardware de quantização fica localizado no bloco de controle onde pode ser ativo via escrita no banco de *flip-flops* para quantizar a entrada.

4.3 Formulação Matemática da Inferência Quantizada

Manipulando a equação da quantização para obter o valor float32, temos a seguinte equação:

$$y_r = (y_q - z_p) * s \tag{2}$$

Sendo assim, a equação de multiplicação que ocorre em um neurônio se dá do seguinte modo:

$$y_r = w * x + b \tag{3}$$

Onde:

- w é a matriz de pesos
- x é a entrada da camada atual (saída da camada anterior)
- b é o vetor coluna com o bias de cada neurônio

Considerando apenas um elemento, temos:

$$y_r^{i,j} = \sum_k (w^{i,j} * x^{j,k}) + b^{i,j} \tag{4}$$

Substituindo:

$$(y_q^{i,j} - z_{py}) * s_y = \sum_k ((w_q^{i,j} - z_{pw}) * s_w * (x_q^{j,k} - z_{px}) * s_x) + (b_q^{i,j} - z_{pb}) * s_b \tag{5}$$

A escala do *bias* é definida como $s_w \times s_x$, logo, podemos colocar em evidência:

$$(y_q^{i,j} - z_{py}) * s_y = s_b [* \sum_k ((w_q^{i,j} - z_{pw})(x_q^{j,k} - z_{px})) + (b_q^{i,j} - z_{pb})] \tag{6}$$

$$y_q^{i,j} - z_{py} = \frac{s_b}{s_y} [* \sum_k ((w_q^{i,j} - z_{pw})(x_q^{j,k} - z_{px})) + (b_q^{i,j} - z_{pb})] \tag{7}$$

$$y_q^{i,j} - z_{py} = M * [\sum_k ((w_q^{i,j} - z_{pw})(x_q^{j,k} - z_{px})) + (b_q^{i,j} - z_{pb})] \tag{8}$$

Pela especificação de quantização do *Tensorflow*, os pontos zero de todo peso e de todo bias é 0.

$$y_q^{i,j} - z_{py} = M * [\sum_k ((w_q^{i,j})(x_q^{j,k} - z_{px})) + (b_q^{i,j})] \tag{9}$$

$$y_q^{i,j} = M * [\sum_k ((w_q^{i,j})(x_q^{j,k})) - \sum_k (w_q^{i,j} * z_{px}) + (b_q^{i,j})] + z_{py} \tag{10}$$

O único parâmetro real dessa equação é o M, mas ele pode ser normalizado. Todas as outras operações são feitas com inteiros, sendo o bias $int32$ e todas as demais $int8$. Além disso, o segundo somatório da equação pode ser calculado *offline*, de modo que apenas o primeiro precisa ser calculado durante a inferência.

$$C = M * [\sum_k (w_q^{i,j} * z_{pi}) + (b_q^{i,j})] + z_{py} \tag{11}$$

$$y_q^{i,j} = M * [\sum_k ((w_q^{i,j})(x_q^{j,k}))] - C \tag{12}$$

$$M = M_0 * 2^{-n} \tag{13}$$

O hardware responsável por realizar esse operação possui a seguinte arquitetura conforme descrito na Figura 3:

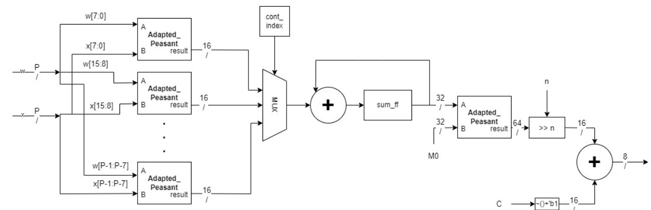

Figura 3: Arquitetura de hardware de inferência do neurônio.

Em seu interior existe um hiper-parâmetro responsável por indicar a quantidade instancias do sub-bloco de multiplicação. Tanto o módulo de convolução quanto de inferência necessitam desse hardware para realizarem seus respectivos algoritmos.

4.4 Mapa de Memória

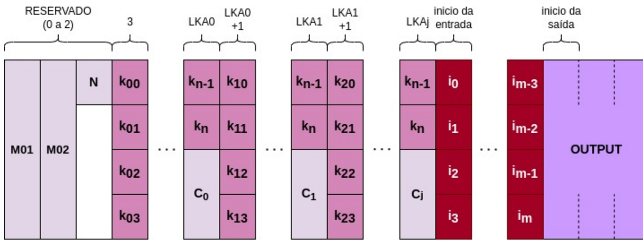

Figura 4: Mapa de memória

Os dados são dispostos na memória por camada, e de acordo com sua necessidade de acesso, conforme apresentado no diagrama da Figura 4. As constantes de quantização podem ser únicas por camada ou variar para cada neurônio, a depender se é uma camada de convolução ou uma camada densa. Dados bidimensionais como as entradas, os pesos e os filtros, são guardados vetorizados em linha, da extremidade superior esquerda para a extremidade inferior direita.

4.5 Descrição do Fluxo

Para o *pipeline*, inicialmente é escrito na memória todos os parâmetros da rede. Essa etapa é concluída com a escrita das informações da topologia no banco de *flip-flops* no bloco de controle. Em seguida o cálculo é iniciado com uma escrita em um endereço específico. Durante o processamento, deve-se realizar leituras no banco de *flip-flops* para saber o momento do fim do processamento. E assim realizar uma leitura na memoria para se obter o resultado.

5 RESULTADOS E DISCUSSÃO

Para avaliar a viabilidade da Unidade de Processamento Neural Extensível a partir de um dispositivo lógico programável, realizou-se a prototipagem da arquitetura descrita no presente estudo através do software Quartus II[4] para carregamento em FPGA. Mediante esta prototipagem, buscou-se identificar e analisar os parâmetros resultantes da síntese do circuito. Assim, foram utilizados dois cenários para avaliação, sendo obtido os seguintes valores.

Cenário 1. Para o hardware referente a equação de quantização, obteve-se os seguintes parâmetros:

- Elementos lógicos totais: 192;
- Funções combinacionais totais: 192;
- Registros lógicos dedicados: 96;
- Total de registros: 96;
- Total de pinos: 83;
- Dissipação Total de Energia Térmica: 38,88 mW;
- Dissipação de energia térmica estática do núcleo: 18.02 mW; e
- Dissipação de energia térmica de E/S: 20,86mW.

Cenário 2. Para o hardware referente a equação de inferência quantizada com uma instancia da multiplicação camponesa russa, obteve-se os seguintes parâmetros:

- Elementos lógicos totais: 1497;
- Funções combinacionais totais: 1495;
- Registros lógicos dedicados: 519;
- Total de registros: 519;
- Total de pinos: 117;
- Dissipação Total de Energia Térmica:42,63 mW;
- Dissipação de energia térmica estática do núcleo: 18,03 mW; e
- Dissipação de energia térmica de E/S: 24,61mW.

Conclui-se que o cenário 1 provê indícios de que o sub-bloco de quantização é viável para a arquitetura. Por outro lado, o cenário 2 provê indícios de que tende a ser mais eficiente a utilização do algoritmo de multiplicação russo do que uma multiplicação convencional.

[4] https://intel.ly/3RrlrI2

6 CONSIDERAÇÕES FINAIS

A arquitetura proposta demonstra a viabilidade do uso de aceleradores de hardware FPGA para o desenvolvimento de soluções que necessitem de processamento de dados em topologia de rede ampla. A integração com o ambiente TensorFlow contribuiu para agilizar a implantação da solução devido ao fato de todo o ferramental estar disponível gratuitamente, facilitando a reprodutibilidade dessa arquitetura por pesquisadores e profissionais do setor. Como passos futuros, será realizado um aprofundamento da arquitetura proposta para otimizar a latência e a adição de um módulo de *clock gating* para avaliar o consumo de energia.

REFERÊNCIAS

[1] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–40. IEEE, 2015.
[2] S. K. Kim, L. C. McAfee, P. L. McMahon, and K. Olukotun. A highly scalable restricted boltzmann machine fpga implementation. In *2009 International Conference on Field Programmable Logic and Applications*, pages 367–372. IEEE, 2009.
[3] C. U. Kumar and B. J. Rabi. Design and implementation of modified russian peasant multiplier using msqrtcsla based fir filter. *Indian Journal of Science and Technology*, 9(7):1–6, 2016.
[4] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
[5] D. L. Ly and P. Chow. A high-performance fpga architecture for restricted boltzmann machines. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 73–82, 2009.
[6] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 26–35, 2016.
[7] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.
[8] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou. A deep learning prediction process accelerator based fpga. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1159–1162. IEEE, 2015.
[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.