C-004

FPGA を用いた組込みデバイス向け AI アクセラレータ実装ツールの開発

Development of an AI accelerator implementation tool for embedded FPGAs

椿 翔太† 増田 信之†
Shota Tsubaki Nobuyuki Masuda

1. はじめに

FPGA によるアクセラレータは，通信や科学計算といった計算量が多いタスクで速度が求められる様々な分野において使用されている[1]．最近では深層学習モデルに対して FPGA を用いて高速に処理するシステムが盛んに研究されている．深層学習モデルとアクセラレータの観点でいえば，GPU もよく使われるデバイスである．GPU と FPGA を比較した際 FPGA の方が低消費電力であり，より組込み分野に向いているデバイスといえる[2]．この 2 つのデバイスでは開発のやり方が大きく異なり，一般的に FPGA の方が開発にかかるリソースが多いされている[3]．

そこで本研究では，FPGA 向けに深層学習モデルの実装を支援するツールを作成した．特に FPGA を用いたアクセラレータの需要が高いと考えられる組込み向けの FPGA を対象とし，既存の高位合成コンパイラと本実験で作成した支援ツールを組み合わせて用いることで，デバッグなどの実装までの時間の低減をしつつ，どの程度の計算性能が実現できるかを検証した．

2. 高位合成

高位合成は HDL を用いて回路を生成するのではなく，C/C++などの普通のプログラミング言語を用いてアルゴリズムを記述し，回路を生成する技術である．高位合成の開発環境は FPGA のベンダーなどから提供されており，本研究においては Xilinx 社製 Vivado HLS を使用した．

3. 回路設計

3.1 アドレス計算

畳み込み演算や MaxPool などの空間フィルタリングの操作は多重ループを用いて記述されることが多い．このため，並列化を行う際にはループ展開を行うことが多い．

ループ展開による並列化について，その並列数は開いたループのカウント上限に依存する．つまり任意の並列数を実現することは困難であり，リソースの使用率や目標速度に応じた並列数を設定することは難しい．

本研究では単一のループ内で複数のカウンタを動作させることで各スレッドに渡すデータのアドレスを計算するアルゴリズムを採用した．具体的には 0 番のスレッドに渡すデータのアドレス(先頭アドレス)を計算し，先頭アドレスからのずれを計算することで別のスレッドのアドレスを計算する．このアルゴリズムにより，並列数はリソースの範囲内であれば任意の値で設定可能になった．

3.2 データ構造に応じた並列化

データのアクセス順番はアドレス計算と密接にかかわっており，アドレスの計算にかかるコストがなるべく小さくなるように順番を決定するべきである．本実験で使用しているアドレス計算の手法は，入力と出力のうちシーケンシャルなアドレスを発行できる方を基準として，基準のアドレスからもう一方のアドレスを計算する方式をとった．

MaxPooling を例に考えると，出力のアドレスを最初に計算しそこから対応する入力アドレスの先頭部分を計算する．その後計算ステップごとに適当なだけ入力側のアドレスを加算することでカーネル内すべての値を参照することができる．この方法は出力のアドレスをシーケンシャルに発行できるので，配列に対するアクセスのペナルティを最小限に抑えられる．

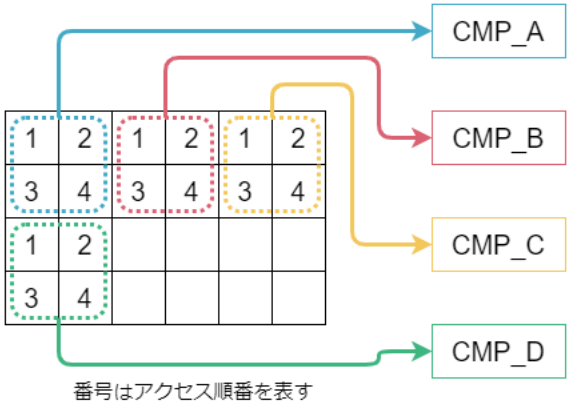

図 1 MaxPooling でのアクセス方法

この計算方法に合わせる形でデータ計算部を設計した．基本的に同種の計算回路を並列数分並べた構成で，SIMD やベクトルプロセッサの演算部と似ている．アドレスの順番の関係で，単一の計算ステップでは入力データ群は互いの計算結果に影響しないことから，一つの計算回路が一つのカーネルの計算を受け持つことになる．

4. 変換ツールの概要

本研究で作成した支援ツールは，学習済みの深層学習モデルから高位合成用に最適化された C++コードを生成する．具体的には，Tensoflow/Keras で作成されたモデルを解析し，メモリや演算器などの C++コードを生成する．この C++コードを高位合成コンパイラで回路に変換し FPGA に実装した．

4.1 モデル解析

モデルの解析は回路の構成や制御手順を確定するために行う．計算の依存関係を最初に解析し，どの順番で演算器を動かすかやメモリの必要な領域をどの計算ステップまで確保するか，メモリサイズの決定などを行う．

4.2 コード生成

モデルの解析結果に基づき，C++コードを生成する．モデルの解析結果に依存するコード部分はそこまで多くはな

† 東京理科大学先進工学研究科電子システム工学専攻
Tokyo University of Science, Graduate School of Advance Engineering, Department of Applied Electronics

いので，解析結果に依存する部分を穴あきにしたコードのひな型を準備し，必要な部分のコードを自動的に生成し組み合わせた．同時に Vivado HLS 用の最適化指示子を付与する．

5. 実験と結果

本研究では図 hoge のような SeparableConvolution を用いた DeepAutoEncoder を Tensorflow+Keras を用いて作成し，それを今回提案したツールを用いて C++コードに変換した．

また変換したコードを Raspberry Pi 4(RPi4)や FPGA へ実装した際の実行速度を計測した．

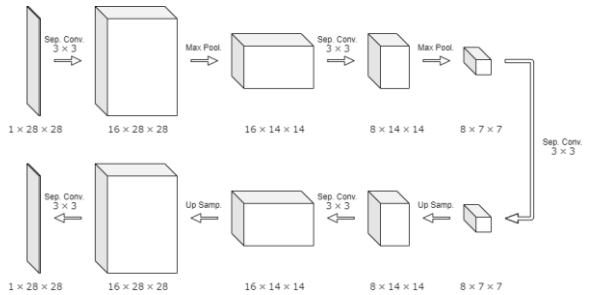

図 2 対象となる DeepAutoEncoder

5.1 実行環境

表 1，表 2，表 3 に本実験で使用した開発環境，FPGA のスペック，RPi4 のスペックをそれぞれ示す[4][5].

表 1 開発環境

CPU	Intel Corei5-6500
RAM	DDR4 32GB
OS	Ubuntu 20.04 LTS
FPGA 開発環境	Xilinx Vivado v2020.1
高位合成環境	Xilinx Vivado HLS v2020.1
C++	C++0x(C++11)
深層学習ライブラリ	Tensorflow 2.0

表 2 FPGA のスペック

ボード	PYNQ-Z2
チップ	Xilinx XC7Z020CLG400-1
LUT	53200
FF	106400
DSP	220
DRAM	4.9Mbits

表 3 RPi4 のスペック

ボード	Raspberry Pi 4 4GB
CPU	Broadcom BCM2711
CPU Clock	1.5GHz
RAM	4GB
OS	Raspbian 10
Python	3.7
C++	C++0x(C++11)
GCC	8.3.0
Temsorflow	2.0

5.2 実行結果

表 4 に生成した回路を FPGA に実装した際のリソースの使用率を示す．

表 4 リソースの使用率

	使用率
LUT	69%
FF	37%
DSP	50%
DRAM	56%

次に生成した C++コードや元の Python を RPi4 で実行した際の速度と FPGA で実行した際の速度を比較した．

表 5 にその結果を示す．

表 5 実行速度の比較

デバイス	実行環境	実行速度[ms]	高速化比
RPi4	Python	121.236	1
RPi4	C++	0.025	4849.44
FPGA	(Hardware)	35.357	3.42

この結果から，Python で実行するよりもコードを C++に変換し実行したり， FPGA に実装した方が高速に動作することが分かった．また RPi4+C++の方が FPGA より約 1400 倍高速に動作していることが分かった．

5.3 まとめと今後の課題

Python で実行した場合と比較して，どちらのデバイスで実行した場合でも高速化できていることが確認できた．しかし変換したコードを RPi4 で実行した方が，FPGA よりも高速に動作することも分かった．この原因として，アドレス計算部の動作速度や動作周波数の低さが挙げられる．特にアドレス計算部分には除算が含まれており，除算を含まない形でアドレスを計算することでさらに高速化できると考えている．今後の展望として，まずはアドレス計算の除算の置き換えを行うことで，アドレス計算の高速化を図りたいと考えている．

参考文献

[1] S. Gandhare and B. Karthikeyan, "Survey on FPGA Architecture and Recent Applications," 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), 2019, pp.1-4, doi: 10.1109/ViTECoN.2019.8899550.

[2] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu and S. Zhang, "Understanding Performance Differences of FPGAs and GPUs," 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2018, pp.93-96, doi: 10.1109/FCCM.2018.00023.

[3] A. Milton, D. Kearney, S. Wong and S. Lemmo, "Development productivity in implementing a complex heterogeneous computing application," 2014 International Conference on Field-Programmable Technology (FPT), 2014, pp. 322-325, doi: 10.1109/FPT.2014.7082809.

[4] Xilinx.inc, “Zynq-7000 SoC Data Sheet: Overview(DS190)”, https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, 2021 年 6 月 13 日閲覧.

[5] Raspberry Pi Foundation, “Raspberry Pi 4 Model B specifications – Raspberry Pi”, https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/, 2021 年 6 月 13 日閲覧.