C-008

エッジ端末による推論処理に必要なデータ転送最適化手段の検討
Investigation of data transfer optimization methods necessary for inference processing by edge terminals

岩本 征弥† 中西 知嘉子‡
Seiya Iwamoto Chikako Nakanishi

1. はじめに

近年，深層学習などの AI 技術が飛躍的に発展している中で，通信遅延・通信コストやセキュリティの観点から，ネットワークを用いず，手元のエッジ端末上で AI 技術を動作させる「エッジ AI」へのニーズが高まっている．

エッジ端末で AI による推論処理を実装する為には，一般に，AI 技術は計算量が大きく，CPU 上で動作するには要求性能が非常に高くなるという問題点を克服しなければならない．GPU を用いることも可能だが，コストや消費電力の問題から，低コスト・低リソース環境下では採用できない．そこで，専用の回路を使用し，低リソースで推論処理を行える「エッジ AI」を実現させる．回路を用いる際，深層学習におけるネットワーク構造を量子化して演算量を削減する手法が広く研究されている．これらの手法では演算量を削減し，処理速度を上げられる反面，推論精度が低下してしまう点が問題となる．また，急速な発展が進んでいる AI 技術の発展速度に合わせる必要もあるため，開発コストや開発期間を抑えられず，仕様変更を柔軟に対応出来ないという問題がある．そこで，本研究ではネットワーク構造をそのまま使う事で精度を維持し，SoC FPGA を採用して開発の柔軟性を保ち，回路とソフトを協調動作させて推論を高速化する手法を提案する．

SoC FPGA で推論処理を行うにあたって，基本的な回路を設計したが，回路部とソフト部間のデータ転送に伴う共有メモリへの読み書きにかかる処理時間に改善の余地があることが分かった[1][2]．そこで，提案している手法に最適なデータ転送手法の再考を行う．また，そのほかにも推論処理時間を短縮させるための考察も行う．

2. 使用機器・使用ネットワーク

本研究では，特に画像処理などで広く取り扱われている，畳み込みニューラルネットワーク(CNN)を取り上げ，その中で EfficientNet を用いて設計する．

2.1 EfficientNet

EfficientNet は，2019 年に Google 社発表の高速かつ高精度な深層学習モデルである[3]．組込み機器への搭載を視野に入れる為，今回は最も小さいモデルである B0 を対象としている．

EfficientNet のアーキテクチャは，SE モジュールを含めた MBConv と呼ばれるブロックを用いて構成されている．提案手法では，このブロックを含めて層単位まで分解し，その中でも特に処理時間がかかる箇所を回路部で処理させ，それ以外のすべての処理をソフト部上で処理させることによって高速化している．回路部での処理は様々な形状の層を回路で実現する為，ソフト・ハード協調設計技術により実現させる．

はじめに，CNN の層毎に推論過程を分析し，本研究で用いる SoC FPGA ボードである Ultra96v2 の CPU 上のみで処理させた場合の演算時間の大きな処理と繰り返し実施される処理を調べた．表 2.1.1 に各層の処理時間と割合を示す．ここで，全体の処理時間は約 23 秒ほどである．

表 2.1.1 EfficientNetB0 の各層の処理時間と割合

層の種類	割合(%)
Conv2D	90.4
Activation	4.5
DepthwiseConv2D	2.5
BatchNormalization	2.0
Otherwise	0.6

表 2.1.1 から，畳み込み層(Convolution 層)が処理の 9 割以上を占めていることが分かる．そこで，Convolution 層の中から特に Convolution2D 層(Conv2D 層)に特化した回路を設計する．次に，回路化を行う箇所の詳細を分析した．Conv2D 層で畳み込みに使用するフィルタサイズを分析した結果を表 2.1.2 に示す．

表 2.1.2 Conv2D 層の詳細分析結果

フィルタサイズ	層数	処理時間(ms)
1x1	64	21305
3x3	1	179

表 2.1.2 より，Conv2D 層のうち，入力層である 1 層を覗いたすべての層でフィルタサイズが 1 の畳み込み演算が行われていることが分かる．

以上より，フィルタサイズ 1 の Conv2D 層を回路化対象とし，それ以外の処理をソフト部で処理させる．

2.2 Ultra96v2

回路部とソフト部の協調動作を実現させるために，Xilinx 社の SoC FPGA を搭載した評価基盤を用いる．

本研究では Ultra96v2 を用いた[4]．表 2.2.1 に Ultra96v2 の性能を示す．

表 2.2.1 Ultra96v2 の性能

CPU	Application Processing Unit	Quad-core Arm Cortex-A53 MPCore with CoreSight; 32KB/32KB L1 Cache, 1MB L2 Cache		
	Real-Time Processing Unit	Dual-core Arm Cortex-R5F with CoreSight; 32KB/32KB L1 Cache, and TCM		
Embedded and External Memory		256KB On-Chip Memory w/ECC; External DDR4; DDR3; DDR3L; LPDDR4; LPDDR3; External Quad-SPI; NAND; eMMC		
RAM		Micron 2GB LPDDR4 Memory		
Programable Logic	System Logic Cells	154,350	Distributed RAM	1.8 Mb
	CLB Flip-Flops	141,120	Block RAM Blocks	216
	CLB LUTs	70,560	Block RAM	7.6 Mb
	DSP Slices			360

3. 実装手法

3.1 回路部

3.1.1 高位合成による回路生成

回路設計は C++言語を用いて処理実行のアルゴリズムを設計し，RTL 言語へ高位合成を行う事で回路化した．この方法を用いることで，演算のアルゴリズムを明確に定義でき，また必要に応じて柔軟に回路を変更できることから，設計の効率向上につなげることが可能となる[5]．また，データのやり取り手法について高位合成前に定義できるため，回路実装後の CPU 側との協調動作を容易に実装できる．

高位合成には Xilinx 社の Vivado HLS 2019.2，回路生成には同社の Vivado 2019.2 を用いた．

3.1.2 DMA 転送によるデータ転送

データ転送には DMA 転送(Direct Memory Access)を用いて共有メモリから演算回路へデータを転送する．DMA のモードには，Simple DMA(Direct Register Mode)と Scatter Gather DMA(Scatter Gather Mode)が使用できる．Simple DMA は連続したメモリ領域の開始アドレスとその長さを設定して転送する方式である．Scatter Gather DMA は小さい連続したメモリ領域のデータをまとめて転送する転送方式である．この転送方式は高速なデータ転送を実現できるが，インターフェースが複雑となり，使用リソースが増加する要因となる．今回はデータ転送を Simple DMA を用いてデータ転送を行う．Scatter Gather DMA に比べると低いが，シンプルなデータ管理が可能となる．

3.1.3 キャッシュ管理

デフォルトではソフト部側でキャッシュの管理が行われている．この場合，CPU はキャッシュを通じてメモリ上の DMA バッファや UIO にアクセスするが，回路側では直接メモリ上の DMA バッファや UIO にアクセスする．ここで，キャッシュコヒーレンシの問題が発生する．キャッシュコヒーレンシとは，共有メモリなどの共有リソースに対し，キャッシュとメモリの値が同じであるかという一貫性の事を指す．ソフト部側でキャッシュ管理を行った場合，キャッシュとメモリ間の同期処理が必要となり，処理時間が長くなる要因となる．

そこで，回路部がデータの一貫性を保証しているという前提で，キャッシュとメモリ間の同期処理を省略することを目的として，キャッシュの管理を回路部で行う．これにより，共有メモリからソフト部への読み出し時間の削減を図る．

3.2 ソフト部

3.2.1 User Space I/O(UIO)

Ultra96v2 ボード上の Linux ユーザ空間から回路部の制御を行う．回路部と DMA 転送制御インターフェースに User Space I/O (UIO)を使用する．UIO とは，ユーザ空間でデバイスドライバを作成するための仕組みであり，指定した範囲の物理メモリをユーザ空間からマッピング可能にする[6]．今回は DMA 転送制御インターフェース部分をメモリにマッピングし，ユーザ空間からアクセスできるようにした．

3.2.2 udmabuf

DMA 転送に Simple DMA を採用する為，連続したメモリ領域を作成する必要がある．Linux のユーザ空間のデータを DMA 転送するために，Linux のカーネル空間に連続したメモリ領域を DMA バッファとして確保するデバイスドライバである User Space Mappable DMA Buffer (udmabuf)を用いて，回路部との共有メモリとして割り当てる[7]．

3.2.3 Ceras

ソフトウェア側の処理には本研究室で開発した Ceras と呼ばれる AI 推論ツールを用いた．Ceras(C++ edge rapid ai simulator)とは C++言語上で Keras の学習済みモデルから推論を行う Keras2cpp[8]をベースに開発された推論処理実行ツールである[9][10][11]．このツールは C++言語の標準ライブラリのみを用いて製作されている．標準ライブラリのみで動作するため，エッジ端末などで動作させる場合に環境構築の難易度が低くなる．

4. 提案手法

4.1 作成回路

回路部では，共有メモリから DMA 転送によって転送されたデータのうち，1 要素と 1 フィルタを乗算し，全チャネル分を加算する積和演算回路を設計した．図 4.1 に処理の流れを示す．

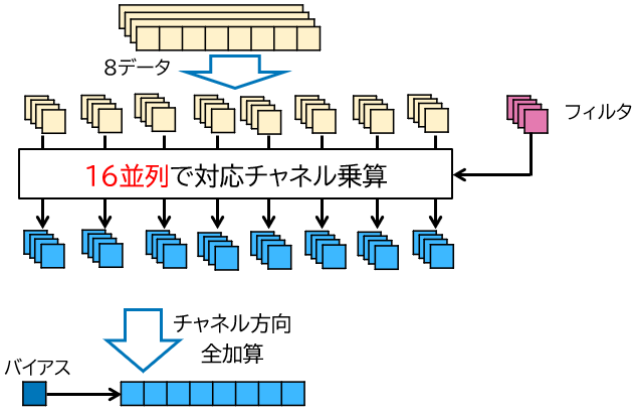

図 4.1.1　作成回路の流れ図

ここでは，すべてのデータを一括で処理するのではなく，8 データ全チャネル分ずつ順に処理する設計とした．これは回路内のメモリと並列で演算するための演算リソースの使用比率を検討して決定した．1 データあたりの処理として，全チャネル分のデータを 16 個のメモリを割り当て，乗算をチャネル方向に 16 並列で行う．これを 8 データ分並列に処理する設計とする．表 4.1.1 に Vivado 上で見積もった Ultra96v2 上のリソース使用率を示す．

表 4.1.1　作成回路のリソース使用率

リソース	割合(%)
LUT	41
FF	23
BRAM	99
DSP	16

4.2 ソフト部の実装

ソフト部側では，回路演算を実行させる層の実行時に，共有メモリへ演算に必要なフィルタ，バイアス，入力データをすべて書き込む．図 4.2.1 に共有メモリから回路部へデータを転送する例を示す．

DMA 転送時は共有メモリ内の転送先頭アドレスと転送サイズを指定する．ここでは，フィルタ，バイアス，入力データそれぞれの先頭アドレスと転送サイズを指定する．指定した先頭アドレスから転送サイズ分を回路部へ転送し，その後に回路を実行する．また，回路部の演算結果が共有メモリへ書き込まれるため，データを読み出したのちに適切に整形を行い次の層へ送る．

また，特定の層によっては，演算に使用するフィルタが回路部のメモリに入りきらない場合がある．その場合は，ソフト部側でフィルタ分割を行う．分割の際は共有メモリに書き込んだフィルタおよびバイアスの先頭アドレスと分割するサイズを指定して転送する．

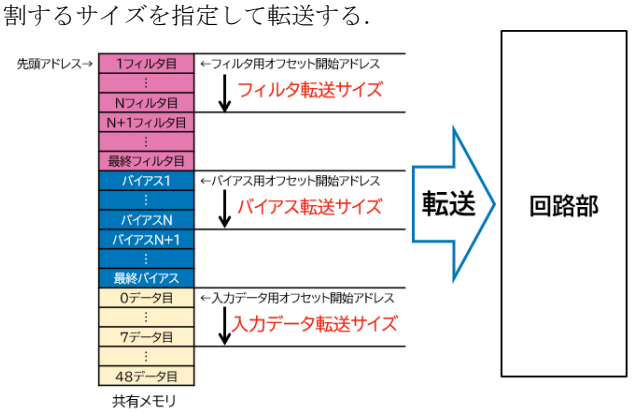

図 4.2.1　共有メモリの扱い例

5. 評価方法と検証

評価方法として，データセット ImageNet で学習済みのモデルを用いて，ソフト部のみで実行した処理時間と提案手法を用いた処理時間を比較する．Ceras を用いて推論を行うにあたって，提案手法は精度が同一であることを確認したため，精度の評価は行わない．

図 5.1 にソフト部のみで実行した処理時間と提案手法の処理時間を示す．この図より，EfficientNet 全体においては約 6.1 倍，高速化対象の Conv2D 層においては約 14.3 倍へ処理の高速化を実現できたことが分かる．

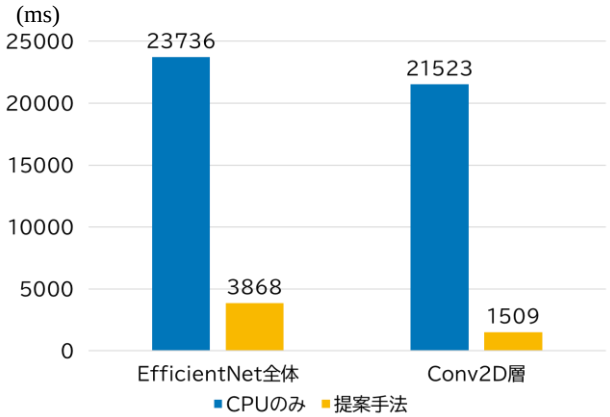

図 5.1　提案手法の処理時間

さらなる高速化を図るために高速化が見込める処理を調査する．表 5.1 に回路化を行ったフィルタサイズ 1 の Conv2D 層の詳細な処理時間を示す．この表より，回路演算時間のほかに，共有メモリの書き込み時間が 3 割程あることがわかる．次にデータ転送にかかる処理時間の削減を行う．

表 5.1　改良設計の Conv2D 層の処理時間

	処理時間(ms)	割合(%)
共有メモリ書込時間	429.534	31.2
共有メモリ読出時間	131.817	9.5
回路演算時間	591.64	42.9
その他の処理時間	223.63	16.2
Conv2D 層総演算時間	1376.63	

6. データ転送処理時間の高速化の検討

6.1 フィルタとバイアスの書き込み時間の隠蔽

共有メモリの書き込みにかかる時間を削減するため，共有メモリに書き込むデータの総数を調査した．表 6.1.1 に転送対象のデータ数と割合を示す．この表より，入力データのほかに，フィルタの転送量が全体の 6 割弱程あることが分かる．1 データ当たり 4 バイトの浮動小数値であることから，すべてのバイアスとフィルタの総データ量は 14.43MB である．このデータを共有メモリへ書き込む時間を隠蔽する．

表 6.1.1　転送対象データ数と割合

転送対象	データ数	割合(%)
バイアス	9340	0.1
フィルタ	3773312	56
入力データ	2896028	43
合計	6678680	100

推論ツールとして用いている Ceras では，プログラム実行時に全ての重み情報を読み込み，Linux ユーザ空間上のメモリに格納している．その後，推論を行う過程で必要なデータを共有メモリへ読み書きすることを通じて回路部への協調動作を実現している．

ここで，Ceras の推論処理時間とは関係のない箇所を利用する．プログラム実行直後の重み情報を読み込む際に，フィルタサイズ 1 の Conv2D 層のすべてのフィルタとバイアス情報を共有メモリへ書き込む．この時，共有メモリとして割り当てるサイズは入力サイズを含めた大きさにする必要がある．今回は回路部へデータを転送するメモリ 100MB と，回路部から返されたデータを格納する共有メモリ 100MB を共有メモリとして割り当てた．DMA 転送時の制御方法は同様に，転送対象の共有メモリの先頭アドレスと転送サイズを設定して転送する．

図 6.1.1 にフィルタとバイアスを共有メモリに書き込む処理を隠蔽した場合の処理時間を示す．フィルタとバイアスを書き込む処理時間が 300ms 程度減少したことがわかる．

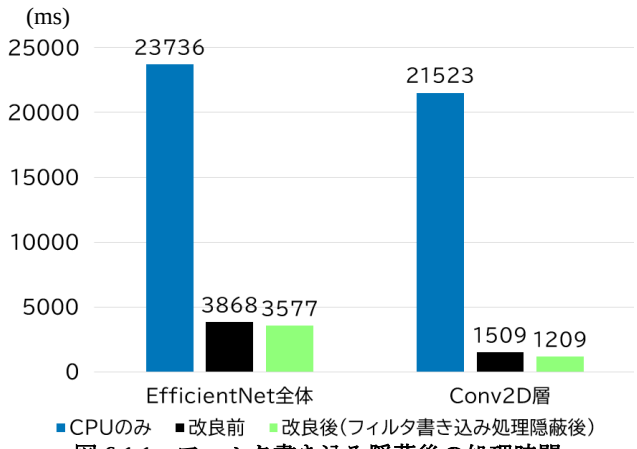

図 6.1.1　フィルタ書き込み隠蔽後の処理時間

6.2 Activation 層の見直し

Conv2D 層の次に処理時間のかかる Activation 層にも触れる．表 6.2.1 に EfficientNet 全体，Conv2D 層，Activation 層それぞれの処理時間と割合を示す．Conv2D 層の処理時間が大きく減らせたため，相対的に Activation 層の処理時間の割合が増えている．

表 6.2.1　EfficientNetB0 の各層の処理時間と割合

層の種類	処理時間(ms)	割合(%)
EfficientNet 全体	3868	100
Conv2D	1209	31
Activation	735	19

Activation 層の処理の詳細を調査した結果，すべての層で Swish 関数が適用されていることが分かる．Swish 関数は式(1)で示される非線形関数である[12].

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

Swish 関数は非線形関数であるため，計算量が大きく演算時間がかかる．そこで，この関数を線形関数として式(2)に示す HardSwish 関数に近似する[13].

$$y = \begin{cases} 0 & (x \le -3) \\ \dfrac{x(x+3)}{6} & (-3 \le x < 3) \\ x & (3 \le x) \end{cases} \tag{2}$$

HardSwish 関数に置き換えた結果を表 6.2.2 に示す．ここでは，学習済みモデルの再学習を行わない．Swish 関数から置き換えることで CPU 上でも処理時間を半分程度削減することができることが分かる．回路へ組み込むことでさらなる処理時間の削減を見込める．また，精度は若干変動するが，大きな変化がないことを確認している．

表 6.2.2　EfficientNetB0 の各層の処理時間と割合

活性化関数	処理時間(ms)
Swish	735
HardSwish	380

7. 結論

本研究では，SoC FPGA を用いて回路部とソフト部を協調動作させ，畳み込みニューラルネットワークである EfficientNetB0 の演算処理の高速化を図った．ネットワーク構造に注目し，処理時間のかかる箇所を調査し，回路を実装した．回路演算にかかる時間のほかに，データ転送にかかる処理時間が大きいことが分かった．データ転送手法を見直すことでさらなる処理時間の短縮を行えることがわかった．また，Conv2D 層以外の層においても，処理時間の短縮を見込めることがわかった．

今後の展望として，Activation 層で HardSwish 関数に置き換えた処理の回路への組み込み，回路部の転送バス幅の変更による回路部側から見た共有メモリの読み書き時間の削減，さらなるデータ転送の最適化手法の検討などが挙げられる．

また，類似手法を EfficientNet 以外のネットワークにも適応したうえで，高速化を図れるかを検証したい．

参考文献

[1] 岩本征弥, 中西知嘉子, "深層学習による推論処理の高速化手法の検討", 信学技報, vol. 122, no. 60, RECONF2022-13, pp. 52-56 (2022)
[2] 川上智也, 中西知嘉子, "ハードウェアとソフトウェアの協調動作による AI 推論処理の高速化の検討", 信学技報, vol. 122, no. 60, RECONF2022-14, pp. 57-62 (2022)
[3] Tan, Mingxing,and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv preprint arXiv:1905.11946 (2019)
[4] https://japan.xilinx.com/products/boards-and-kits/1-vad4rl.html
[5] Vivado Design Suite ユーザーガイド 高位合成, https://docs.xilinx.com/v/u/2019.2-Japanese/ug902-vivado-high-level-synthesis
[6] Katsuya MATSUBARA, Hisao MUNAKATA,"Using UIO in an embedded platform", https://elinux.org/images/b/b0/Uio080417celfelc08.pdf (2008).
[7] https://github.com/ikwzm/udmabuf/blob/master/Readme.ja.md
[8] https://github.com/gosha20777/keras2cpp
[9] 大戸彰馬, 中西知嘉子, "推論処理における畳み込み処理の回路化の検討",電子情報通信学会総合大会(2022).
[10] 西岡駿, 中西知嘉子, "機械学習ライブラリの C 言語化の実現", 電子情報通信学会ソサイエティ大会(2021).
[11] 西岡駿, 中西知嘉子, 那須督, "エッジ AI の実現手法の検討", 電子情報通信学会総合大会(2022).
[12] Prajit Ramachandran, Barret Zoph, Quoc V. Le, Google Brain, "SWISH: A SELF-GATED ACTIVATION FUNCTION" arXiv(2017)
[13] https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html

† 大阪工業大学 情報科学研究科 情報科学専攻
Graduate School of Information Science and Technology Osaka Institute of Technology
‡ 大阪工業大学 情報科学部 情報知能学科　Department of Information and Computer Science Osaka Institute of Technology