C-009

SoC FPGA における推論処理の高速化に最適なデータ処理の検討
Optimal Data Processing Methods for Accelerating Interface Processing in SoC FPGAs

川上 智也‡　　中西 知嘉子‡
Tomoya Kawakami　Chikako Nakanishi

1. はじめに

近年，AI 技術が飛躍的に進化しており，様々な分野において活用されるようになっている．AI 技術をエッジ端末上で動作させる「エッジ AI」は，通信コストが削減できることや通信遅延が発生しないこと，セキュリティを強化できるといった利点があり，特に注目されている．

しかし AI は演算量が膨大であるため，エッジ端末上で高速に推論処理を行うことが難しい．そこで我々は，CPU と FPGA が同一チップ上にある SoC FPGA を採用し，ソフトとハード(回路)を協調動作させることで高速化する方法を検討している．画像認識 AI モデルの一つである EfficientNet[1]を対象に，畳み込み処理を回路で実行し，残りの全ての処理はソフトで実行する方法で高速化を試みた．その結果，ソフトによるデータ転送に改善の余地があることが分かった[2][3].

さらなる高速化のために，データ転送の最適な手法を考案した．本発表では，さらなる高速化のためのデータ転送方法を提案する．

2. 開発環境・使用ネットワーク

2.1 環境

ここでは設計した回路および本研究で使用した機器と開発環境について述べる．回路は C++言語を用いて作成し，Vivado HLS の高位合成[4]によって回路を生成し，Vivado を用いて回路を設計した．設計した回路は Xilinx 社の SoC FPGA ボードである Ultra96-V2[5]に実装した．

2.2 EfficientNet

回路を設計するために，使用する AI である EfficientNet の中でも最も小さいモデルである B0 について分析した．Ultra96-V2 の CPU のみで推論処理を行った際の処理時間を表 2.2.1 に示す．

表 2.2.1　EfficientNet の分析結果

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

全体の処理時間は約 23 秒ほどであり，その 9 割を Conv2D 層という畳み込み処理を行う層が占めていることが分かった．そこで，畳み込み処理をアクセラレートする回路を設計することとした．次に回路を設計するために Conv2D 層の詳細を分析した．表 2.2.2 に分析結果を示す．

表 2.2.2　Conv2D 層の分析結果

入力サイズ	入力チャネル	カーネルサイズ	層数	処理時間 (ms)
7 の倍数	8 の倍数	1×1	30	20735
1	8 の倍数	1×1	34	570
7 の倍数	1	3×3	1	179

表 2.2.2 より，Conv2D 層 65 層のうち 30 層を占める層の処理時間が約 97%を占めていることが分かった．そのため，入力サイズが 7 の倍数，カーネルサイズが 1×1 の Conv2D 層を回路化対象とした．また，カーネルサイズが同じである入力サイズ 1 の場合についても回路で処理できるように設計した．

3. 設計

3.1 回路部

3.1.1 回路内で扱うデータ

Conv2D 層では入力データに対し複数のカーネルデータを用いて畳み込み処理を行っている．そこで回路内で入力データを保持することで，入力データの転送回数削減を図った．保持するデータ量について，実装する FPGA リソース量を考慮し表 3.1.1 のように設定した．なお，カーネルについては膨大な量であるため，1 回の畳み込みで使用するカーネルのみを回路へ転送するように設計している．後述するがデータは DMA 転送によって転送している．

表 3.1.1　回路内で保持するデータ量

	保持するデータ量
入力サイズ	7×7
入力チャネル	全チャネル
バイアス	全て

3.1.2 作成回路

回路内で保持する入力サイズから，回路部では 49 並列で演算を行う回路を設計することにした．高位合成を用い 49 並列で回路の作成を試みたが，7 の倍数の制御のため，無駄な処理が挿入されてしまい，高速な回路が生成できなかった．そのため，64 並列で演算できる回路を設計した．処理の流れを図 3.1.2 に示す．

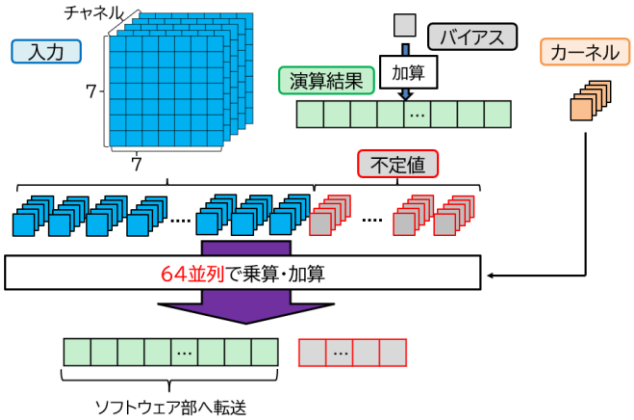

図 3.2.1　並列演算部

回路では，入力データを 49 個受け取ったのち不定値 15 個で拡大する処理を行っている．これを入力チャネル回繰り返すことで，畳み込み処理を 64 並列で実現している．

3.2 ソフトウェア部

3.2.1 サイズ分割

回路化対象の Conv2D 層では，入力サイズが 14×14 以上の場合がある．その場合は，ソフトウェアで 7×7 へのサイズ分割を行っている．処理の流れを図 3.2.1 に示す．

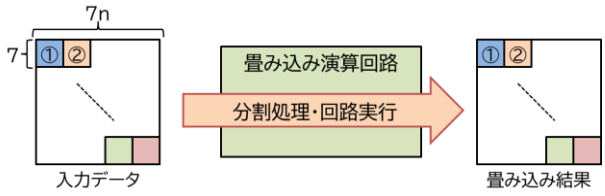

図 3.2.1　サイズ分割処理

3.2.2 パディング処理

入力サイズが 1×1 の場合についても回路で処理するために，ソフトウェア上で 0 埋めによるパディング処理によって 7×7 へ拡大させている．処理の流れを図 3.2.2 に示す．

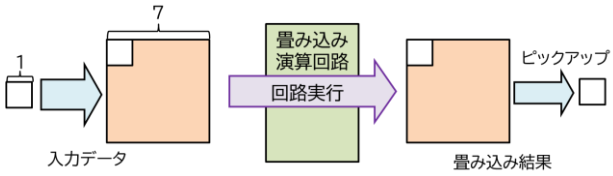

図 3.2.2　パディングによるサイズ拡大

この場合，演算結果も拡大された状態で得られるがソフトウェア上で取り出す演算結果の選択を行っている．

4. 実装

4.1 DMA 転送

データ転送には DMA 転送(Direct Memory Access)を用いて，共有メモリから回路部へデータを転送している．DMA 転送には Simple DMA(Direct Resister Mode)と Scatter Gather DMA(Scatter Gather Mode)が選択できる．Simple DMA は連続するメモリ領域の開始アドレスと長さを設定し転送を行う方式であり，Scatter Gather DMA は不連続なメモリ領域のデータをまとめて転送する方式である．Scatter Gather DMA は高速なデータ転送を実現できるが，設計が複雑になり，使用リソースが増加してしまう．そのためメモリ管理が容易な Simple DMA によって転送を行う．

4.2 キャッシュ管理

共有メモリを使用したデータ転送を行う場合，キャッシュとメモリの値が同じであるかの一貫性の問題が発生する．ソフトウェア側でキャッシュの管理を行った場合，キャッシュとメモリ間の同期処理が必要となり，処理時間が長くなってしまう．そこで回路部でキャッシュの管理を行うように変更することでキャッシュ管理にかかる処理時間の削減を図る．

4.3 udmabuf

Simple DMA を使用するためには，DMA バッファとして連続するメモリ領域を用意する必要がある．そのために，User Space Mappable DMA Buffer(udmabuf)[6]を使用する．udmabuf とは，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し，ユーザ空間から mmap によってアクセス可能にするデバイスドライバである．今回，作成した回路や DMA 転送回路内のレジスタへのアクセスは，User Space I/O(UIO)[7]を使用した．

4.4 Ceras

ソフトウェア部の処理には本研究室で開発した AI 推論処理ツールである Ceras 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ツールである[2][3][9][10][11]．このツールは C++言語の標準ライブラリのみで動作するため，ソフトウェア部の設計が容易であり，エッジ端末に実装する際の環境構築の難易度が低くなる．

5. 処理時間による評価

EfficientNet の高速化のために設計した，Conv2D 層をアクセラレートする回路について，EfficientNet と Conv2D 層にかかる処理時間の測定結果を図 2.3.1 に示す．

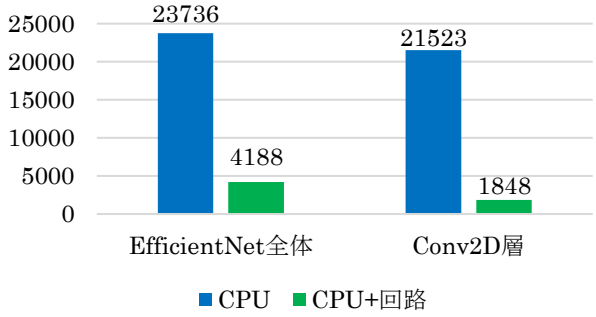

図 5.1　処理時間測定結果

図 5.1 より EfficientNet 全体では 23736ms から 4188ms と約 5.7 倍の高速化，Conv2D 層では 21523ms から 1848ms と約 11.6 倍の高速化となった．

次に Conv2D 層の処理時間についての詳細な測定結果を表 5.1 に示す．

表 5.1　Conv2D 層の処理時間詳細

	処理時間(ms)	割合(%)
共有メモリへの書き込み	549	30
共有メモリからの読み出し	127	7
回路演算	812	44
その他の処理	360	19
合計	1848	

表 5.1 より，回路演算の他にソフトウェアが回路へ転送するデータを共有メモリに書き込む処理に時間がかかっていることが分かった．そこでさらなる高速化のために，共有メモリへの書き込み時間の高速化方法を考える．

6. 書き込み時間の高速化の検討

6.1 書き込まれるデータ数

共有メモリへの書き込み時間の高速化のため，推論処理中の Conv2D 層で共有メモリに書き込まれるデータ数について調査した．表 6.1.1 に調査結果を示す．

表 6.1.1　書き込みデータ数の調査結果

転送データ	データ数	書き込み数	増加数
入力データ	2,896,028	3,335,008	438,980
バイアス	9,340	9,340	0
カーネル	3,773,312	7,536,128	3,762,816
合計	6,678,680	10,880,476	4,201,796

表 6.1.1 について，データ数は全 Conv2D 層で使用する各データ数を表しており，書き込み数は回路を動作させるために共有メモリに書き込んだ各データ数を表している．増加数は書き込み数とデータ数の差である．これより，回路を動作させた際，入力データとカーネルが余分に共有メモリに書き込まれていることが分かった．次にこの原因について調査する．

6.2 書き込みデータ数増加の原因

6.2.1 入力データ

入力データの書き込み数増加は，入力サイズ 1×1 時のパディング処理が原因である．0 埋めで拡大した分も含めて共有メモリに書き込んでいるため，書き込み数が増加した．そのため，パディング処理を行わず回路を動作できれば入力データの書き込み数をデータ数まで削減できる．

6.2.2 カーネル

カーネルの書き込み数増加は，入力サイズが 14×14 以上の時に発生していることが分かった．サイズ分割によって入力データを 4 つに分割した例を用いて，図 6.2.2-1，6.2.2-2 に共有メモリへの書き込みの流れを示す．

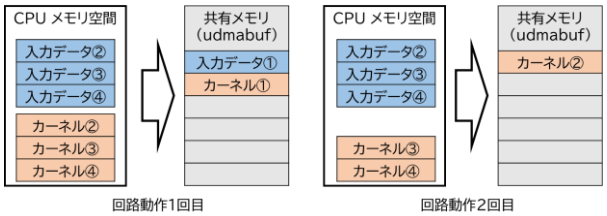

図 6.2.2-1　共有メモリへの書き込み例 1

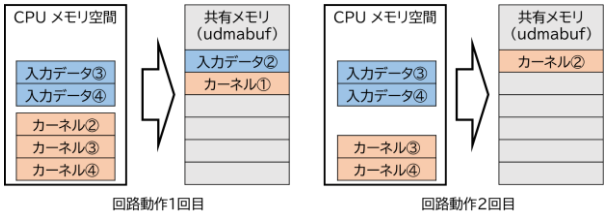

図 6.2.2-2　共有メモリへの書き込み例 2

図 6.2.2-1 は 1 つ目の入力データで演算を行う際の共有メモリへの書き込みを表している．1 回目の回路動作では，入力データ①とカーネル①を共有メモリに書き込む．2 回目以降の回路動作では，回路内で入力データ①を保持しているため，カーネル②のみを共有メモリに書き込み，最後のカーネル④を書き込んで演算が行われたら，次に入力データ②の処理に入る．この時，図 6.2.2-2 のように再度カーネル①から書き込んでいる．この再書き込みが原因で，カーネルの書き込み数がデータ数の 2 倍ほどに増加している．

バイアスを使用するのは入力サイズ 1×1 の場合のみであるため，再書き込みは起こらない．

6.3 全カーネル・バイアスの書き込み

上記の共有メモリへの書き込み時間の短縮を図るため，下記の手法をとることにした．

① 全カーネル・バイアス情報を共有メモリに保持する

② 入力サイズ 1×1 の場合に対応可能な回路を作成する

①については，6.3 節，②については 6.4 節で説明する．

Ceras を用いて推論処理を行う際，推論処理開始前にすべてのカーネルデータとバイアスを読み込み Linux のユーザ空間上のメモリに格納している．その後，推論処理中に必要なデータを共有メモリに読み書きすることで回路とのインターフェースを実現している．

そこで，①が可能であるかを検討するため，すべてのカーネルデータとバイアスを保持するために必要なデータサイズを調査した．その結果，必要なサイズは，14.43MB であり，udmabuf で割り当て可能な容量以下であることが分かった．そこで，入力データで使用する分も含めて共有メモリを 100MB 割り当てることにした．

共有メモリに全カーネル・バイアスを書き込んだ際の回路部へのデータ転送について図 6.4.1 に示す．

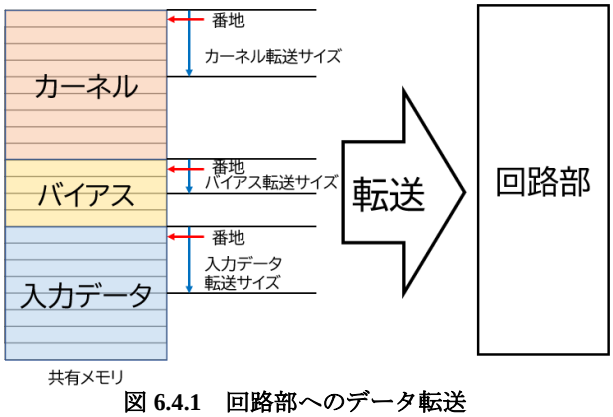

図 6.4.1　回路部へのデータ転送

DMA 転送によって回路部へデータを転送する際，共有メモリにあらかじめ書き込まれたカーネル，バイアス，そして回路動作のため書き込んだ入力データ，それぞれの共有メモリ上での先頭アドレスと転送サイズを指定することで回路部へ必要なデータを転送している．サイズ分割によってカーネルの再転送が発生する場合は，最初に転送したアドレスと転送サイズを再び指定する．

6.4 回路部の改良

入力サイズ 1×1 時の処理による入力データの書き込み数増加を防ぐために回路内での入力データの扱い方を変更した．入力チャネルが 64 の入力データを例に，図 6.5 に回路内での入力データの様子を示す．

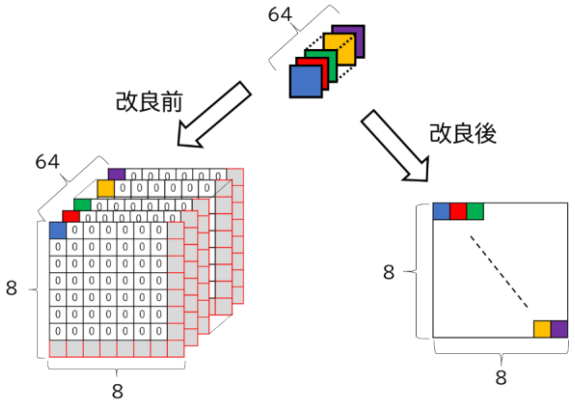

図 6.5　入力サイズ 1×1 時の回路内での扱い方

改良前の回路では，入力サイズが 7×7 の演算回路を使用している．そのため，回路へは 7×7 に拡大した状態で転送され，回路内では 1 チャネルにつき入力データが 1 つとなるように格納している．この例では，64 チャネルで全データを格納することとなり，拡大したデータを含む 1 チャネル 64 個のデータと 1 つのカーネルで 64 並列の演算を行っている．

一方，改良後の回路では，1×1 の状態で転送されるため，回路内では保持するデータサイズの縦横方向に敷き詰めるように入力データを格納している．この例では，1 チャネルで全データを格納することとなり，入力データ 64 個とカーネル 64 個で 64 並列の演算を行っている．

7. 処理時間の再測定と評価

改良を行ったソフトウェアと回路を用いて再度処理時間の測定を行った．図 7.1 に測定結果を示す．カーネルとバイアスを共有メモリに書き込む時間が 500ms 減少したことが分かる．

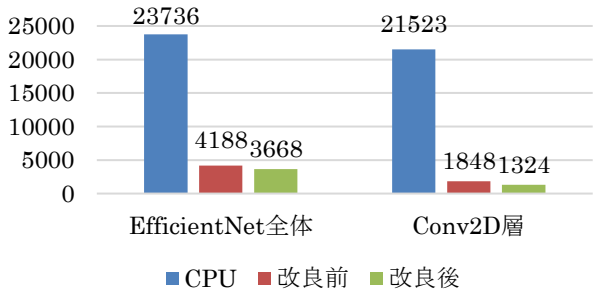

図 7.1　改良後の測定結果

8. 最後に

ソフトウェア部から回路部へのデータ転送を高速化するために，重み情報を読み込む段階でカーネルとバイアスを全て共有メモリに書き込むことで，推論処理中にかかる共有メモリへの書き込み時間の短縮を図った．結果として書き込み時間は大きく短縮された．

今後の展望として，Conv2D 層の次に処理時間がかかっている Activation 層の回路化，転送バス幅の変更による回路部側のデータ転送の高速化などが挙げられる．

また，設計した回路を他のネットワークに適用し，高速化できるかの検証を行いたい．

参考文献

[1] Tan, Mingxing,and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." arXiv preprint arXiv:1905.11946 (2019)
[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] 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)
[4] 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
[5] https://japan.xilinx.com/products/boards-and-kits/1-vad4rl.html
[6] https://github.com/ikwzm/udmabuf/blob/master/Readme.ja.md
[7] Katsuya MATSUBARA, Hisao MUNAKATA,"Using UIO in an embedded platform" https://elinux.org/images/b/b0/Uio080417celfelc08.pdf (2008).
[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).

† 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