Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

高位合成による小規模FPGA向けDNN推論器の設計

岡本 卓也[1,a)]　山本 椋太[1]　本田 晋也[1]　中本 幸一[2]　若林 一敏[3]

概要：近年，Deep Neural Network（DNN）の発展および IoT 技術の普及により，エッジコンピューティングによる推論器の需要が高まってきているが，組込みシステムの厳しいリソース制約に対応する必要がある．汎用計算機の分野では，DNN の各層の処理は GPU を用いて行われる．しかし，推論時においては CPU よりも低いスループットとなる場合がある．また，CPU は逐次処理によって並列演算ができないため，低速となる．そこで本研究では FPGA を用いて，DNN の推論プログラムに対して高速化を行う．C 記述レベルの変更を加えることで，高速化率や FPGA のリソース使用量を計測する．また，小規模 FPGA に対する実装結果も計測する．設計にはシステムレベル設計環境である SystemBuilder を使用し，HW 部には高位合成を適用する．既存の 2 つのプログラムに対して様々な高速化手法を重ねて適用し，それぞれ実装結果を計測する．結果として，最も効率の良い実装で約 51.4 倍スループットが向上した．小規模 FPGA を用いた実装においては，約 2.5 倍スループットが改善された．

A Design for High-Level Synthesis to Configure DNN Inference Circuit on Small Scale FPGA

1. はじめに

近年，Deep Neural Network（DNN）の発展および IoT 技術の普及により，エッジコンピューティングによる推論器の需要が高まってきている．しかし，組込みシステムの厳しいリソース制約を考慮した推論器の実装が求められる．

汎用計算機の分野では，DNN の各層の処理は GPU を用いて行われる．しかし推論時のように 1 つの入力データを対象としてバッチ処理を行わない場合においては，ハイエンドの GPU における実行速度はハイエンドの CPU に劣るという報告もある [1]．また，CPU はシングルコアでは逐次処理によって並列演算ができないため，低速となる．

このような問題から，論理回路を多数組み合わせることにより，エンドユーザが回路を柔軟に何度でも再構築することができる FPGA が使用される傾向にある [2]．

FPGA の回路設計を行うためのレジスタ転送レベル（RTL）記述は専門性が高いため，プログラミング言語から自動で RTL 記述を生成できる高位合成（HLS）が注目されている．HLS により，開発者は効率的に FPGA の回路設計を行うことが可能になった [3]．

本研究では，C ソースコードの DNN の推論プログラムを元に，システムレベル設計環境の SystemBuilder と HLS を用いて FPGA への実装を行う．高速かつ面積の小さい推論器となる HLS のための C 記述を目標とする．

また，面積やメモリの小さい小規模 FPGA に対する実装も検討し，DNN の処理における外部メモリへのアクセスのレイテンシをどの程度隠蔽できるかも検討する．

2. 設計環境

2.1 SystemBuilder

本研究では，SW-HW 間インタフェースの設計抽象度を高め，システムレベルで設計を行うことができるシステムレベル設計環境 SystemBuilder[4] を用いる．本節では，SystemBuilder について概説する．

SystemBuilder の主な機能には以下のものがある．

- デザインの C 言語レベルシミュレーション
- デザインの SW/HW 分割指定後の実装モデル生成
- SW と RTL レベルの HW のコシミュレーション
- デザインの FPGA への実装

[1] 名古屋大学
Nagoya Uniersity
[2] 兵庫県立大学
Uniersity of Hyogo
[3] 日本電気株式会社
NEC Corporation
[a)] okamoto@ertl.jp

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

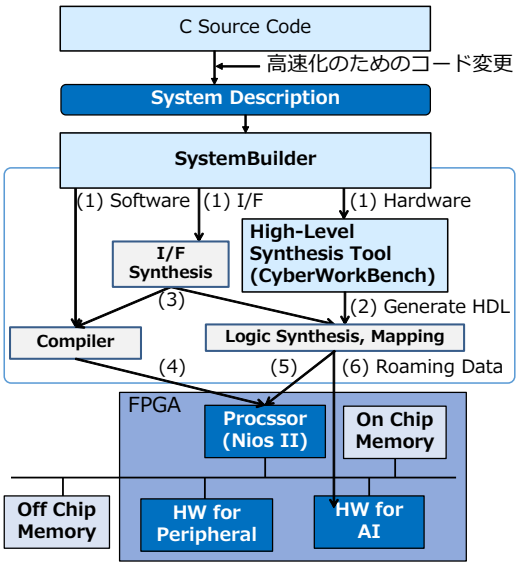

図 1　SystemBuilder の内部フロー

SystemBuilder を用いた設計では，はじめにシステムの各機能を C 言語によって記述する．このとき，システムを機能単位に分割したそれぞれのモジュールのことをプロセスと呼ぶ．続いて設計者は System DeFinition（SDF）ファイルと呼ばれる設定ファイルを作成する．SDF ファイルは YAML 形式に基づいて，それぞれのプロセスについて SW・HW の指定，通信プリミティブの指定などを行う．

SystemBuilder の内部フローの概要を図 1 と以下に示す．

(1) C ソースコードとユーザの作成した SDF ファイルに基づき，SW プロセスと HW プロセスを生成する．
- プロセス間通信のインタフェースも同時に生成する．

(2) HW プロセスに指定された C ソースコードに HLS ツールを適用して HDL を生成する．

(3) 通信プリミティブは HW，SW の両方に埋め込まれる．

(4) SW プロセスおよびデバイスドライバはコンパイラによって NiosII 向けにコンパイルされる．

(5) SystemBuilder によって用意されたベースシステムに生成されたハードウェア記述言語（HDL）を追加して論理合成を行う．
- ここで，NiosII プロセッサシステムも合成される．

(6) 論理合成により，FPGA に書き込むためのマッピングデータが生成される．

本研究で使用する通信プリミティブを以下に示す．

BC（Blocking Channel）： データを読み込む側のプロセスはデータが書き込まれるまで待ち状態となる．また，チャネルは FIFO で扱われる．

MEM: メモリアクセスのためのチャネルである．HW プロセスでは，デュアルポートメモリとして扱われる．

PFBC（Prefetch FIFO BC）： バースト転送で読み込んだデータを内部 FIFO にバッファリングすることで，外部メモリにある連続データを高速に読み出すことができる FIFO である．詳細は後述する．

また，SystemBuilder では，生成された Verilog-HDL ファイルを用いて動作シミュレーションを行い，生成された HDL についての詳細なデバッグや分析を行うことができる．

表 1　小規模 FPGA の例

デバイス名	最大ロジックセル [K]	最大メモリ [MB]
Zynq®-7000	444	3.3
Zynq®UltraScale+ MPSoC	1,143	8.8
Arria 10 GX	1,150	8.2
Cyclone IV	150	0.79
Stratix IV	820	2.9

2.2 FPGA

表 1 に，小規模から中規模の FPGA の面積やメモリを例として提示する．DNN の推論器におけるパラメータ数は，層数が 8 層の AlexNet[5] で 60M 個，19 層の VGG-Net[6] で 138M 個であり，小規模の FPGA デバイスではそれらのネットワークの全パラメータを収められないことがわかる．そのため，FPGA を用いた DNN の推論器の実現には，メモリ容量をいかに低減するかが重要となる．

3. 関連研究

DNN の推論器におけるメモリ容量を低減するため，近年ではパラメータ量子化の研究が行われている．低精度の DNN，すなわち 2 値化された重みおよび特徴量画像を有する 2 値 NN（BNN）の可能性を実証している [7]．BNN は，主な演算がビット単位の論理演算である．2 値データを扱うことでメモリ要件が大幅に削減されるため，FPGA 実装に非常に適している [7]．

また，パラメータおよび特徴量画像を 3 値化する 3 値 NN（TNN）[8] の研究も行われている．TNN では，BNN による推論の精度よりも高い精度で推論を行うことができることが示されている．3 値による DNN の処理ではパラメータや特徴量画像データを-1, 0, 1 として扱う．0 は乗算を行う必要がないため，実質の積和演算量は 2 値のものと変わらないことが示されている [9]．

また，BNN の推論器を，HLS によって実現する開発環境を構築する研究として FINN[10] や FINN-R[11] がある．FINN が 1 つの HW アーキテクチャのみをサポートしているのに対し，FINN-R は，2 つの HW アーキテクチャをサポートしている．FINN-R では任意の精度の重みや特徴量画像データを選ぶことができる．

Nakahara ら [12][13] によって開発されている GUINNESS は GUI 開発可能な DNN の開発環境であり，単純かつ少ない操作で学習から実装までを実施することが可能である．GUINNESS では重みやバイアスを 2 値化している．

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

表 2　使用するネットワークおよびデータセット

	ネットワーク	データセット
NUM-class10	LeNet	MNIST
RGB-class3	TinyCNN	CL3-Dataset*3

表 3　NUM-class10 の層構成

層名	層の種別	入力			出力		
		高さ	幅	CH	高さ	幅	CH
layer0	Conv.	32	32	1	28	28	6
layer1	Ave.Pool.	28	28	6	14	14	6
layer2	Conv.	14	14	6	10	10	16
layer3	Ave.Pool.	10	10	16	5	5	16
layer4	Conv.	5	5	16	1	1	120
layer5	F.C.	1	1	120	1	1	10

GUINNESS は Linux 上で動作し，生成されるソースコードは SDSoC 向けである．

4. 対象ネットワークとデータセット

本研究では，以下の 2 種類の推論プログラムを使用する．

NUM-class10: 手書き数字推論プログラム *1[14][15]

RGB-class3: DNN フレームワーク GUINNESS*2 により生成された RGB 画像推論プログラム

ここで RGB-class3 は，C++言語によって記述されたプログラムを，実行結果に変化がないように C 言語による記述に書き換えたプログラムである．各プログラムにおいて使用されるネットワークとデータセットを表 2 に示す．

4.1 NUM-class10 について

NUM-class10 は，2 値の手書き数字画像データを入力として DNN の処理を行い，0 から 9 の 10 種類の数字を推論結果として出力する．NUM-class10 のネットワークの構成，各層における入出力の特徴量画像サイズ，そしてチャネル数をまとめたものを表 3 に示す．本稿では，特徴量画像の枚数をチャネル数（CH）と呼ぶこととする．

NUM-class10 では，入力画像と各層で出力される特徴量画像は，layer5 の出力以外はすべて 2 値であり，layer5 の出力は 32bit 型の整数データである．また，各層において使用する重み・バイアスは最大 8bit 型の整数データである．特徴量画像データ，入力画像データおよび各層で用いる重み・バイアスは，グローバルの 1 次元配列に保持する．

NUM-class10 には主にメイン関数と，DNN の処理を行う DNN_Body 関数の 2 種類がある．メイン関数が DNN_Body 関数を呼び出す際には，2 値化された入力画像が保持された配列のポインタを渡す．DNN_Body 関数では，各層ごとの関数を呼び出すことによって DNN の処理を行う．DNN_Body 関数における処理が完了した後，メイン関数では推論結果の評価値が最も高いものを算出して出力する．

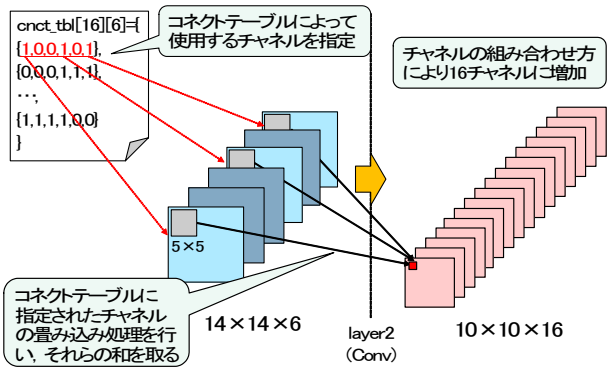

図 2　NUM-class10 の layer2 における畳み込み処理

それぞれの層における処理について説明する．layer0 では，入力画像に対してフィルタサイズ 5 × 5，ストライド 1 の畳み込み処理を行う．フィルタは 6 種類用いる．これにより，32 × 32 の画像である 1 チャネルの入力画像が，28 × 28 の画像となり，かつ，CH は 6 となって出力される．

layer1 および layer3 はアベレージプーリング層である．入力された特徴量画像について，フィルタサイズ 2 × 2，ストライド 2 のプーリング処理を行う．

layer2 では，フィルタサイズ 5 × 5，ストライド 1 の畳み込み処理を行う．16 チャネルの特徴量画像の出力には，図 2 のように，入力の 6 チャネルの特徴量画像のうち，複数チャネルの畳み込み処理結果を総和して用いる．和をとるチャネルの組み合わせは layer2 の持つコネクトテーブルによって決定される．特徴量画像のチャネルごとに使用する重みを変えるため，layer2 は 6 × 16 種類のフィルタをもつ．

layer4 では，フィルタサイズ 5 × 5 の畳み込み処理を行う．入力特徴量画像サイズも 5 × 5 であるため，ストライドは存在しない．入力特徴量画像サイズが 5 × 5，チャネル数 16 である入力に対して，16 × 120 種類のフィルタを用いる．それぞれのチャネルに対して畳み込みを行い，その総和をとってバイアスを付加する．これにより，チャネル数 120 である特徴量画像を出力する．

layer5 は，全結合層であり，120 個の入力データに対して，10 種類のフィルタを用いて 10 クラスの分類を行う．

4.2 RGB-class3 について

RGB-class3 は，RGB 画像データを入力として DNN の処理を行い，車・ペット・飛行機の 3 種類の推論を行うプログラムである．本プログラムにおけるネットワーク構成および各層における入出力の特徴量画像サイズとチャネル数をまとめたものを表 4 に示す．

本プログラムでは，入力画像の画像サイズは 48 × 48 で

*1 http://www.cqpub.co.jp/interface/download/contents.htm 参照.
*2 http://github.com/HirokiNakahara/GUINNESS 参照.
*3 車・ペット・飛行機の 3 種類を用いて学習したデータを用いる．本稿では，これら 3 種のデータで構成されたこのデータセットを CL3-Dataset と呼ぶこととする．

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

表 4　RGB-class3 の層構成

層名	層の種別	入力			出力		
		高さ	幅	CH	高さ	幅	CH
layer0	Conv.	48	48	3	48	48	64
layer1	Conv.	48	48	64	48	48	128
layer2	Conv.	48	48	128	48	48	128
layer3	Max.Pool.	48	48	128	24	24	128
layer4	Ave.Pool	24	24	128	1	1	128
layer5	F.C.	1	1	128	1	1	3

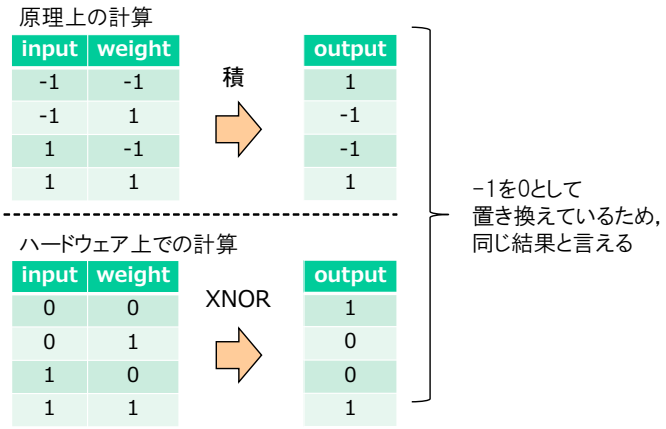

図 3　XNOR 演算による畳み込み処理

あり，RGB の 3 チャネル分のデータを 64bit 型データで表現する．各層において使用する重みは 2 値であり，バイアスは最大 20bit 型データである．各層の出力である特徴量画像データはグローバル変数の配列に保持することで，前層の処理結果を扱うことができる．入力画像データおよび各層で用いる重み・バイアスについては，グローバル定数の 1 次元配列として保持されている．

本プログラムにはメイン関数と，DNN の処理を行う kernel 関数がある．NUM-class10 と同様に，メイン関数では入力画像が配列内に保持されており，kernel 関数の呼び出しにはその配列のポインタを渡して DNN の処理を行う．

それぞれの層における処理について説明する．layer0 では，入力画像に対して 3×64 種類のフィルタや 64 種類のバイアスを用いて，フィルタサイズ 3×3，ストライド 1 の畳み込み処理を行う．本プログラムの重みデータは 2 値であるが，各層では入力の CH に合わせてパックして使用する．畳み込み処理後は，48×48 の画像サイズで CH が 3 の入力が，46×46 の画像サイズで CH が 64 となって出力される．畳み込み処理には，XNOR 演算と popcount 演算 [*4] を用いる．BNN における積和演算は，図 3 のように入力と重みについて XNOR 演算を行った後，popcount 演算を行うことで積和演算の代替処理となる．

layer1 および layer2 では，入力画像に対してそれぞれ 64×128 種類と 128×128 種類のフィルタ，そして 128 種類のバイアスを用いて，フィルタサイズ 3×3，ストライド 1 の畳み込み処理を行う．それぞれの入力の特徴量画像は 46×46 の画像サイズであるが，ゼロパディングを行うことにより前層の入力と変わらず 48×48 となる．layer1 および layer2 における重みも，layer0 と同様に入力の CH に合わせてパックされており，それぞれ 64bit と 128bit 型データとして扱う．これらの層の畳み込み処理も，layer0 と同様の処理で代替する．

layer3 と layer4 はそれぞれ，入力について，フィルタサイズ 2×2，ストライド 2 のプーリング処理を行う．

layer5 は，全結合層であり，128 個の入力データに対して，3 種類のフィルタを用いて 3 クラスの分類を行う．

[*4] 各ビットのうち 1 であるものの数を数える演算．

5. 高速化の適用

本研究では，4 章で説明した 2 つのプログラムについて，SystemBuilder を用いて SW と HW に分割指定して設計を行う．入力画像の受け付けと推論結果を出力する部分を SW，推論処理を行う部分を HW として実装する．重み・バイアスは既に学習済みのものを使用する．

本研究で扱うプログラムにおいて，各層の処理を行うモジュールは入力を受け付けるメイン関数から関数呼び出しによって呼び出され，実行される仕様となっている．本研究ではこの部分を，SystemBuilder 向けに BC を用いた起動/終了信号に書き換える．また，バスを経由してアクセスするデータについては MEM を用いた通信に書き換えた．

本研究では，対象の DNN の推論プログラムに対して複数の高速化手法を組み合わせて実装し，高速化率や面積を測定する．高速化手法の適用順序を以下に示す．

(1) 層レベルのパイプライン化
(2) 層間の通信の FIFO 化，シフトレジスタの導入
(3) パッキング
(4) ループ展開
(5) ループフォールディング
(6) ボトルネックの層の多重化
(7) バースト転送を用いたプリフェッチ FIFO チャネル

入力の画像データは 6 枚入力され，それぞれについて推論を行う．これ以降，DNN の 6 層分の処理が完了するまでの時間を実行時間と呼び，実行時間と面積を各実装の評価の指標とする．スループットとは 1 枚目の入力画像の推論が開始されてから，6 枚目の推論が完了するまでの実行時間をもとに算出したものを指す．

5.1 ベースライン実装（singleloop）

使用する 2 つのプログラムはどちらも，以下の機能を持つプロセスを SW プロセスとして実装する．DNN の処理を行うプロセスを HW プロセスとして実装する．

- 入力画像の受け付け
- 内部メモリへの入力データの書き込み

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

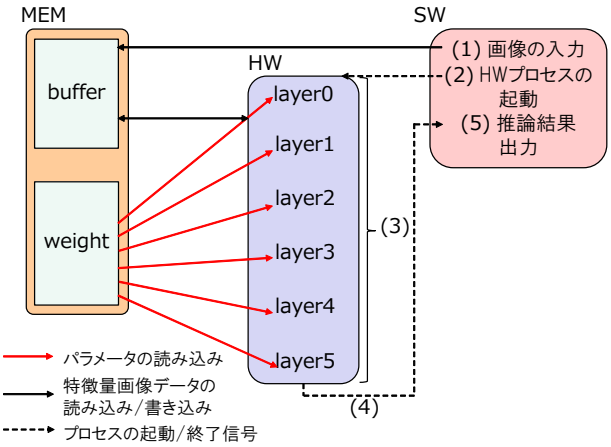

図 4　プログラムの処理フロー概略図

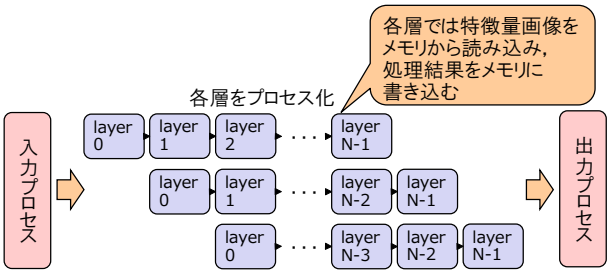

図 5　層レベルのパイプライン概略図

- HW プロセスへの起動信号の送信と終了信号の受信
- 推論結果の出力

プログラムの処理の流れを，図 4 と以下に示す．

(1) SW プロセスは内部メモリに入力データを書き込む
(2) BC を用いて HW プロセスを起動する
(3) 各層はバッファからデータを読み込み，それぞれの処理を行い，処理結果をバッファに書き込む
(4) BC を用いて終了信号を送信する
(5) バッファの情報から推論結果を出力する

各層における畳み込み処理やプーリング処理はループにより実現されている．入力画像データおよびパラメータ，そして各層の出力である特徴量画像データは，すべて内部メモリに保持する．

5.2　層レベルのパイプライン化（pipe）

ベースラインの実装では，各層の処理は逐次実行されるためスループットが低い．そこで，各層をそれぞれプロセスとして分割し，層レベルでのパイプライン化を行う．図 5 のように，プロセスを分割したことでそれぞれの層の処理を並列処理することができるため，入力画像が連続で入力された場合に，スループットが向上する．ただし，各層ごとにプロセスとして分割することにより，各層の出力を保持するためのメモリ領域が必要となる．

この実装では，各層は出力を MEM チャネルを用いて内部メモリに書き込み，それがすべて完了すると次の層の起動信号を BC チャネルによって送信する．次の層は起動信号を受け取った後，MEM チャネルによってメモリから特徴量画像データを読み込み，処理を行う．

5.3　層間の通信の FIFO 化（pipe_fifo）

層間のパイプライン化は各層間で特徴量画像のためのメモリを保持しなければならないため，必要なメモリ領域が増大する．また，すべての特徴量画像データを出力するまで次の層を起動できないと効率が悪い．

そこで，層間の通信の FIFO 化を適用する．各層間の特徴量画像の通信を，MEM に代えて BC を使用し，逐次 FIFO で通信する．各層にはラインバッファを用意し，BC によって送信されたデータは，シフトレジスタに蓄積する．シフトレジスタに蓄積されたデータは，次の範囲の畳み込み処理の際に再利用できる．ただし，NUM-class10 の layer2 は，1 枚の特徴量画像を何度も使用するため，FIFO で通信されたデータはすべて内部バッファに蓄積して使用する．また，NUM-class10 の layer4 は，それぞれのチャネルの畳み込み処理の結果を加算する必要があるため，内部バッファを用意する．これらの内部バッファや各層のラインバッファは内部メモリの配列として実現する．層間を FIFO で通信することで，次の層は 1 回目の畳み込み処理に必要なデータを受信した時点で処理を開始できる．

RGB-class3 においては，特徴量画像データの型が 64bit や 128bit のものがある．SystemBuilder でサポートしているチャネルで扱える型は，32bit 型データが上限であるため，複数回 32bit 型データを送信することで実現している．

5.4　データのパッキング（pipe_fifo_pack）

出力データを次の層に FIFO で送る際に，1 ピクセルずつ送ると効率が悪いため，複数データを一度に送信する．この仕様変更に際し，特徴量画像データの読み込み部分，層出力データを送る書き込み部分に対して，データをパック/アンパックする処理を追加した．

ここで，RGB-class3 においては，例えば 128bit 型データの書き込みは 32bit 型データを 4 回書き込むため，複数データをパックして通信することはできない．そのためこの実装は，特徴量画像データを 8bit 型データとして扱っていた NUM-class10 にのみ適用した．

5.5　ループ展開（pipe_fifo_pack_unroll）

FIFO 通信の実装により，各層の処理の並列性を高めることができた．次は，それぞれの層の内部演算の並列性を高めることによって高速化を図る．具体的には，HLS ツールのループ最適化技術を用いる．CWB では，繰り返し回数が固定と判断できる for ループに対してループ展開を行う指示を与えるオプションを持つ．ループ展開オプションの付加により，for ループ内の配列の自動分割とループ処

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

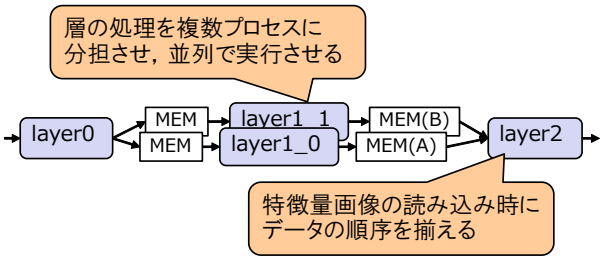

図 6 ボトルネックの層の多重化

理の自動並列化が行われる．本研究では，高速化手法の適用によって増大する面積や適用のコストを考慮し，すべての層に対してループ最適化技術を適用するわけではなく，ボトルネックの層にのみループ展開を適用する．

コシミュレーションを行うことにより，どの層がボトルネックになっているかを判別する．NUM-class10 は，layer0, layer2, そして layer4 が，RGB-class3 は，layer0, layer1, そして layer2 がボトルネックであると判定できた．それらの層に対し，ループ展開を適用する．

ループ展開の適用に際して，重みデータを保持している配列を 1 次元配列から 2 次元配列に変更する．また，各層のラインバッファおよび NUM-class10 の layer2 と layer4 の内部バッファについては，内部メモリの配列からレジスタ指定の配列に変更することで，ループ展開による演算の並列性を高める．

5.6 ループフォールディング（pipe_fifo_pack_folding）

ループ展開は対象ループ内の処理を並列で行うことができるという利点がある反面，それらの処理を並列で行うための演算器が必要となり，面積が増大する場合もある．そこで，ループ回数の多いループに対してはループフォールディングを適用する．

ループ内の処理速度を向上させるために，ループ内の処理をパイプライン化することをループフォールディングと呼ぶ．ループフォールディングを用いると，演算器を共有することができ，ループ展開を行う場合に比べて面積の増加を抑えることができる場合がある．

コシミュレーションの結果，ループ展開を行っているループのうち，面積は増えるものの高速化率が大きくないと判断したものについては，ループフォールディングを代替手法として適用し，面積の増加を抑える．

5.7 ボトルネックの層の多重化（pipe_fifo_pack_folding_d）

各層のそれぞれの実行時間が短縮されたが，これまでの高速化手法を適用するだけでは高速化に限界が生じる．そこで，面積は著しく増加するが，ボトルネックの層の多重化を行う．ボトルネックの層を多重化することにより実行時間を低減すると，スループットの向上が予想される．コシミュレーションの結果，NUM-class10 においては layer2 が，RGB-class3 においては layer0, layer1, そして layer2 がボトルネックであると判断できたため，これらの層をそれぞれ 2 プロセスに多重化し，図 6 のように並列処理を行った．このとき，多重化したそれぞれの層は，例えば，特徴量画像の偶数枚目の処理，奇数枚目の処理を担当する．その場合，図 6 における layer2 は MEM(A) から特徴量画像 1 枚分のデータを読み込んだ後，MEM(B) から読み込むといったように，layer2 が読み込むメモリを交互に切り替えることでデータの整合性を保つ．

5.8 処理速度を維持したまま，面積を削減する手法（pipe_fifo_pack_folding_d_small）

ここまでの実装のうち，最もスループットの大きい実装は pipe_fifo_pack_folding_d であった．この実装について，スループットを維持しつつ面積を小さくする．

各層ごとに高速化手法を適用する前に戻すことを試行し，スループットを維持したまま面積が小さくなるような変更があるかを調べる．結果として，NUM-class10 の pipe_fifo_pack_folding_d に対して以下の変更を加えた．

(a) ループ展開の適用時にレジスタに指定した layer2 の内部バッファを内部メモリの配列に戻す
(b) layer3 と layer4 の間の通信を BC を用いた FIFO から MEM と BC を用いた通信方法に戻す
(c) layer3 と layer5 のフォールディングオプションの除去

RGB-class3 の pipe_fifo_pack_folding_d は，スループットを維持して面積を低減することはできないと判断した．

5.9 singleloop の高速化（singleloop_loop_opt）

各層を逐次実行するプログラムにおいて，どこまで高速化を施すことができるかを調べる．以下の高速化手法を NUM-class10 の singleloop に適用する．

- 畳み込み処理を行うループに対するループ展開
- 各層の出力を保持するバッファはレジスタに指定
- layer0, layer1, layer2 に対するループフォールディング
- 重みを保持した 1 次元配列を 2 次元配列に変更

5.10 外部メモリアクセスの効率化（small_fpga，small_fpga_pref）

ここまでの実装はすべてのパラメータおよび特徴量画像データを内部メモリに保持していた．ここで，小規模 FPGA における実装を考える．

画像サイズの大きい RGB-class3 の，最高スループットを実現する実装である pipe_fifo_folding_d について，外部メモリを使用する場合のスループットを計測する．本実装では，畳み込み層である layer0, layer1, layer2 の重みを外部メモリに配置する．外部メモリへのアクセスは MEM チャネルを使用し，各層の畳み込み処理が行われる直前に

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

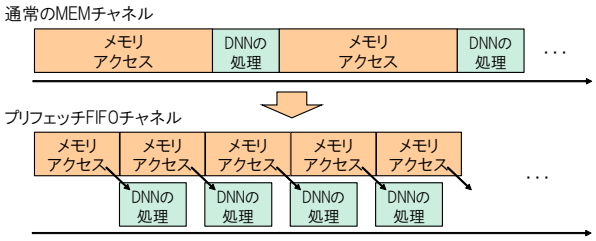

図 7　プリフェッチ FIFO チャネルによるメモリアクセスの並列処理

表 5　実装環境

Intel®環境	QuartusII	ver.17.1.1
FPGA ボード	Cyclone®IV	・面積: 114,480 (LEs)
		・メモリ: 3,888 (Kbits)
	Stratix®IV	・面積: 228,000 (LEs)
		・メモリ: 17,133 (Kbits)
HLS ツール	CWB®	ver.6.1

必要な重みデータを読み込む（この実装を small_fpga とする.）.

外部メモリアクセスのレイテンシを隠蔽する高速化手法を適用する．バースト転送を用いたプリフェッチ FIFO チャネルを適用することにより，外部メモリ使用時にどこまで高速にすることができるのかを試行する．

外部メモリへのアクセスはレイテンシが大きいため，そのレイテンシを隠蔽するために SystemBuilder がサポートしているプリフェッチリード FIFO チャネルを用いる．

プリフェッチ FIFO チャネルでは，バースト転送によりメモリアクセス効率を向上する．バースト転送は，メモリに保持された連続データを高速に読み込むことができるメモリアクセス手法である．

プリフェッチリード FIFO チャネルは，読み込んだデータを内部メモリにキャッシングする．これを DNN の処理と並列に行うことで，図 7 のように，DNN の処理が行われている間に外部メモリとの通信を並列に行うことができる．図 7 の上と下においてメモリアクセス時間が異なるのは，バースト転送により 1 回分のメモリアクセス時間が短くなることを示す．これにより，外部メモリへのアクセスのレイテンシを隠蔽できる．

外部メモリへのアクセスに使用していた MEM チャネルを，すべてプリフェッチ FIFO チャネルに変更した．これにより，メモリアクセスのレイテンシを隠蔽してスループットがどれだけ向上するかを調べる．

6. 評価結果

6.1 実装環境

本研究の実装において使用した環境を表 5 に示す．本研究では主に Stratix®IV FPGA を用いて DNN の推論プログラムの実装を行った．また，5.10 節以降は小規模 FPGA への実装を想定し，Cyclone®IV FPGA を用いた．

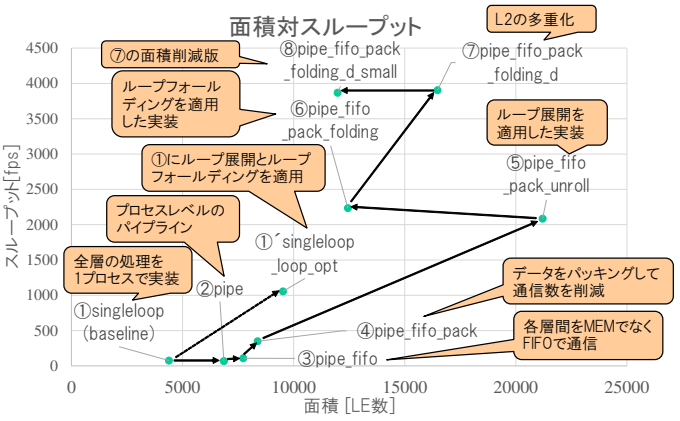

図 8　NUM-class10 の高速化結果

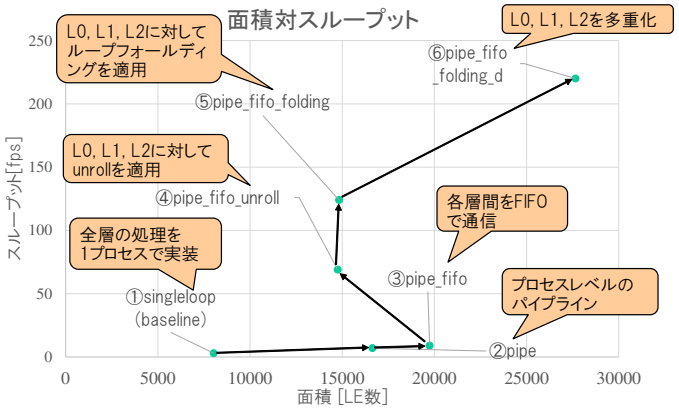

図 9　RGB-class3 の高速化結果

6.2 結果と考察

図 8, 図 9 は NUM-class10 と RGB-class3 各実装の計測結果であり，面積に対するスループットを表している．

図 8 における②では，スループットは向上していない．これは，パイプライン化による効果よりもメモリアクセスのレイテンシが上回ったためだと考えられる．③でも，大きくスループットが向上していない．これは，各層の実行時間の差が大きく，FIFO によるパイプラインの効果を大きく得られなかったためであると考えられる．

④は，③に比べ，スループットが約 3.2 倍になっている．これは，メモリアクセスが効率化されたため，②では効果がなかった層レベルのパイプライン化の効果を引き出すことができたためだと考えられる．

⑥は④と比べて，スループットが約 6 倍になっている．これは，ループ最適化を行うことによりボトルネックの層の実行時間が短縮され，各層の実行時間の差が縮まり，パイプラインの効果が得られたためであると考えられる．

また，①′ を②，③，④における変更を行わなかった⑥の実装と考えると，①′ に比べて，⑥は 2 倍以上のスループットを実現できているため，⑥のスループットの向上は②，③，④の変更による効果であることがわかる．

⑤および⑥はボトルネックの層の多重化を適用した結果

Vol.2019-ARC-235 No.37
Vol.2019-SLDM-187 No.37
Vol.2019-EMB-50 No.37
2019/3/18

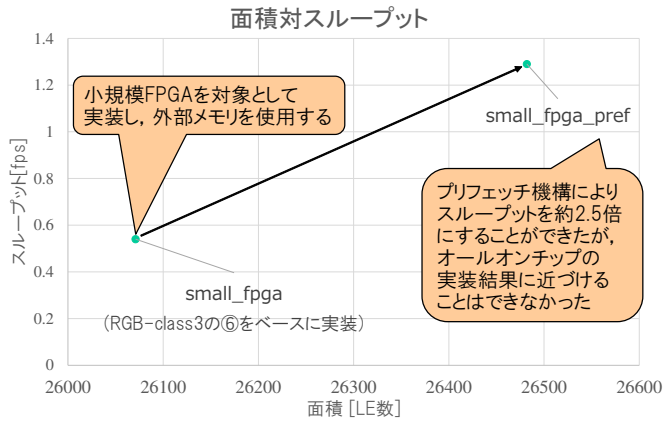

図 10 小規模 FPGA（Cyclone®IV FPGA）を用いた実装結果

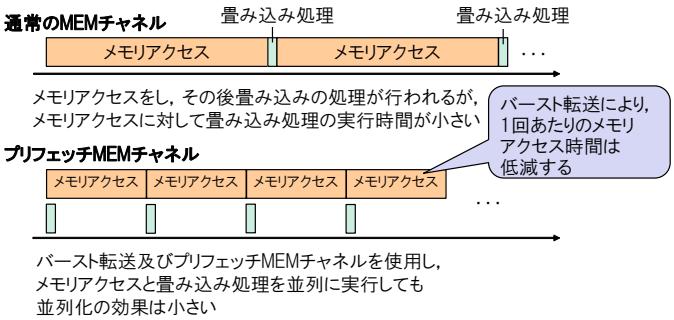

図 11 プリフェッチ FIFO チャネルの考察

であるが，これも先述の通り，ボトルネックの層の実行時間を短縮したことにより，パイプライン化の効果を引き出すことができたため，スループットを大きく向上することができているものと考えられる．

RGB-class3 の各実装についても，NUM-class10 の実装と同様の効果が見られた．

図 10 は small_fpga と small_fpga_pref の評価結果である．small_fpga に比べて small_fpga_pref は約 2.5 倍高速になった．しかし，RGB-class3 の最高のスループットの実装である pipe_fifo_folding_d を Stratix®IV FPGA で実行した場合と比べると著しくスループットが低減している．これは，重みの読み込みに対して畳み込みループにおける処理時間が小さく，図 11 のように並列アクセスによって隠蔽できる時間が小さかったためであると考えられる．

7. おわりに

本研究では，SystemBuilder によって高速な DNN の推論器を設計した．結果として，最も効率の良い実装で約 51.4 倍スループットが向上した．また，小規模 FPGA に対する実装を検討したが，中規模 FPGA を使用した場合と比べて，スループットの改善率の向上が低かった．

今後は，SystemBuilder の機能改善を行い，プリフェッチ FIFO チャネルのバス幅を広げることで，small_fpga_pref の実装から約 8.6 倍スループットが改善されることを見込んでいる．

謝辞 本研究の一部は，国立研究開発法人 新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果得られたものです．

参考文献

[1] Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G. and Marr, D.: Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC, *FPT 2016*, IEEE, pp. 77–84 (2016).
[2] 荒井航平，井倉将実：FPGA/CPLD の基礎と最新動向，FPGA 活用チュートリアル，Vol. 2007, pp. 7–20 (2006).
[3] 三好健文：FPGA 向けの高位合成言語と処理系の研究動向，コンピュータソフトウェア，Vol. 30, No. 1, pp. 76–84 (2013).
[4] 本田晋也，冨山宏之，高田広章：システムレベル設計環境: SystemBuilder，電子情報通信学会論文誌， Vol. 88, No. 2, pp. 163–174 (2005).
[5] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Advances in Neural Information Processing Systems 25* (Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q., eds.), Curran Associates, Inc., pp. 1097–1105 (online), available from ⟨http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf⟩ (2012).
[6] Simonyan, K. and Zisserman, A.: Very deep convolutional networks for large-scale image recognition, *ACPR 2015* (2015).
[7] Zhao, R., Song, W., Zhang, W., Xing, T., Lin, J.-H., Srivastava, M., Gupta, R. and Zhang, Z.: Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs, *FPGA 2017*, ACM, pp. 15–24 (2017).
[8] Prost-Boucle, A., Bourge, A., Pétrot, F., Alemdar, H., Caldwell, N. and Leroy, V.: Scalable high-performance architecture for convolutional ternary neural networks on FPGA, *FPL 2017* (2017).
[9] Li, F. and Liu, B.: Ternary Weight Networks, *arXiv preprint arXiv:1605.04711* (2016).
[10] Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M. and Vissers, K.: FINN: A Framework for Fast, Scalable Binarized Neural Network Inference, *FPGA 2017*, ACM, pp. 65–74 (2017).
[11] Blott, M.，Preußer, T. B.，Fraser, N. J.，Gambardella, G.，KennethO’ brien，Umuroglu, Y.，Leeser, M.，Vissers, K.：FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks, *TRETS 2018*, Vol. 11, No. 3, p. 16 (2018).
[12] Yonekawa, H. and Nakahara, H.: On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA, *IPDPSW 2017*, IEEE, pp. 98–105 (2017).
[13] Nakahara, H., Fujii, T. and Sato, S.: A fully connected layer elimination for a binarizec convolutional neural network on an FPGA, *FPL 2017*, IEEE, pp. 1–4 (2017).
[14] 中原啓貴：Interface2016 年 8 月号，CQ 出版 (2016).
[15] 中原啓貴：Interface2016 年 9 月号，CQ 出版 (2016).