卒　業　論　文

題　　目

マルチFPGAボード上でのGoogLeNetの実装に関する研究

年　　度

平成29年度

所　　属

慶應義塾大学

理工学部　情報工学科

指導教員

天野 英晴 教授

氏　　名

61401007

飯塚　健介

卒 業 論 文 要 旨

学 科	情報工学	学 籍 番 号	61401007	フリガナ 氏 名	イイヅカ ケンスケ 飯塚 健介

(論 文 題 名)

マルチ FPGA ボード上での GoogLeNet の実装に関する研究

(内 容 の 要 旨)

昨今，人工知能が最新技術のトレンドとして様々なメディアに取り上げられている．人工知能技術が組み込まれる自動運転，レコメンドシステム，自動翻訳などのサービスは日々の生活をより豊かにすると期待されている．人工知能技術を実現させる機械学習の中でも，画像認識や自然言語処理，物体検出などの分野で大きな貢献を果たしているニューラルネットワークは一躍注目されていて，研究開発が盛んに行われている．ニューラルネットワークの一種である畳み込みニューラルネットワーク (CNN: Convolutional Neural Network) は畳み込み演算を主な計算とする．CNN は認識精度向上を目指し様々なモデルが提案されているが年々その計算量が増加する傾向にあり，研究サイクルを早くする，データセンターでのアプリケーションとしての利用に耐えうる，高速化，電力性能向上が求められている．

しかし，汎用プロセッサではその要求を満たすことができないので，各半導体メーカや研究機関は専用のアクセラレータの開発に取り組んでいる．日本でも国立研究開発法人新エネルギー・産業技術開発機構 (NEDO) は「省電力 AI エンジンと異種エンジン統合クラウドによる人工知能プラットフォーム」プロジェクトで複数の FPGA，GPU，メモリなどの異種ノードを多数接続した大規模計算基盤 Flow-in-Clowd(FiC) を開発している．複数の FPGA は高機能スイッチノードとして多数の高速リンクが接続され，FiC の高速通信のスイッチングの役割を担う．FiC システムにおいて主演算を行うのは GPU ノードであるが，FPGA ノードもスイッチを実装した上で余った計算資源を利用して AI エンジンとしての役割も担うことができる．本研究の目的はマルチ FPGA システムに 2014 年の国際画像認識コンペで最高精度をマークした CNN モデルの 1 つである GoogLeNet を実装し，評価することで GoogLeNet の高速化を図るとともに，マルチ FPGA システムの深層学習アクセラレータとしての活用ができるかを検討することである．GoogLeNet が持つネットワークモデル特有の計算並列性，畳込み演算の計算並列性を利用してマルチ FPGA システムへの実装を検討するとともに

シュミレーション結果から CPU の○○倍の高速化を達成した．

（内容の要旨は約25行程度で記入のこと）

慶 應 義 塾 大 学 理 工 学 部

目 次

図 目 次

表 目 次

第1章

序論

1.1 本研究の背景

人工知能は爆発的な普及を見せていて，自動運転やスマートスピーカー，スマートフォン向けアプリケーションなど様々なシステムに取り込まれている．人工知能は機械学習が主な技術として実現されているが，機械学習には大量のデータとそれを取り扱う大量の演算が必要となる．そのため人工知能のさらなる普及，発展にはその計算基盤が必要である特に画像認識や物体検出など人工知能の発展に不可欠な分野で活躍する畳み込みニューラルネットワーク (CNN: Convolutional Neural Network) は計算の特性から汎用 CPU では効率よく演算処理ができない．インテルや NVDIA など大手半導体メーカーを始めとして Google や Micrsoft など人工知能サービスを提供する大企業も人工知能向け専用アクセラレータの開発に力を注いでいる．各社，研究機関は GPU，ASIC，FPGA など様々なハードウェア，手法で高速化を図る．日本でも国立研究開発法人新エネルギー・産業技術開発機構 (NEDO) が「省電力 AI エンジンと異種エンジン統合クラウドによる人工知能プラットフォーム」プロジェクトでエッジ側では推論処理 (カメラに何が映っているかを特定するなどの処理) の省電力化を目指したエンジンを，クラウド側では複数の FPGA，GPU，メモリなどの異種ノードを多数接続した大規模人工知能計算基盤 Flow-in-Clowd(FiC) を開発している．この FiC はデータセンターなどに導入されるクラウドシステムとして IoT のセンサから取得したデータなどを学習処理 (教師データから推論モデルの構築) を行う．FiC システムの主演算装置となる複数の GPU を複数の FPGA のスイッチノードに接続し，高速通信を行う．高機能スイッチノードととなるマルチ FPGA は多数の高速リンクが接続され，FiC の高速通信のスイッチングの役割を担う．

FPGA は電力効率のよさ，開発周期の短さ，再構成可能であることなどから注目され推論処理のアクセラレータとしても様々な研究が行われている．GPU は学習処理は非常に高い性能を示すが，推論においては効率が良いとはいえず，また消費電力も大きい.，また ASIC 開発は非常に開発に時間がかかり，たとえ開発に成功したとしてもその莫大な開発コストに見合う大きな市場を探すのは困難である．以上の理由から FiC を構成するマルチ FPGA システムはスイッチノードという役割に加え，AI エンジンとしての役割を担うことも期待されている．そこで本研究ではマルチ FPGA システムの試作ボードである FiC-SW1 を複数枚用いて，CNN のモデルである GoogLeNet[1] を実装し，評価を取った．

1.2 研究目的

本研究の目的は FiC に搭載されているマルチ FPGA システムの試作ボード FiC-SW を複数枚用いて GoogLeNet を実装し，評価を取ることで既存研究や汎用 CPU，GPU と比較してどの程度の性能向上を達成し，実際にクラウドシステムの AI エンジンとして実用的かどうか調べることである．

1.3 本論文の構成

2 章では実装対象である GoogLeNet と畳込みニューラルネットワークの概要を説明する．3 章では FiC プロジェクトの概要と本研究で用いるマルチ FPGA システムを紹介する．4 章では本研究に関連する先行研究について説明する．5 章では GoogLeNet の並列化手法について説明する．6 章では 5 章での並列化を考慮した実装方法について説明する．7 章では本研究の評価を行う．8 章では本論文の結論を述べる．

第2章

GoogLeNet

本研究でマルチ FPGA 上に実装するアプリケーションである GoogLeNet[1] について説明する．GoogleNet は畳込みニューラルネットワーク (CNN: Convolutional Neural Network) のモデルの 1 つである．

2.1 Convolutional Neural Network

2012 年に行われた国際的な画像認識のコンペティション，ImageNet Large Scale Visual Recognition Challenge(ILSVRC) で登場した AlexNet[2] が高い認識精度を出して優勝したことから CNN は特に画像認識の分野で優れた識別精度をマークすることがわかり世界で注目されるようになった．ILSVRC では与えられた画像に何が映っているのか，画像のどこに映っているのかなどをいかに高精度に認識させるかを競う大会である．AlexNet の ILSVRC 優勝を皮切りに様々な CNN のモデルの検討，実装が行われ最新のモデルは人間の認識精度を超える精度を出すようにまでなった．

2.1.1 Neural Network

ニューラルネットワークは動物の神経ニューロンが接続され，神経物質が伝搬されるように演算モジュールを層として複数，結合していく．演算ネットワークでは神経物質の代わりに例えば画像における画素値のような入力ベクトルの演算結果が伝搬していく．ニューラルネットワークでは 2 つの演算フェーズ，学習と推論がある．学習では教師データ (識別された画像) をもとに各演算層のパラメータを決定する．推論では学習で得たパラメータを用いて，入力値 (未識別の画像) から演算 (入力画像の識別) を行う．本研究では CNN の推論アクセラレータを実装するので推論演算で用いられるアルゴリズムについて説明する．

2.1.2 Convolution

CNN はその名前にも含まれているように式で表される畳み込み演算と呼ばれる行列の積和演算がその主なアルゴリズムである．CNN では各層の演算結果を次の層の入力値として受け渡す．これを特徴マップと呼ぶ．特徴マップは学習フェーズで得られた重みフィ

ルタと呼ばれる行列と積和演算が行われ，同じく学習フェーズで得られたバイアスと呼ばれるパラメータを加算することで出力値を求める．

$$\text{output}(f_o, x, y) = \text{bias}(f_o, x, y) + \sum_{f_i=0}^{N_i f} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} \text{weight}(f_o, f_i, k_x, k_y) * \text{input}(f_i, x + k_x, y + k_y) \quad (2.1)$$

2.1.3 Other layer calculation

CNN には畳み込み演算の他にもいくつかの演算を必要とする．その代表的なものとして，全結合とプーリング処理がある．これらはそれぞれ式 2.2，式 2.3 で表される．

$$\text{output}(i) = \text{bias}(i) + \sum_{j=0}^{N} \text{weight}(i, j) * \text{input}(j) \quad (2.2)$$

$$\text{output}(x, y)^i = max/average(\text{input}(f_i, x + k_x, y + k_y))(0 \leq (k_x, k_y) < K) \quad (2.3)$$

全結合層では畳み込み演算同様，入力値と重みフィルタの値を乗算するところは同じである．しかし，それぞれの出力値に入力値全てが必要という点で異なる．また各出力層に対して入力値全てについて計算しなければいけないので畳み込み演算に比べて演算量，パラメータ数ともに大きくなってしまうという特徴もある．プーリング層では入力特徴マップの圧縮を行う．特徴マップのそれぞれのフィルタサイズの領域内の平均値もしくは最大値を取ることで入力特徴マップの特徴はそのままにデータサイズを圧縮することができる．これらの演算を各層で処理し伝搬していき出力されるという構造をとっている．

2.2 GoogLeNet

GoogLeNet は 2014 年の ILSVRC で最高精度をマークし優勝した CNN モデルである．構成要素は上述の畳み込み演算，プーリング処理などが主であるという点においては AlexNet などと違いはないがそれぞれの層の接続の仕方，層の深さが異なり AlexNet に見られる全結合層がないという点で異なる．AlexNet， GoogLeNet それぞれについて図にその全体像を示す．

図 2.1，2.2 のブロックはそれぞれの演算層を示している．青のブロックは畳込み層，赤は pooling 層，ピンクは全結合層，黄色は softmax 層と呼ばれる画像分類のために用いられる．GoogLeNet の出力に近いピンクのブロックは全結合層ではなく dropout と呼ばれる精度を落とさないために前の層の出力結果を次の層に伝搬させない，具体的にはゼロを乗算するという操作を行っている．緑のブロックは DepthConcat 層で並列に処理した出力特徴マップを結合する．両者を比較すると，GoogLeNet のほうが，層がより深くなっていること，さらに横に広がっていることがわかる．GoogLeNet は層を深くする代わりにそのフィルタサイズを小さくすることで，計算量，メモリアクセスを減らすように設計されている．さらに一つの大きなフィルタによる畳込みよりも複数の小さなフィルタによる畳込みを連続して行い，層を深くすることでより高い精度を出すことができるということが知られている [3]．さらに GoogLeNet ではモデル内の横に広がった複数の層を Inception 層と名付け，これを複数層重ねる設計をしている．

2.3 Inception

Inception 層は次の図で示される (これは図 2.2 の一部を拡大したものと同じである)

これを見ると横に広がった層の中で 1×1conv と記されている層がある．この層ではサイズ 1 のフィルターを用いて，畳込み演算を行っている．この層は，次元削減を行っている．これは入力チャネル (入力行列の深さ) に対して少ない層のフィルタを畳み込み演算することでその深さを削減する，これによって計算量が減るだけでなく，精度向上が実現できる．Inception 層の出力の手前の DepthConcat 層は図 2.2 の緑のブロックで示され，横に広がった層のそれぞれの出力結果を図 2.4 のように深さ方向に結合することで一つの行列として出力値にまとめる処理を行っている．

この特徴的な Inception 層を積層していくことで，GoogLeNet は構成される．表 2.1 に GoogLeNet での各層の入出力サイズと必要なパラメータ数をまとめる．

それぞれの Inception 層はサイズは異なるが，図 2.3 に示す構造を取っている．表 2.1 中の#3×3reduce や#5×5reduce は 3×3 や 5×5 の畳み込み演算の前に行われる 1×1 の畳み込み演算の重みの深さを示している．また pool proj も同じように max pool の後に行われる，1×1 の畳み込み演算層の重みフィルタの深さを示している．この表を見ると AlexNet に見られるような全結合層がないことがわかる．GoogLeNet では計算量の多い全結合層を削除することで全体の計算量を減らしながら高い精度を保っている．

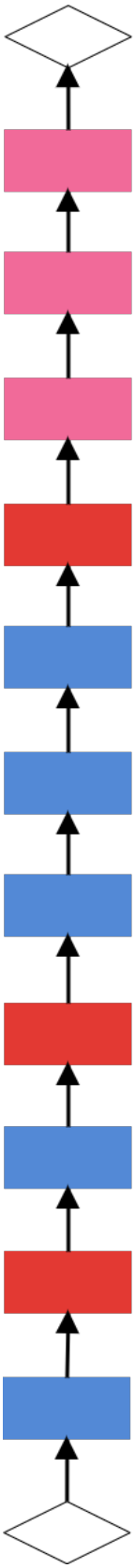

図 2.1: AlexNet のアーキテクチャ

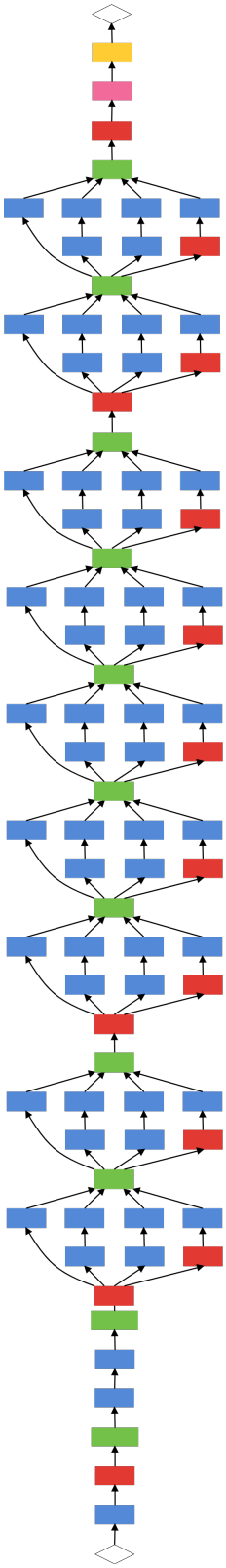

図 2.2: GoogLeNet のアーキテクチャ

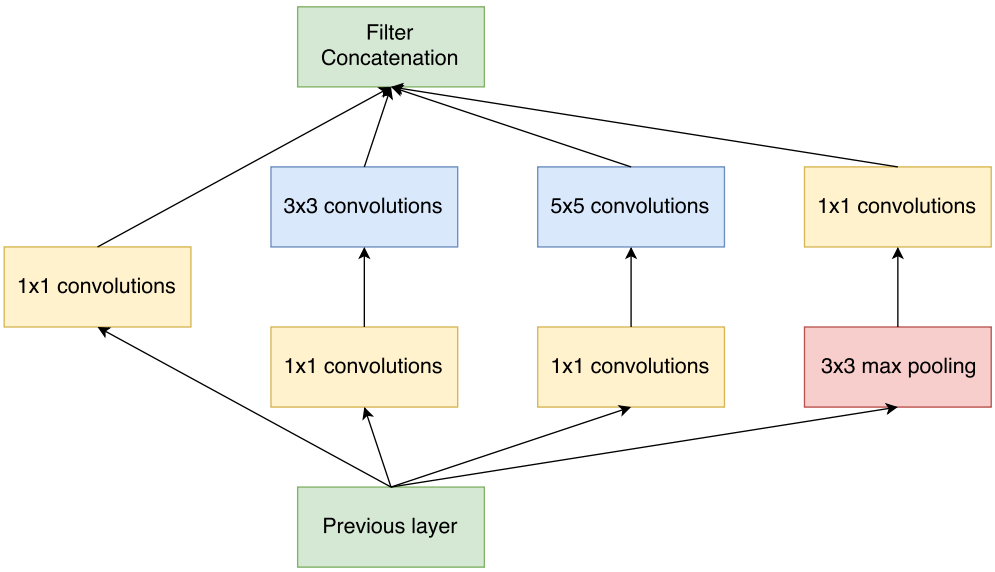

図 2.3: Inception 層

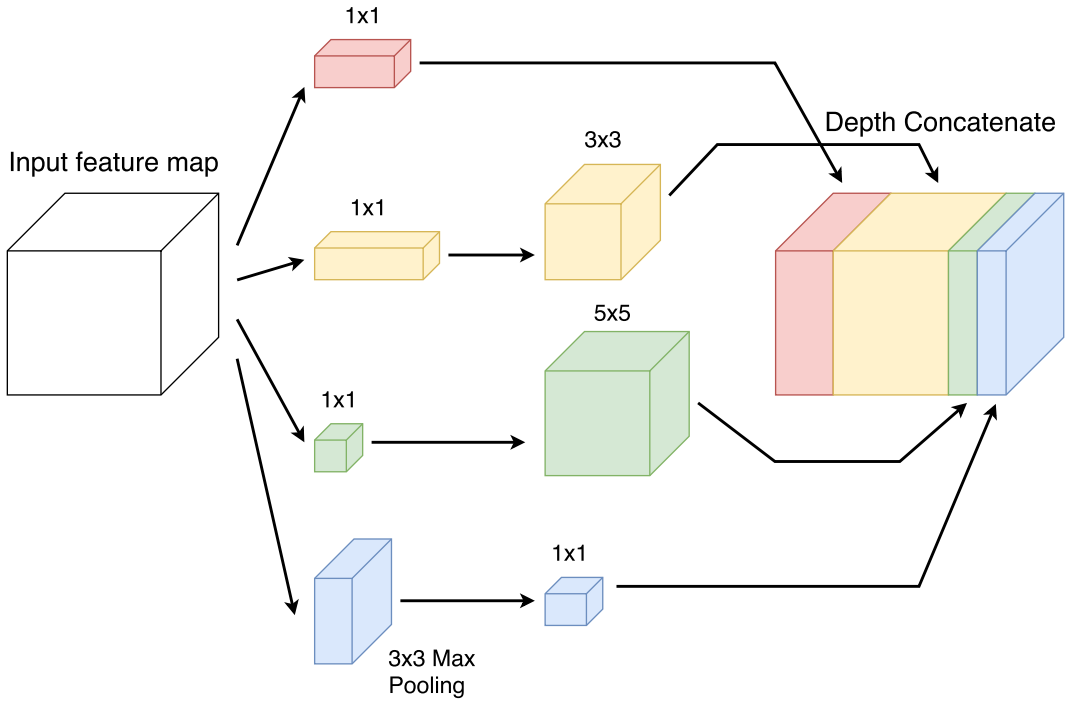

図 2.4: DepthConcat 層

表 2.1: GoogLeNet における各層の構成

type	patch size/ stride	output size	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64							2．7K	34M
max pool	3×3/2	56×56×64								
convolution	3×3/1	56×56×192		64	192				112K	360M
max pool	3×3/2	28×28×192								
inception (3a)		28×28×256	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480								
inception (4a)		14×14×512	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832								
inception (5a)		7×7×832	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024								
dropout (40%)		1×1×1024								
linear		1×1×1000							1000K	1M
softmax		1×1×1000								

第3章

FiCSW

本章では，FiC システムとマルチ FPGA システムである FiCSW の概要を説明する．

3.1 FiC の概要

FiC は NEDO のプロジェクトとして以下のようなシステムの構成を目指している．

1. 様々な種類の処理装置を適材適所で組み合わせて運用する「異種エンジンシステム」とすることで，汎用プロセッサを遥かに超える演算能力を有する
2. 光通信技術の導入により，高速通信が可能なインターコネクトを実現することでホストプロセッサを介することなくエンジン同士を接続し通信のオーバーヘッドの小さなシステムを構築すること

人工知能計算基盤としてのクラウドシステムなので，大量データによる学習フェーズなど即時性を必要としない演算処理を担うことが想定される，それに加え，様々なアプリケーションで柔軟にエンジンを効率よく割り当てるためにエンジン間を動的に変更することが可能なネットワークを構成する必要がある．

3.1.1 FiC のアーキテクチャ

FiC は 3 つの基板 (ノード) とそれらをつなぐネットワークから構成される．ノードには本研究で用いる FPGA ノードだけでなく GPU ノード，メモリノードがある．これらは図 3.1 に示すような接続により計算クラスタとして各種アプリケーションに用いられる．

FPGA はシステムにおいて高機能スイッチノードとしての役割が期待され，他の FPGA や GPU，メモリノードと接続されることで GPU 間の高速通信を実現するとともに FPGA にプログラムされたデータ処理機構などを組み込むことでホストプロセッサの介在を不要とする．メモリノードは FPGA ノードに接続されることで，個々の GPU に大容量 DRAM をもたせる必要がなくなり，GPU デバイスへのメモリコピーをなくすことができ，メモリの利用効率を向上させることができる．

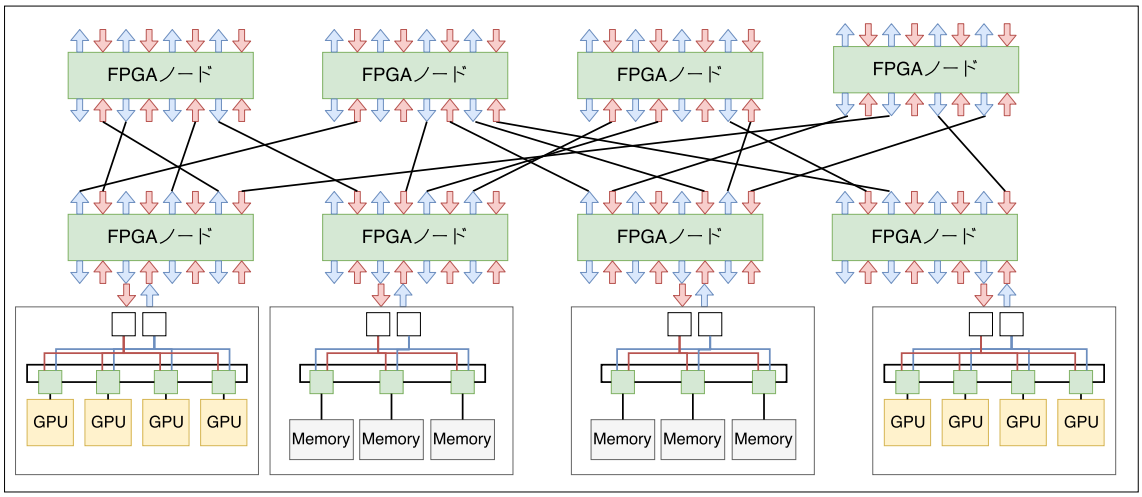

図 3.1: FiC のアーキテクチャ

3.2 FiC-SW の概要

FiC において高機能スイッチとなる FPGA ボードは FiC-SW と名付けられ，その試作ボードは図 3.2 である．

FiC-SW は，図 3.2 中に赤枠で囲まれた 4 つの要素で構成されている．図の注釈の順に

1. Xilinx 社製の UltraScale XCKU095-FFVB2104
2. 16GB の DRAM2 バンク
3. 高速シリアルリンク 8 チャネル，32 リンク
4. Raspberry Pi3

を搭載している．高速シリアルリンクを用いることで，9.9Gbps の通信速度を達成する．Raspberry Pi3 を用いることで，FPGA を遠隔操作して動的に再構成することができる．

3.3 FiC-SW での通信様式

FiC では将来的に広帯域光通信技術をシステムの相互接続ネットワークに用いることが想定されている．光信号のままスイッチすることで数十 Tbps の性能をもつインターコネクトをコストを抑えながらクラウド内で利用することができる．光信号はサーキットスイッチを用いることが現実的である．今回のボードは従来の電気信号による通信を行うが来たる光通信の将来を想定して，サーキットスイッチによるネットワークを構成する．サーキット数を増やすために静的時分割多重 (STDM: Static Time Division Mutipling) による通信様式をとる．

図 3.3 は 4 リンクからなるシリアル入力から同じく 4 リンクからなるシリアル出力へのスイッチの模式図である．各リンクが 4 スロットを有していると仮定している．図中に示される入力スロットと出力スロットの接続が回線を確立する．図の破線矢印で示されるよ

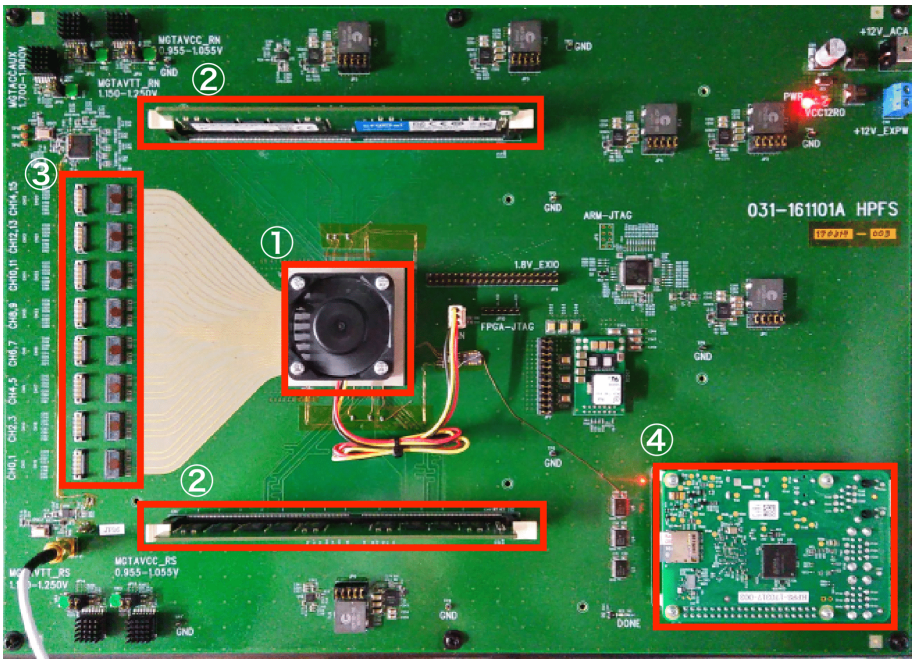

図 3.2: FiC-SW の試作ボード

うに 1 スロットから複数スロットへの出力を設定することも可能なので，容易にブロードキャストを行える．また図 3.3 の下部に示されている FPGA 内部のロジックへの入出力とスイッチのスロットを接続することで FPGA で入力に対して演算を行い，その結果を再びスイッチのスロットに送信することで演算回路と他の FPGA との通信を設定することができる．スイッチングにかかる時間は，入力と FPGA の内部ロジックの関係により決定するため，例えばアクセラレータを実装したときに複数の FPGA にまたがった動作時間が確実に定まるようにすることで通信のオーバーヘッドを削減することができる．

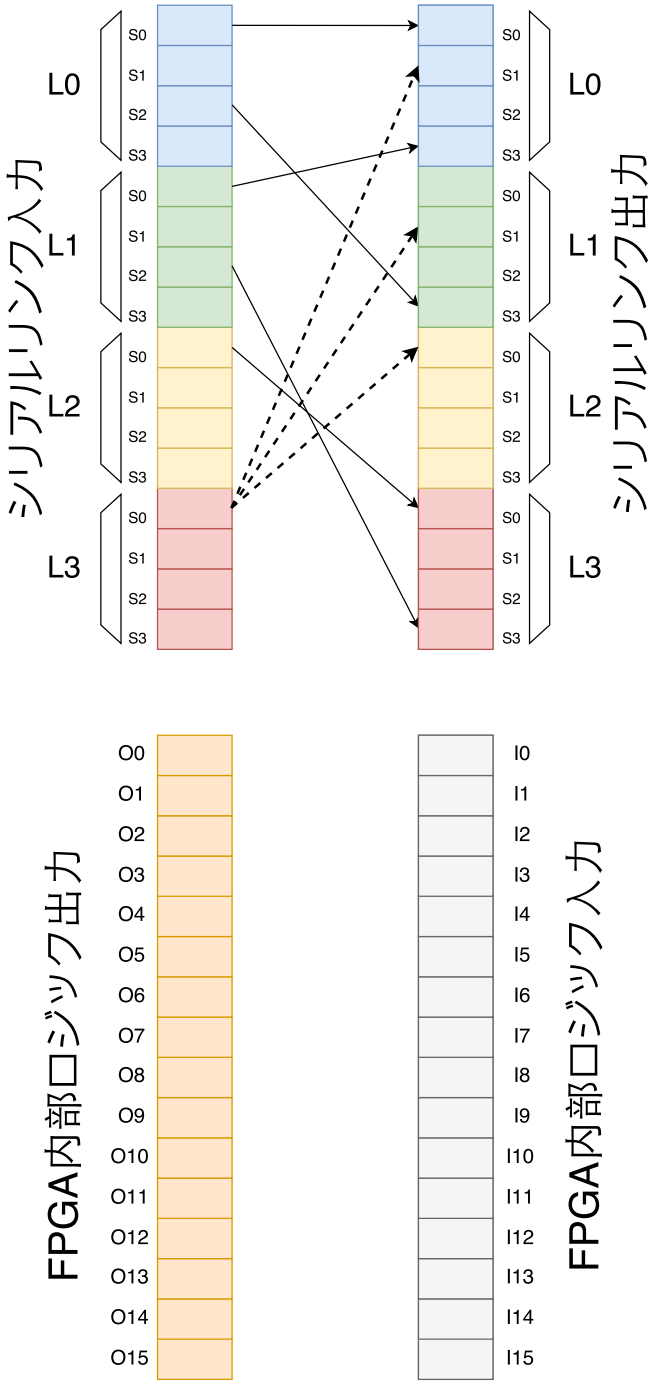

図 3.3: FiC のアーキテクチャ

第4章

関連研究

4.0.1 マルチ FPGA システム

複数の FPGA により構築されるシステムとしては Microsoft 社の提供する Bing 検索エンジンに利用されている Catapult[4] が提案されている．このシステムはデータセンター内の CPU に FPGA アクセラレータが接続されているだけではなく FPGA で 2 次元トーラスのネットワークを作ってノード間の通信も担っている．FPGA では Bing 検索のランク計算部の処理を担っている．通信方式はパケット交換方式なので回線交換を利用する FiC システムとはこの点で異なる．

4.0.2 CNN アクセラレータの研究

FPGA や GPU，専用チップを用いた CNN アクセラレータには，メモリバンド幅によるボトルネックなども含め，複数の層の演算を行うことができる畳み込み演算のアクセラレータを実装した研究 [5] がある．本研究の畳み込み演算器の実装はこの研究のループタイリングなどの技術を参考にしている．Winograd 変換を用いて畳み込み演算が高速化されるという研究 [6] をもとにして、Aria10 上で nVidia の GPU，TitanX にせまる電力性能を達成した研究 [7] もある．またメモリアクセスがボトルネックや消費電力の増大につながるという問題点を解決すべく，圧縮した重みで効率よく畳み込み演算を行う専用チップの開発をした研究もある [8]．本研究では 32bit の重みを利用した演算を行っているが圧縮した重みを利用することで認識精度を落とすことなく，メモリアクセスや通信遅延が改善されるという結果から今後の研究で圧縮した重みを利用した演算器の実装は検討すべきである．

4.0.3 GoogLeNet の FPGA による実装

Caffe フレームワークの FPGA 拡張と Winograd 変換による 3×3 畳み込み演算の高速化検討 [9]

CNN のフレームワークとして有名な Caffe を FPGA 設計でも利用できるように OpenCL で拡張し，さらに Winograd 変換を利用したフィルタサイズが 3×3 の畳み込み演算の高速化を検討し実装した．CPU や GPU に比べて，処理に時間がかかるという結果になってし

まったがフレームワークを FPGA 設計にも導入することで CNN アクセラレータの開発コストが下がり，さまざまなネットワークモデルを FPGA に実装することが容易になることが期待されている．

リソース分割による CNN アクセラレータの効率最大化に関する研究 [10]

[5] など先行研究の多くの FPGA を用いた CNN アクセラレータは，畳み込み演算器 (CLP: Convolutional Layer Processor) を 1 つしか持たない設計をしていることが多い．しかし複数の畳み込み層でこの CLP 単体を使いまわすと各層の入出力のサイズの違いにより，リソース使用量が低下してしまったり，何度も使いまわさなければいけなくなってしまう．複数の CLP を実装することでリソース使用率を最大化することを目指し, ネットワークのサイズによる最適化を行った．その結果，[5] で実装した畳み込み演算器と比較して，AlexNet[2] では 3.8 倍, SqueezeNet[11] や GoogLeNet[1] ではそれぞれ 2.2 倍，2.0 倍の高速化を達成した．

第5章

GoogLeNetの並列化検討

5.1 概要

本章ではまず，GoogLeNet の並列化検討を行う要素を明確にする．次に並列化検討要素ごとに考察を行う．本研究では次の 3 つについて並列化を検討する．

- Inception 層
- 畳み込み演算
- Pooling 処理

5.2 Inception 層の並列化

表 2.1 からもわかるように GoogLeNet はサイズの違いはあるが，Inception 層がその大半を占めている．これが Inception 層に注目した理由である．Inception 層は図 5.1 に示すように 4 つの計算に分割することができる．便宜上図に示すように Thread1 から Thread4 と名

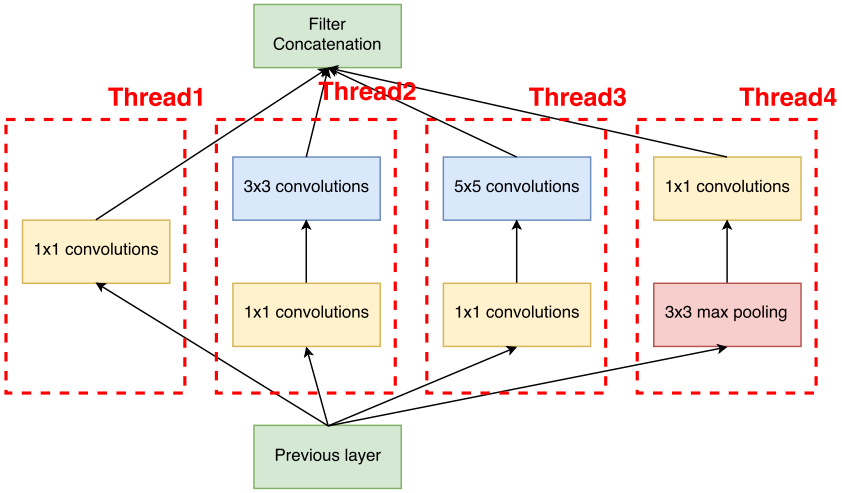

図 5.1: Inception 層の並列化

付けて議論する．各スレッドには前層の出力特徴マップがそれぞれ入力特徴マップとしてブロードキャストされる．このことから Inception 層全体の並列化を考えると各スレッドごとに並列化することができる．マルチ FPGA に搭載する際には各スレッドごとに 1 枚のボードを割り当てるだけでなく，計算処理の重たいスレッドに関しては，スレッド内の畳込み演算部などを並列に処理することができるので，あるスレッド内の処理も複数 FPGA ボードに割り当てることが可能である．

5.3 畳み込み演算の並列化

畳み込み演算の並列化を検討する．CNN はその名の通り畳み込み演算が主演算となるのでこの並列化については 4 で挙げた関連研究をはじめ，様々な研究で検討される．畳み込み演算はその演算の性質上，とくにデータ並列度が高い．本研究では重みフィルタパラメータを各 FPGA ボードのオンチップメモリ (BRAM) に保存し，外部からの入力特徴マップとの演算を行い，出力特徴マップを求めるという設計を行う．これはある FPGA がネットワークの特定の層のある箇所のみの演算を行い次々と FPGA ボードを伝搬させていくという設計方針を満たすためである．次々と入力値を FPGA が受け取りストリーム処理を行うことを想定している．畳み込み演算の並列化には次のようなものが考えられる．

- 出力値分割
- 入力値分割

それぞれについて以下で詳しく述べる．

5.3.1 出力値分割

出力特徴マップを分割することを考える．出力特徴マップ同士には依存性がないが，次層での入力値となることを考えると，すべての特徴出力マップが揃うのを待つ必要がある．この並列化の様子を図 5.2 に示す．図中では Inception3a 層の Thread2 の 3×3 畳み込み演算の入出力サイズを用いている．入力サイズ (96,30,30) に対して重みフィルタサイズ (128,96,3,3) の重みとの演算を行い，出力サイズ (128,28,28) に対して，32 並列化を実行している．並列化された各ノードは入力サイズ (96,30,30) を受け取り出力サイズ (4,28,28) を出力する．こうすることで逐次的に処理するときと比較して理論的には 32 倍の速度性能を達成することができる．しかし演算の最後に並列演算したそれぞれの出力特徴マップを DepthConcat 層の処理のように深さ方向での結合をしなければならない．

5.3.2 入力値分割

入力値について並列化することも考えられる．入力特徴マップのサイズが大きいものについて分割して処理することでスループットの向上の可能性がある．この並列化の様子を図 5.3 に示す．ここでも同じく Inception3a 層の Thread2 の 3×3 畳み込み演算の入出力サイズを用いて説明する．16 個に入力特徴マップを分割するとそれぞれ (6,30,30) の入力サイ

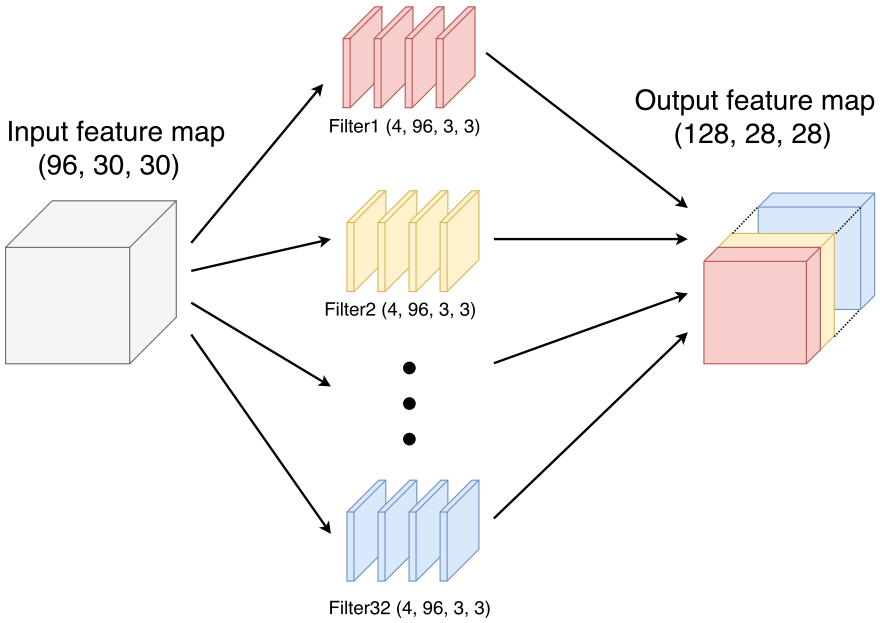

図 5.2: 出力値分割

ズとなる．これに対し 128 個の (96,3,3) のサイズも 16 個に分割し (6,3,3) のサイズにする．分割された入力特徴マップは対応する 128 個の分割された重みフィルタと畳込み演算を行い (128,28,28) の中間特徴マップを得る．この 16 個の中間特徴マップのそれぞれの深さ d ($1 \leq d \leq 128$) の二次元特徴マップの総和をとることで最終的な出力特徴マップの深さ d の二次元特徴マップが得られる．この操作を各 d に対して行うことで最終的な出力特徴マップを得ることができる．分割された入力特徴マップと重みフィルタはそれぞれ並列に演算が可能である．

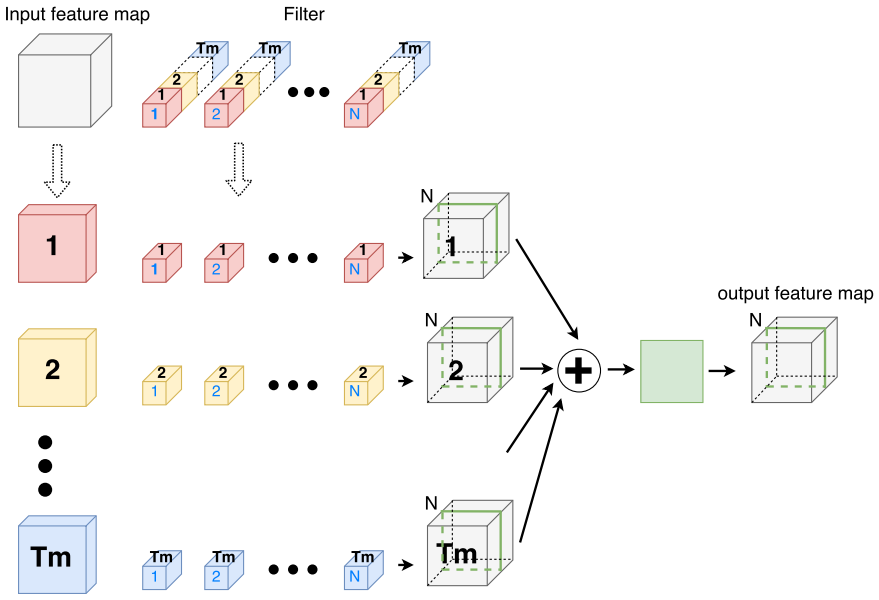

図 5.3: 入力値分割

第6章

実装

本章では5章の並列化に対する検討を元にして、FPGAに実装するアクセラレータについて解説する。

6.1 スレッド並列化の実装

図5.1で示した4つの計算スレッドについて、各スレッドにそれぞれ1枚のFPGAで演算を行うようにする。

6.2 スレッド内モジュールの実装

FiCのマルチFPGAシステムでは、流れてきたデータをそれぞれのFPGAボードでの処理を終えて、次のFPGAに出力を渡す、というフローが想定される。そのため各ボードでは特定の演算処理のみを実行すればよいので[5]のように複数のサイズの畳み込みを行うことは考えずに、それぞれの層に適した入出力サイズのモジュールを実装することが可能である。また特定の層の重みフィルタのみをBRAMに保存して読み出し演算を行う。各スレッドのモジュールはまず、ブロードキャストされる同一サイズの入力値を受け取る。その後、BRAMの重みフィルタとの畳み込み演算を行う。図6.1にモジュールの模式図を示す。入出力のインターフェイスにはXilinxが提供するAXI4-Streamを利用する。AXI4-Streamはデータストリーム用のインターフェイスである。これにより入出力データをストリーム形式で送受信できる。図6.1に示すように入出力、重みフィルタにダブルバッファを設けることで演算に必要なデータを受け取ったら、演算を行うと同時にもう片方のバッファで次のデータの書き込みが可能となる。

畳み込み演算はPEで行う。図5.1で示した4つの計算スレッドのうちThread2、Thread3、Thread4については畳み込みやpooling処理が連続して行われているので図6.1が2つ接続するようなモジュールとなる。ストリーム処理を行い、パイプライン化するためにモジュール間にはバッファを設ける。

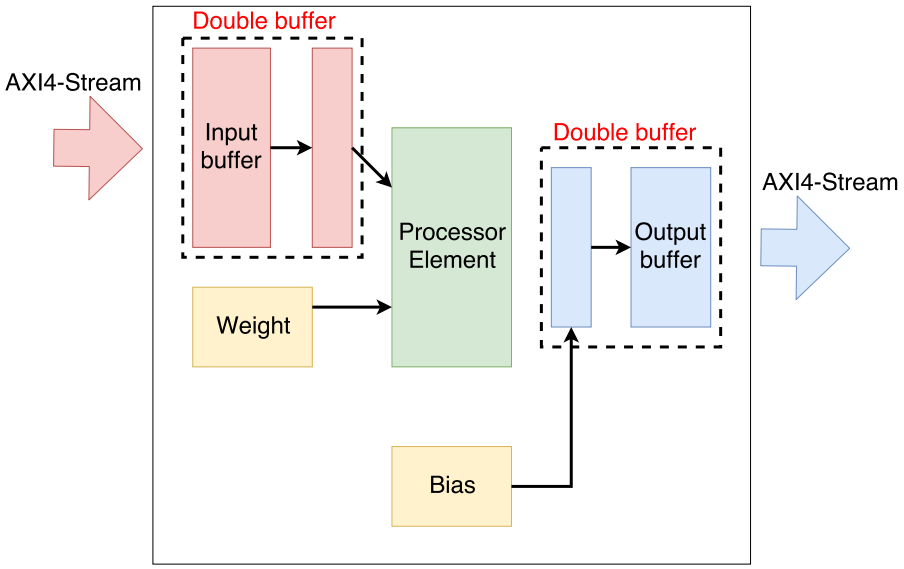

図 6.1: モジュールの模式図

6.3 畳み込み演算器の実装

6.2 節で述べたモジュールの模式図内の PE 部の設計は [5] を参考にした。ここでは 6.3 に示す、畳み込み演算の C コードをループタイリングによって分割することで並列化することを考える。

畳み込み演算の C コード

```
for (int r = 0; r < R; r++)
  for (int c = 0; c < C; c++)
    for (int to = 0; to < M; to++)
      for (int ti = 0; ti < N; ti++)
        for (int i = 0; i < K; i++)
          for (int j = 0; j < K; j++)
            output[to][r][c] +=
              input[ti][S*r+i][S*c+j] *
              weight[to][ti][i][j];
```

出力特徴マップチャンネルで並列化された畳み込み演算の C コード

```
for (int r = 0; r < R; r++)
  for (int c = 0; c < C; c++)
    for (int to = 0; to < M; to += Tm) //Tm ごとに出力を分割
      for (int ti = 0; ti < N; ti++)
        for (int i = 0; i < K; i++)
          for (int j = 0; j < K; j++)
            output[to][r][c] +=
              input[ti][S*r+i][S*c+j] *
              weight[to][ti][i][j];
```

6.3 がループタイリングを行った疑似コードである。T_m が 5.3 節で説明した、出力値による分割である．この疑似コードに倣って演算モジュールを作ると図 6.2 のような演算器を設計できる。

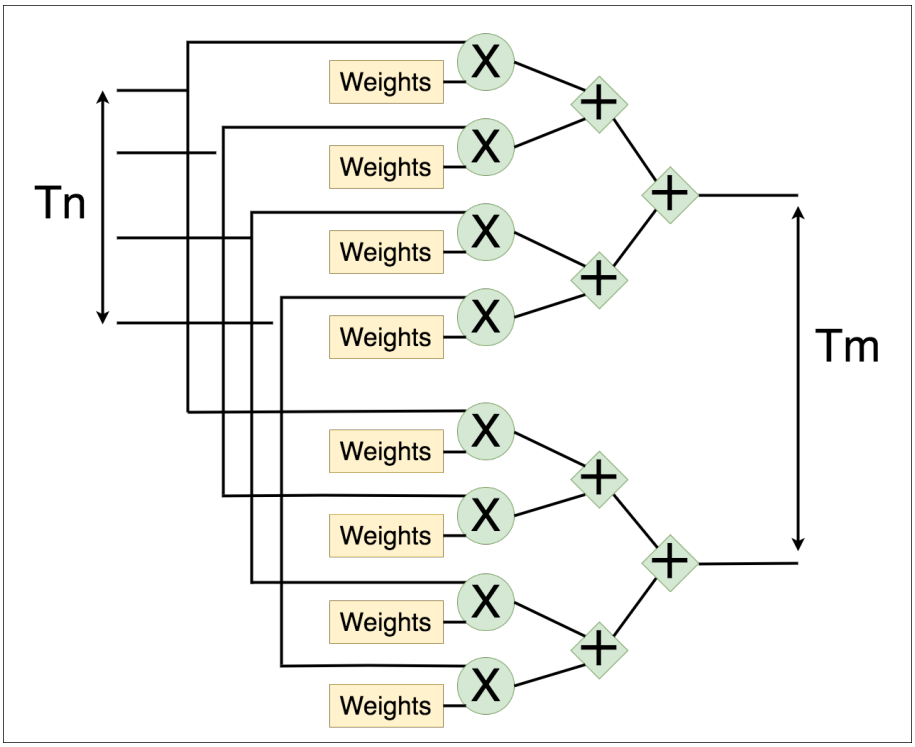

図 6.2: 畳み込み演算器の模式図

この演算器は複数の積和演算器を並列に並べたものとなっている．この演算器は T_n 個の入力値と $T_n \times T_m$ 個の重みから T_m 個の出力値を得る．

6.4 maxpooling 演算器の実装

maxpooling 処理は式 2.3 で示された最大値を取る演算を行う。カーネルサイズの領域を

入力特徴マップから抽出し、その領域内での最大値を取得し出力特徴マップに書き込むという処理を行う。領域内の各要素について逐次的に前後の値を比較し、出力値の候補を探していく処理となるが、本研究では高速化を図るため、トーナメント方式に値を比較する演算器を実装した。その模式図を maxpooling 処理の具体例とともに図 6.3 にモジュールの模式図を示す。

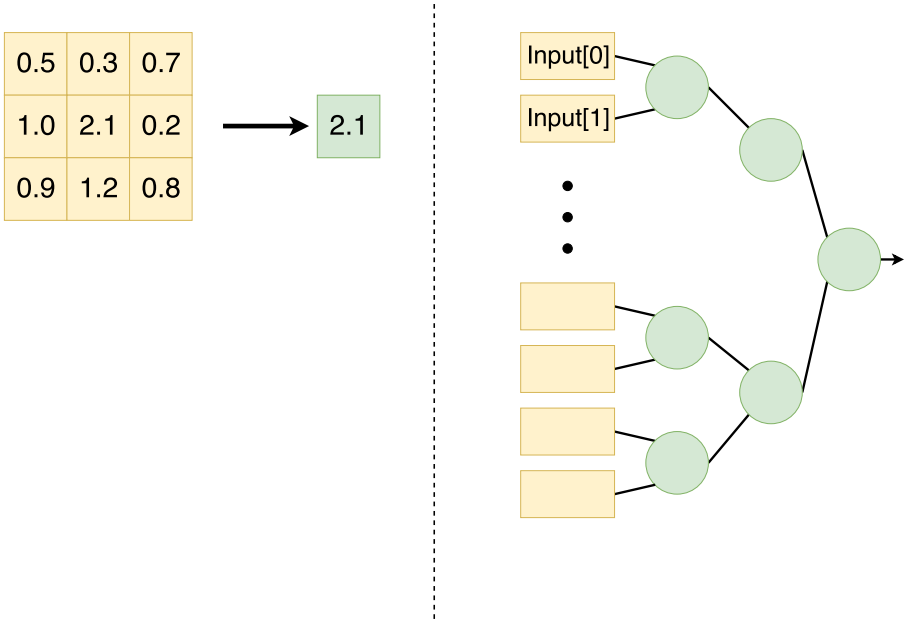

図 6.3: モジュールの模式図

第7章

評価

7.1 評価環境

実装はVivado HLS(2017.3)を用いて実装を行った．対象となるFPGAボードは3章の3.2節でも言及したXilinx社のKintex Ultrascale　XCKU095-FFVB2104である．動作周波数は100MHzとした．実装対象，ベンチマークとして表2.1に示されるInception(3a)層を用いた．Vivado HLS上でのシュミレーションの実行サイクル数から実行時間を計算した。比較対象となるCPUはIntel(R) Xeon(R) CPU E5-2667 0(@2.90GHz)を用いて，g++4.4.7でのO3最適化とOpenMPを使ってコンパイルしたアプリケーションの実行時間を計測した．

7.2 リソース使用量と割合

表7.1に各スレッドモジュールのFPGA合成結果のリソース使用率を示す．

7.3 実行時間の比較

比較対象となるCPU，FPGAにそれぞれGoogLeNetの実装を行い，その実行時間を計測した結果を図7.1に示す．グラフの縦軸は実行時間を対数で示している．また詳しい数値は表7.2に示す．

表7.2に各スレッドモジュールのFPGA実行時間とCPUでの実行時間を示す．また全ての演算を行った際の合計の実行時間も示す．

Inception層では前層の出力特徴マップが全て揃うまで演算を行うことができないので各スレッドはお互いの処理が終了するまでは待機していなければいけない．そのため，Inception層自体の実行時間は一番処理に時間がかかるスレッドに律速されてしまう．

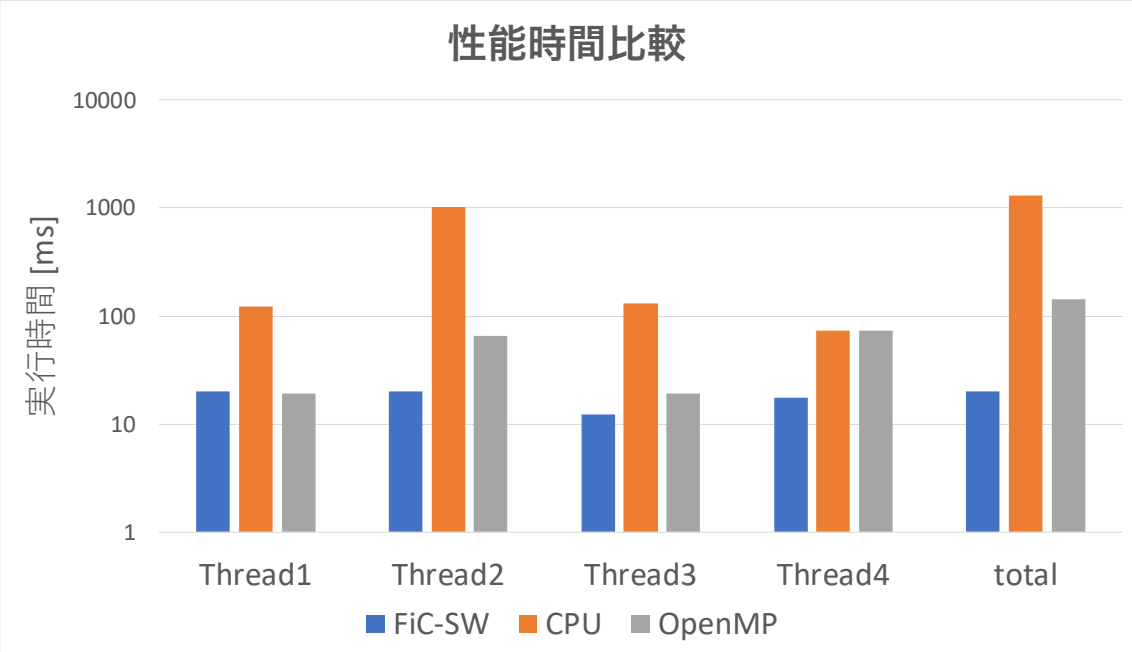

図 7.1:

表 7.1: 各スレッドにおける FPGA ボードのリソース使用量とその割合 [] 内は%

Thread	BRAM_18K	DSP48	FF	LUT
Thread1	10[1]	10[1]	10[1]	10[1]
Thread2	10[1]	10[1]	10[1]	10[1]
Thread3	10[1]	10[1]	10[1]	10[1]
Thread4	10[1]	10[1]	10[1]	10[1]

表 7.2: 各スレッドと Inception(3a) 層全体の実行時間の比較

Thread	CPU	CPU -O3 & OpenMP	FPGA
Thread1	10[1]	10[1]	10[1]
Thread2	10[1]	10[1]	10[1]
Thread3	10[1]	10[1]	10[1]
Thread4	10[1]	10[1]	10[1]
Total	10[1]	10[1]	10[1]

第8章

まとめと今後の課題

8.1 結論

本研究ではNEDOが開発している大規模人工知能計算基盤FiCにおいてスイッチノードとしての役割を担うマルチFPGAシステムが，スイッチとしての機能だけでなく，深層学習アクセラレータとして利用できるのかをGoogLeNetの高速化とともに検討し実装，評価を行った．実装はGoogLeNetの特徴に注目し，Inception層の並列性，畳込み演算の並列性を利用した並列化を検討した．Inception層の各計算スレッドにそれぞれ，1枚ずつFPGAボードを割り当て，畳み込み演算器，Maxpooling演算器についてそれぞれ処理の高速化を図り，Inception(3a)層をベンチマークとしてシュミレーションを行い評価をした．シュミレーション結果として性能比で最適化をしないCPUに対して○○倍，最適化を図ったCPUに対して○○倍の高速化を実現した．

8.2 今後の課題

本論文では基本的な高速化の手法のみを提案し，実装を行った．シュミレーション結果ではあるがCPUに比べて高速化できたことから，FiC-SWで構築されるマルチFPGAシステムはCNNアクセラレータとして，活用が期待できる．しかし，Inception層の各スレッドのロードバランスの検討やマルチFPGAをどのように繋いで，計算に適したネットワークを構築するか，また畳み込み演算における入出力値分割の最適化など更なる性能向上，今回は計測しなかった電力効率の向上などを目指し，検討をしていく予定である．

第9章

謝辞

本研究に取り組むにあたりご指導ご鞭撻を賜りました天野英晴教授に深く感謝いたします．主査をしていただくとともに Vivado の使い方，アクセラレータ実装での助言を頂いた武者千嵯氏，副査として論文の基本的な書き方を教授してくださった安戸僚汰氏にも深く感謝いたします．同じ研究室のメンバーにも大変お世話になりました．志村英樹氏には Vivado の使い方や実装の最適化に関する，アドバイスを数多く頂きました。同期である風見亮佑氏，池添赳治氏には日頃から研究にあたり困ったことを相談したり，卒業論文執筆の際に不具合を直してもらいました．奥原颯氏，Anh Vu Doan 氏には研究グループも部屋も違うのに励ましの声をかけていただいたり，コーヒーをごちそうしてくださいました。研究室の同期，先輩の皆様には本当に日頃からお世話になっており，この論文がなんとか形になったのも皆様のおかげです．深く感謝いたします．最後に大学に通わせてくれ，日々見守ってくれた両親と弟にも深く感謝いたします．

参考文献

[1] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[3] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, Vol. abs/1312.4400, , 2013.

[4] Andrew Putnam. Large-scale reconfigurable computing in a microsoft datacenter. In *Proceedings of the 26th IEEE HotChips Symposium on High-Performance Chips (HotChips 2014)*. IEEE, August 2014.

[5] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason 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*, FPGA '15, pp. 161–170, New York, NY, USA, 2015. ACM.

[6] Andrew Lavin. Fast algorithms for convolutional neural networks. *CoRR*, Vol. abs/1509.09308, , 2015.

[7] Utku Aydonat, Shane O'Connell, Davor Capalija, Andrew C. Ling, and Gordon R. Chiu. An opencl(tm) deep learning accelerator on arria 10. *CoRR*, Vol. abs/1701.03534, , 2017.

[8] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pp. 243–254, Piscataway, NJ, USA, 2016. IEEE Press.

[9] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi. Caffeinated fpgas: Fpga framework for convolutional neural networks. In *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 265–268, Dec 2016.

[10] Yongming Shen, Michael Ferdman, and Peter Milder. Maximizing cnn accelerator efficiency through resource partitioning. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 535–547, New York, NY, USA, 2017. ACM.

[11] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, Vol. abs/1602.07360, , 2016.