卒　業　論　文

題　　目

マルチFPGAボード上におけるCNNの実装に関する研究

年　　度

平成28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

61321882

武者千嵯

卒 業 論 文 要 旨

学 科	情報工学	学 籍 番 号	61321882	フリガナ 氏 名	ムシャ カズサ 武者千嵯

(論 文 題 名)

マルチ FPGA ボード上における CNN の実装に関する研究

(内 容 の 要 旨)

近年、人工知能は、単なる情報サービスにとどまらず、医療、自動車、翻訳、その他多くの分野にまたがって活発に利用されている。身近なセンサー類を用いた機械学習技術が普及する一方で、その高度なデータ処理に伴う消費電力の増大が懸念されている。そこで、国立研究開発法人 新エネルギー・産業技術開発機構（NEDO）は 2016 年に「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」のプロジェクトをスタートさせた。このプロジェクトでは、FPGA、GPU、メモリなどの異種ノードを多数接続した大規模システム FiC(Flow-in-Clowd) が開発されている。多数の高速リンクを接続した FPGA ボードである FiC-SW1 は、FiC システム上でのスイッチノードとしての役割だけでなく、初期のシステムソフトウェア開発用テストベッドとしての役割もある。本稿では、FiC-SW1 ボードの構成を紹介し、ボードの計算性能、転送性能の予備評価を行った。そのために、FiC-SW1 ボード上の FPGA に畳み込みニューラルネットワーク（CNN）の畳み込み層を処理するアクセラレータを実装し評価した。その結果、最適化を施していないアクセラレータでも一般的な CPU に比べて 658 倍高速化することに成功した。また、通信時間に対する計算時間の比率は、2 倍から 10 倍、最大で 26 倍になることがわかった。

（内容の要旨は約25行程度で記入のこと）

目 次

図 目 次

表 目 次

第1章

序論

1.1 本研究の背景

近年、人工知能は、単なる情報サービスにとどまらず、医療、自動車、翻訳、その他多くの分野にまたがって活発に利用されている。たとえば、いくつかの市販のカメラには、機械学習による画像識別によって、撮影された画像から誰が映っているかを判断し、人物ごとに分類しタグ付けを行う機能が組み込まれている。このようなカメラやマイクを利用した身近な機械学習技術が普及する一方で、高度なデータ処理を逐次行うために多くの電力資源が消費されている。機械学習技術の多くは、高度なビッグデータ処理が前提となっており、高性能化のためにも莫大な計算資源が必要となる。

この問題を解決するために、国立研究開発法人 新エネルギー・産業技術開発機構（NEDO）は 2016 年に「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」のプロジェクトをスタートさせた。これは、省電力 GPU やセンサーを搭載した小規模システムのエッジ側と、FPGA や GPU を結合した大規模システムのクラウド側のふたつを開発し、これらをタスクに合わせて効率的に運用することで電力性能の向上を目指すものである。

さて、このプロジェクトのクラウド側にあたる、FPGA、GPU、メモリなどを組み合わせた大規模システム FiC(Flow-in-Clowd) の開発が始まった。これでは、FPGA 搭載の高性能スイッチノードを中心とした大規模計算システムを構築し、特に、小規模システムのエッジ側では処理しきれない機械学習の学習の処理の高性能化・低電力化を目的としている。

また、FiC システムでは、初期のシステムソフトウェア開発用テストベッドとして、FPGA に多数の高速リンクを接続した FiC-SW1 ボードを利用することを計画している。この FiC-SW1 ボードは FPGA として Xilinx 社の Kintex Ultrascale XCKU095 を採用し、8Gbps 全二重の光リンクと STDM（Static Time Divi Time Division Multiplexing）によるサーキットスイッチングネットワークによってボード間を接続するという特徴がある。

本稿では、今後の FiC システム開発のために障壁となる課題を発見するために、この FiC-SW1 ボードの予備評価を行った。そのために、このボード上に、画像識別において機械学習技術の中核となっている畳み込みニューラルネットワーク（CNN : Convolutional Neural Network）のうち、ベンチマークとしてカラー入力画像を 1000 のカテゴリに分類する AlexNet [1] を実装し、FiC-SW1 ボードの計算性能・転送性能をシミュレーションにより計測した。

1.2 本稿の構成

本稿の構成を示す。2 章では本研究の要となる FiC システムと FiC-SW1 ボードについて紹介し、3 章では、今回ベンチマークとして実装する CNN について解説を行う。4 章では本研究に関連の深い研究を紹介する。5 章では本研究の目的と課題を明らかにする。6 章では、CNN を FiC 上に実装するために、CNN の並列化を検討する。7 章では今回実装を行う畳み込みアクセラレータの構成を解説し、8 章ではそれを FiC-SW1 ボード上に実装した際のシミュレーションによる評価を行う。9 章では本研究の結論と今後の課題について述べる。

第2章

FiC-SW

2.1 FiC(Flow-in-Cloud) システム

FiC(Flow-in-Cloud) システムは、NEDO 人工知能プロジェクトにおいて、FPGA ノード、GPU ノード、メモリノードなどの異種ノードを多数組み合わせた大規模計算システムである。図 2.1 は FPGA 搭載スイッチノード (FiC-SW) を中心とした FiC システムの構成例を示す。このシステムはデータフローによって各ノードが動作し、目的の処理が進められるようになっている。図 2.2 は FPGA、GPU、メモリを組み合わせ機械学習処理を行うシステムの構成例である。

さらに、FiC システムでは、各ノード間の接続は低コストなサーキットスイッチングで行う。これは、通信の帯域を保証することで、このシステム上で動作する専用のシステムソフトウェアのオーバーヘッドを可能な限り減少させ、アプリケーションを高速化させる狙いがある。

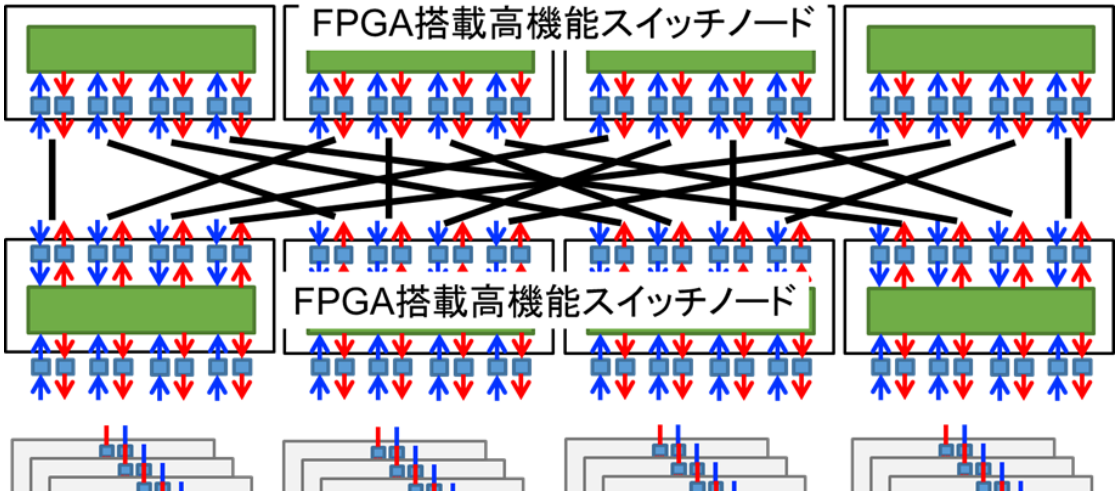


図 2.1: FiC の構成例

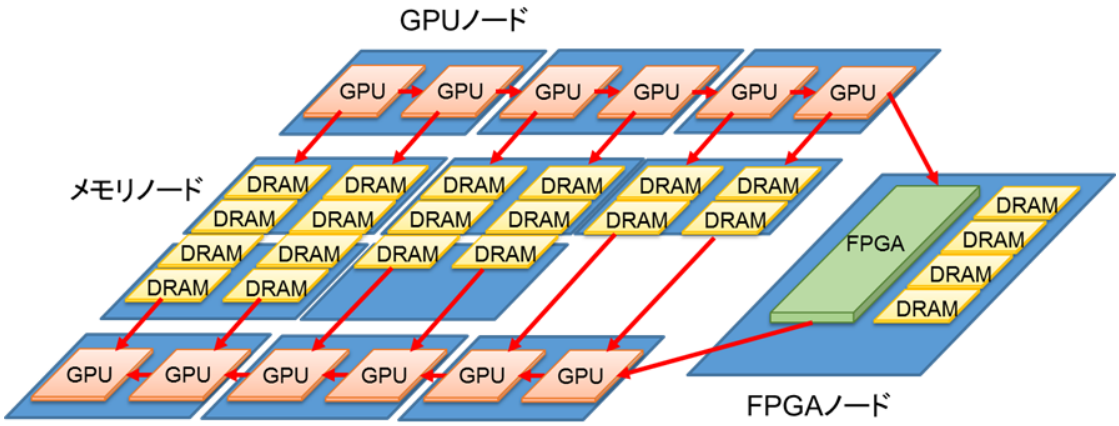


図 2.2: FiC による学習システム構成例

2.2 FiC-SW1

FiC-SW1 は、初期のソフトウェア開発用のテストベッドのために使う、高速リンクを多数接続した FPGA ボードである。この FiC-SW 1 ボードはシステムソフトウェア開発用テストベッドとしての役割だけだでなく、将来の FiC システムのスイッチングハブとしての役割、GPU 開発が完了するまでの演算コアとしての役割もある。

図 2.3 に現在開発中の FiC-SW1 ボードの構成を示す。FPGA には Xilinx 社の Kintex Ultrascale KU095 を用いる。ストレージとしては DDR-4 SDRAM16GB を 4 組持つ。ボード間接続のためには 8Gbps の高速光シリアルリンクを 8 組装備する。リンクは全二重のため、合計の転送容量は 16Gbps となる。

また、制御用に制御用に RaspberryPi 3 ボードをドーターボードの形で接続する。FPGA の再構成などはこの RasPi3 から Ethernet 経由で行う。

2.3 FiC-SW1 のサーキットスイッチング

FiC システムでのスイッチノードの役割も持つ FiC-SW1 ボードのサーキットスイッチングによる通信についてを解説する。

サーキットスイッチングとは、通信の前に事前に通信路を設定し、一定の帯域を占有してから送受信を行う通信の方式である。FiC システムが扱う機械学習アルゴリズムの多くは、1 対 1、1 対全、全対全など、特定の通信路をあからじめ予測することが可能である。そのため、通信帯域と通信遅延を保証することができるサーキットスイッチングが FiC-SW1 には採用された。

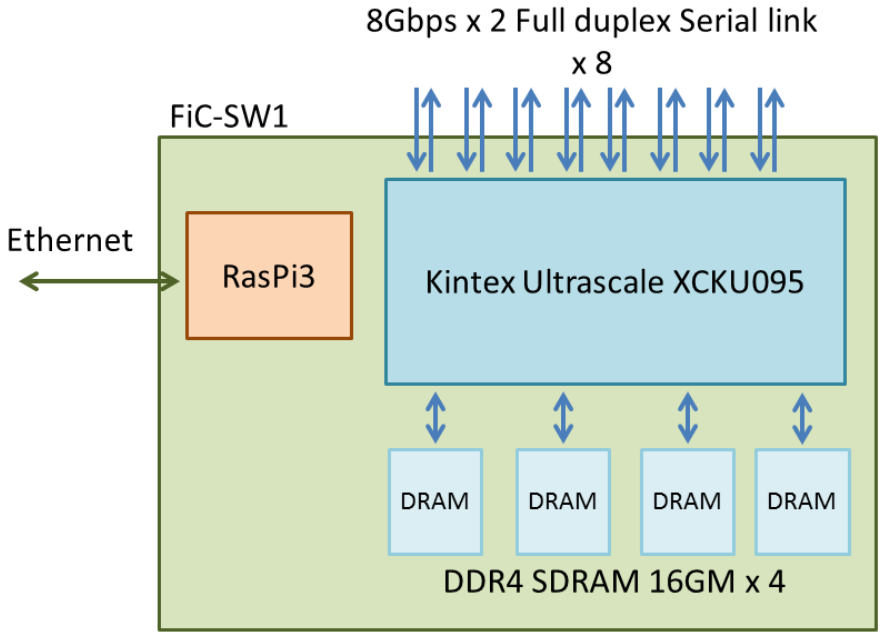


図 2.3: FiC-SW1 ボードの構成

図 2.4 は FiC-SW1 のスイッチの概要図である。このスイッチが 1 サイクルあたりに受信する転送データを格納する容量を持つバッファとして「スロット」を定義する。そして、図 2.4 にように入出力スロット間の接続によって経路を構成する。経路は 1 対 1 だけでなく、1 対多が可能で、効率的にマルチキャストできる。各ポートでは、1 サイクルごとに順番にスロットを巡回して送受信を行うことで時分割多重（TDM : Time Division Multiplexing）を実現する。各スロットの読み書きは同期を取らず、RAW ハザード (データ書き込み前にデータ読み込みをしてしまうエラー) と、WAR ハザード (データ読み込み前にデータ書き換えをしてしまうエラー) のみを保証するだけで十分である。

スイッチのポートには隣接するスイッチに繋がるものと、FPGA などの計算ノードに繋がるものが存在する。

このスイッチでは、スロットの書き込み位置によって経路が一意に決定されるよう事前に経路を設定しなければならない。入出力スロット間の接続や再接続には、FiC-SW1 内部向けに用意されたスロットへ特定の値を書き込むことで行われる。図 2.5 は、1 対 2 通信を利用してこのスロットに値を書き込み、マルチキャストへと接続を再構成している例である。このように、定期的に入出力スロット間の接続は更新することが可能となっている。ただし、内部向けのこのスロットは、FiC-SW1 上で演算を行う際の入出力が本来の目的である。

このように、現状の FiC-SW1 ボードでは光リンクと電気スイッチによってサーキットスイッチングを行う。しかし、FiC システムでは将来的に、光ネットワークを採用したより高速な相互通信を行う計画がある。そのため、この電気スイッチは、光スイッチに置き変えられることを想定して設計された。この場合、時分割多重は波長多重に置き変えられ、1 スロットは 1 チャンネルに相当することになる。

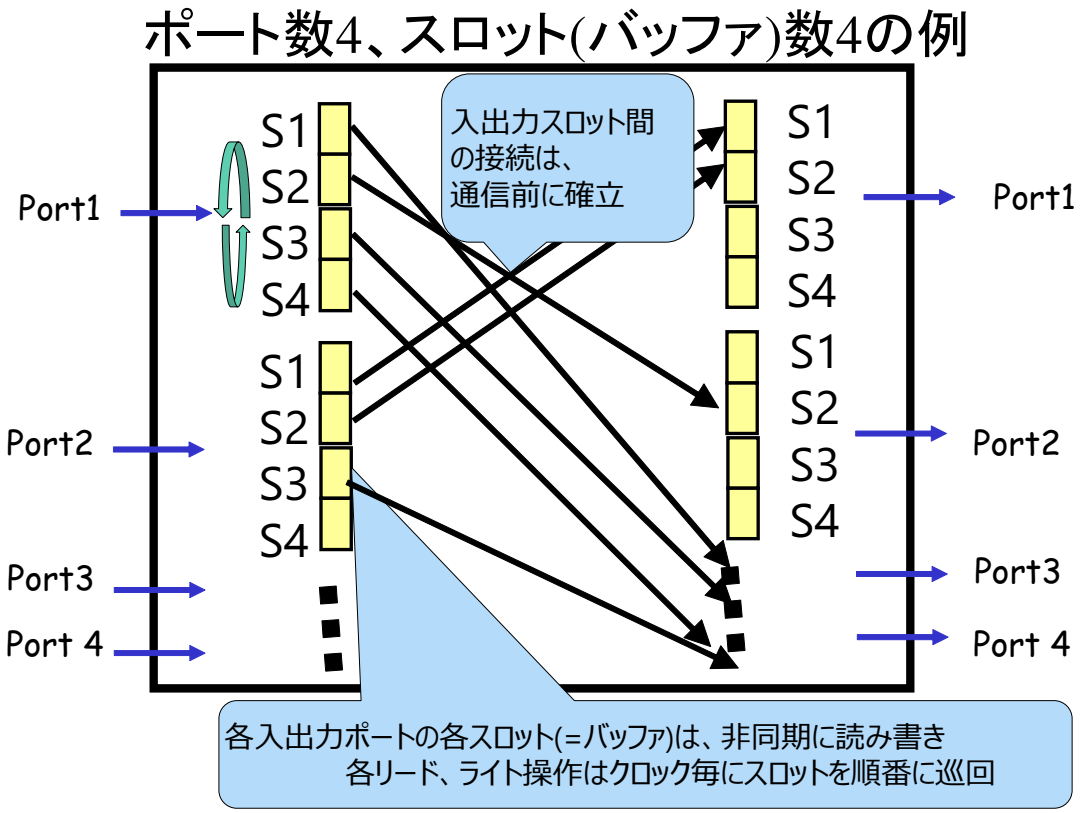
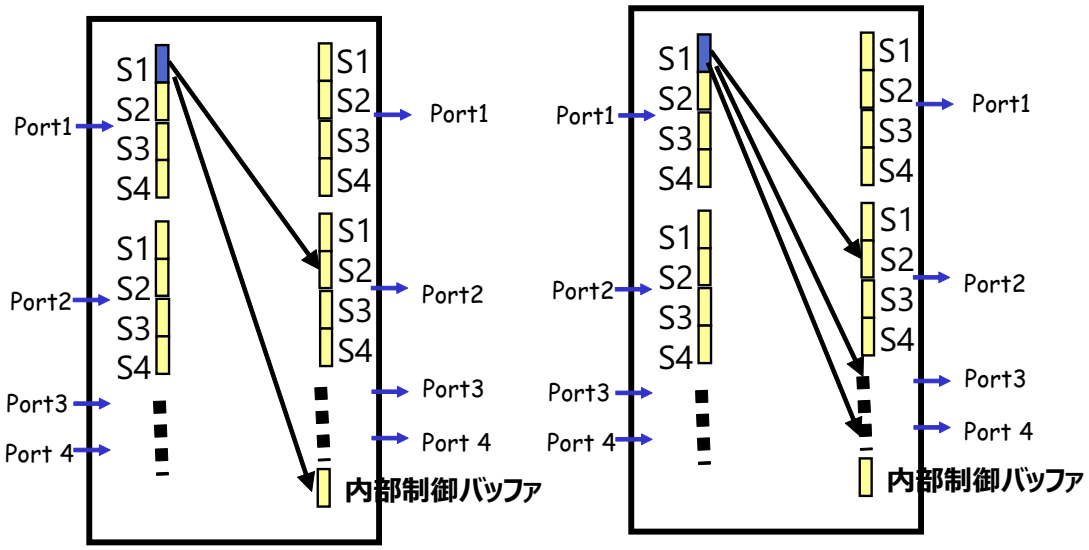


図 2.4: FiC-SW1 のサーキットスイッチ



(a) 内部制御バッファへの書き込み(1:2) (b) 他スイッチへのマルチキャストの設定例

図 2.5: 入出力スロットの再接続の例

第3章

畳み込みニューラルネットワーク（CNN:Convolutional Neural Network）

3.1 CNNの概要

この章では、FiC-SW1の予備評価のためにベンチマークアプリケーションとして実装する畳み込みニューラルネットワーク（CNN:Convolutional Neural Network、以下CNN）について解説する。

CNNは、主に画像識別に用いられる順伝搬型ネットワークの一種である。順伝搬型ネットワークは前の層と後ろの層が互いに全対全で結びつく全結合が一般的だが、CNNではカーネルと呼ばれるフィルタを導入して結合に局所性を持たせている。これにより、全結合に比べ計算処理を減らし、効率的に処理を行うことが可能となった。

図3.1に一般的な画像識別を行うCNNの推論の概要を示す。CNNでは2次元画像データのことを特徴マップと呼び、特徴マップに対してカーネルを用いて畳み込み演算を行う畳み込み層と、一定の範囲から値を絞るプーリング層の2種の層を組み合わせるのが構成と成っている。畳み込み層は、カーネルと同じパターンが画像のどこに存在するかを判定する効果がある。プーリング層は画像に写っている対象の位置感度を低下させ、位置に対する依存性を低くする効果がある。

また、ほとんどの場合において、畳み込み層は識別層の前に設けられる。これは、複数の畳み込み層、プーリング層を組み合わせることで、識別層の入力値の数を絞り込み、識別層の処理の負荷を軽くする狙いがあるからである。

さて、学習の処理では、教師データを用いて誤差逆伝搬法によって、畳み込み層と識別層の重みを繰り返し再計算していく形となる。今回扱うのはこのようなCNNの推論処理のみなので、学習処理については詳しくは扱わないこととする。

3.2 畳み込み層の処理

CNNの特徴のひとつである畳み込み層は、入力特徴マップに対してフィルタの役割を持つカーネルを用いて畳み込み演算を行い、特徴を抽出するためのレイヤーである。

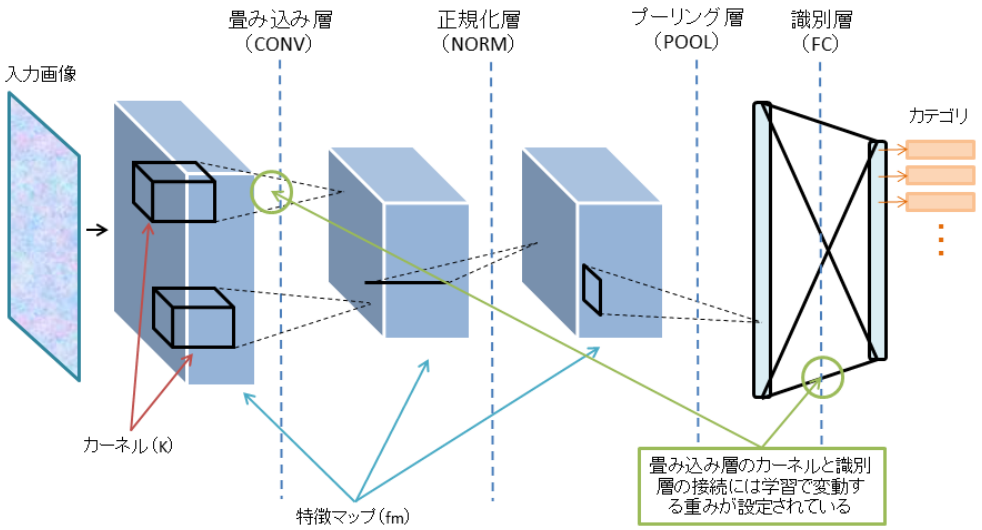


図 3.1: 畳み込みニューラルネットワークの推論

3.2.1 畳み込み演算

図 3.2 は、畳み込み層の最も重要な処理である畳み込み演算について解説したものになる。畳み込み演算の処理は画像のフィルタリングに似ており、N 次元の入力特徴マップと N 次元のカーネルの行列演算の結果を出力特徴マップの値として出力する。カーネルの範囲 K × K は、横方向にずらしていき、端まで到達したら縦方向にずらしてから同様に横方向に移動させていく。このように左上から右下へとフィルタリングした結果が 1 枚の出力特徴マップとなる。さらに、カーネルは M 枚（チャンネル）存在し、出力特徴マップもカーネルによって M 枚（チャンネル）に分かれることになる。つまり、入力特徴マップ N 枚とカーネル M × N 枚から出力特徴マップ M 枚を計算するのが畳み込み演算である。

また、畳み込み層のカーネルにはそれぞれ重みが与えられており、これが学習で変動する。つまり、M × N × K × K が重みデータの総量となる。さらに、畳み込み層には学習で変動するバイアス (bias) のデータが存在する。これは畳み込み演算終了後に対応する出力特徴マップに加算する数値で、出力特徴マップのチャンネル別に M 個のデータがある。

畳み込み演算を数式で表すと以下のようになる。

$$output(tr, tc)^{to} = \sum_{ti=0}^{N} \sum_{i=0}^{K} \sum_{j=0}^{K} weight_{to,ti}(i, j) * in(S * tr + i, S * tc + j)^{ti}$$

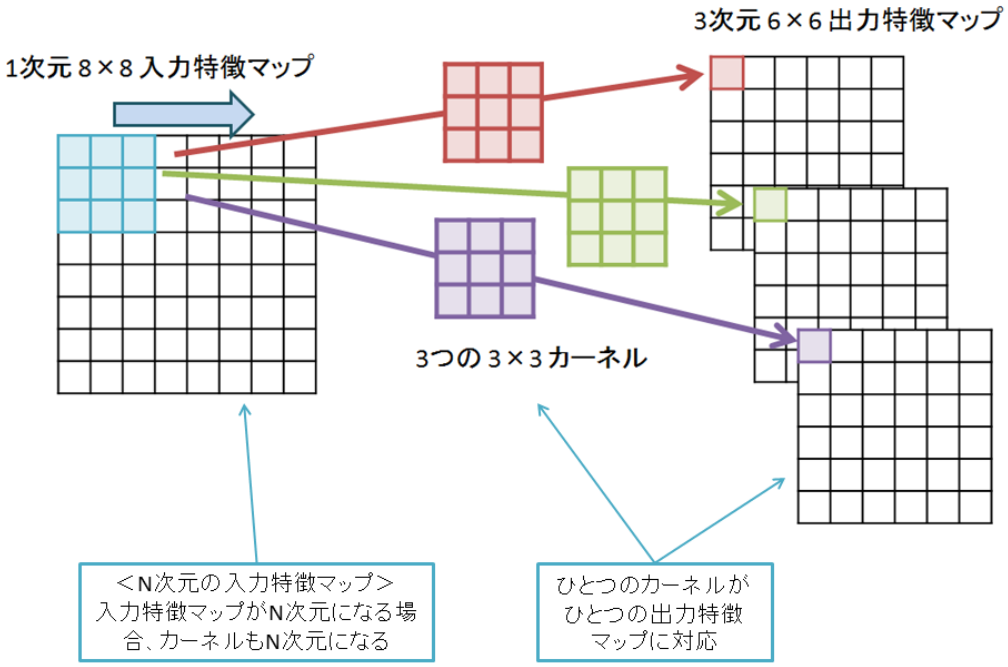


図 3.2: 畳み込み層の演算

このように、フィルタリング処理と同様、畳み込み演算は大規模な積和演算として捉えることができる。また、この演算を擬似コードで表すと以下のようになる。コードを見るとわかるように、このデータ処理では、出力特徴マップのデータ間ではデータ並列性が存在するため、並列演算器によって高速化できる余地があることがわかる。このことは今回の畳み込みアクセラレータの実装を検討する上で最も重要な事実となる。

畳み込み演算の C コード

```
for (int tr = 0; tr < R; tr++)
  for (int tc = 0; tc < C; tc++)
    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][tr][tc] +=
              input[ti][S*tr+i][S*tc+j] *
              weight[to][ti][i][j];
```

3.2.2 活性化関数

さて、畳み込み層ではもうひとつ重要な処理が行われる。それは畳み込み演算の結果を活性化関数と呼ばれる関数にかける処理である。活性化関数は畳み込み演算の結果を正規化する目的で行われる計算で、正規化線形関数 (ReLU : Rectified Liner function) やシグモイド関数などが使われる。

たとえば、最も一般的な正規化線形関数は数式にすると以下のようになる。

$$f(u) = max(u, 0)$$

この関数は他の関数に比べ、処理が単純で、入力の変動が大きさに対して寛容なため、近年のニューラルネットワークで使われる機会が多い。

3.3 その他の層の演算

今回は畳み込み層の処理を主なベンチマークとして扱うが、CNN には他にもプーリング層や正規化層、識別層が存在している。ここではそれらの層の処理について解説する。

プーリング層（POOL）では、主に特徴の位置感度を低下させる目的で、最大値プーリング (maxpooling) や平均値プーリングが行われる。たとえば、最大値プーリングでは、プーリングの範囲 P × P の中で最も大きい値を抽出し、出力とする。P × P の範囲が大きいほど特徴マップのサイズは小さくなって出力されることになる。

正規化層（NORM）では、画像のコントラストや露出の影響を抑え、識別の精度をあげる目的で設置される。ここでは局所コントラスト正規化 (LRN : Local Response Normalization) を解説する。

LRN は AlexNet の正規化層で用いられる処理で、数式で表すと以下のようになる。

$$output(x, y)^i = input(x, y)^i / (k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a_{x,y}^j)^2)^\beta$$

k、n、α、βがパラメータであり、たとえば、AlexNet の正規化層では、k=1、n=5、α=?、β=?となっている。

識別層（FC）は純粋な順伝搬型ニューラルネットワークであり、すべての入力値と出力値の間に重みとバイアスが与えられている全結合の形になっている。この重みは畳み込み層の重み・バイアスと同様、学習によって変動する。畳み込み層、プーリング層、正規化層を経て抽出さえた値はこの識別層によって最終的な分類が行われる。

識別層の処理を数式で表すと以下のようになる。

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

3.4 大規模な画像識別 CNN : AlexNet

AlexNet[1] は ImageNet2012 で発表され、現在の CNN の基礎となっている一般的な画像識別ニューラルネットワークのひとつである。5 つの畳み込み層、3 つのプーリング層、2 つの正規化層、3 つの識別層（全結合層）から成るこの CNN は、227 × 227 ピクセルのカラー画像を 1000 のカテゴリに分類することができる。図 3.3 は AlexNet の全体の流れを示している。

今回はこの AlexNet の畳み込み層の推論（順伝搬）をベンチマークアプリケーションとして扱う。表 3.1 は今回利用する AlexNet の各層のパラメータをまとめたものである。N は入力特徴マップの次元数、M は出力特徴マップの次元数、R、C は出力特徴マップサイズ、K はカーネルサイズ、S はストライド、pad はパディングのパラメータとなる。パディングとは畳み込み演算を行う前に、入力特徴マップの周りを 0 で埋める操作である。

一般的に畳み込み層は CONV、プーリング層は POOL、正規化層は NORM、識別層は FC と略され、後ろにその層が何層目にあたるかの数字をつけて区別する。

実際に 2012 年に発表された AlexNet では、CONV1、CONV2、CONV5 は独立した 2 つのグループに分けられていた。たとえば、CONV 1 の出力特徴マップは前半 48 層と後半 48 層といったように半分に分けられる。しかし、今回の実装ではそれをひとつに結合したものを扱う。AlexNet がこのような 2 つのグループに分かれていたのは、2 つの GPU で処理を分担する都合上であり、それをひとつのシステム上で処理できるならばひとつのグループにまとめてしまっても問題はないためである。

表 3.1: AlexNet の各層の詳細なパラメータ

層	入力 fm N	出力 fm M	サイズ (R,C)	カーネル K	ストライド S	パディング
CONV1	3	96	55	11	4	0
NORM1	96	96	55	-	-	-
POOL1	96	96	27	3	2	-
CONV2	96	256	27	5	1	2
NORM2	256	256	27	-	-	-
POOL2	256	256	27	3	2	-
CONV3	256	384	13	3	1	1
CONV4	384	384	13	3	1	1
CONV5	384	256	13	3	1	1
POOL5	256	256	6	3	2	-
FC6	9216	4096	-	-	-	-
FC7	4096	4096	-	-	-	-
FC8	4096	1000	-	-	-	-

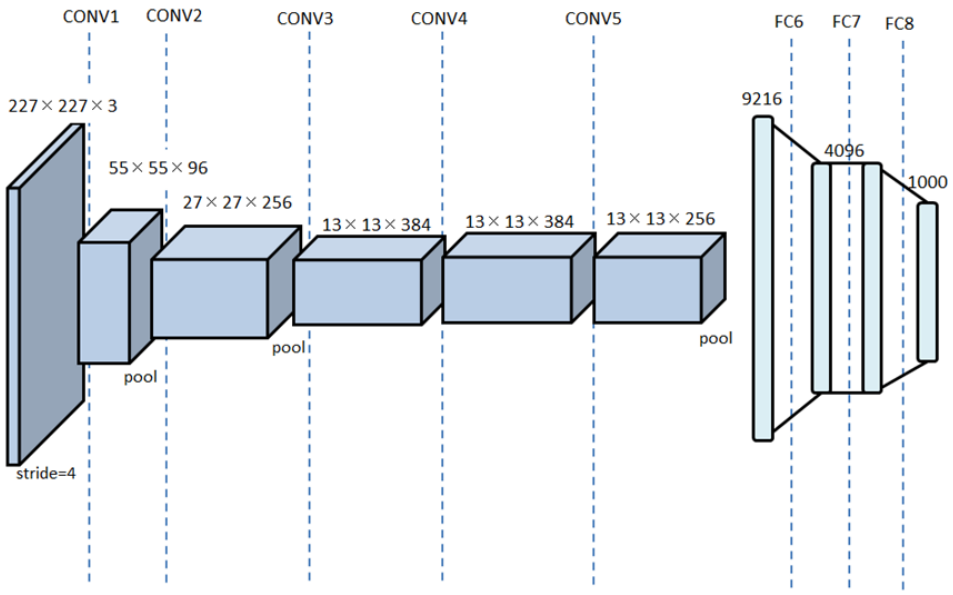


図 3.3: AlexNet[1] の概要

第4章

関連研究

4.1 マルチFPGAシステムでのボード間通信

有名なマルチFPGAシステムとしては、たとえば、Microsoft社の提供するBing検索エンジンに利用されているCatapult[2]がある。Catapultでは、ボード間の接続にSerial Attached SCSIの物理層を用いた独自のシリアルI/Oを装備している。また、EthernetやPCIeを使う例としては、Berkeley Univ.のBEE3[3]、EthernetとInfiniBandを使う例として、Imperial CollegeのAXCEL[4]がある。国内の例では、たとえば、東北大学の密結合クラスタ[5]もEthernetの物理層を利用している

これらのマルチFPGAシステムはすべてパケット交換方式によってボード間の通信を行っている点で、回線交換方式のFiCシステムとは異なっている。

4.2 大規模畳み込みニューラルネットワーク

今回の検証ではAkexNet[1]をベンチマークとして使用したが、近年注目されている比較的小規模な畳み込み層を何層も重ねた形のGoogLeNet[6]やResNet[7]についても検討しなければならない。一般的には深いネットワークのほうがパラメータ数が少なくなる傾向があり、この期もその傾向が続く場合はハードウェアデザインに大きな影響を与える可能性がある。ただし、これに関しては反論[8]もある。

4.3 FPGAベースのCNNアクセラレータ

CNN、もしくはより汎用的なニューラルネットワークの処理を大規模なシステムによって高速化、省電力化する代表的な研究としてDaDianNaoマシンラーニングスーパーコンピュータ[9]が挙げられる。これは推論学習のミリ秒単位での高速化に焦点を当てたプロジェクトで、重みデータを可能な限りローカルに保存できるような設計をしている。それによってメモリ帯域幅のボトルネックを改善するというコンセプトではFiCプロジェクトとも関連が強い。ただし、[9]はASICチップシステムであり、FPGA-GPUシステムを用いる点では異なっている。

CNNに対するFPGAを用いた高速化の最初期の研究としては、2009年のCNP[10]が挙げられる。これはソフトウェア実行を中心としながら、畳み込み演算のフィルタリング

処理をハードウェアで高速化したものである。2010 年には FPGA 上で完全な CNN 実装 [11][12][13] が行えるようになった。

現在、FPGA ベースの CNN アクセラレータは UCLA のマルチレイヤー CNN アクセラレータ [14] が有名である。今回実装を行ったアクセラレータの演算部はこの設計を参考にした。

他にも、Microsoft 社の Catapult の FPGA で動作する CNN アクセラレータ [15] が挙げられる。この Catapult のアクセラレータは ULCA の設計を参考に、Catapult サーバと Aria10 を組み合わせて 3 倍以上の性能向上を実現している。

第5章

目的

本稿は、FiC-SW1 ボード上の FPGA に CNN の畳み込み層の処理をする並列畳み込みアクセラレータを実装し、その計算時間と通信時間の割合をシミュレーションによって測定することが目的である。そのためには、以下の 2 つの課題に取り組まなくてはならない。畳み込み層の並列化と相互通信の検討については第 6 章、畳み込みアクセラレータの実装については第 7 章で扱う。

畳み込み層の並列化と相互通信の検討

大規模な畳み込み演算を複数の FiC-SW1 ボード上で処理するために最適なタスク分割を行う。また、畳み込み演算を分割処理するにあたって相互通信が必要になる。その相互通信量と時間コストを静的に計算する。

畳み込みアクセラレータの実装

分割された畳み込み演算を高速に行うために、FiC-SW1 ボード上の FPGA に畳み込みアクセラレータの実装を行う。

第6章

畳み込み層並列化と相互通信についての検討

6.1 概要

この章では、まず、マルチ FiC-SW1 システム上で CNN を扱うために畳み込み層の並列化を行う。そして、その並列畳み込み演算を FiC-SW1 上に実装した場合の相互通信について検討する。

[14] によれば、識別時において、畳み込み層での演算は全体の実行時間の 90 %以上を占めることがわかっている。今回の実装では特にこの畳み込み演算をマルチ FiC-SW1 システムによって高速化することを重要な課題とした。

6.2 畳み込み層の並列化

畳み込み層の演算は多くの場合において、高いデータ並列性があるため、様々な並列化方式を考えることができる。

今回は、ほかの多くのニューラルネットアクセラレータ [14][9] と同様に、入出力の特徴マップを相互通信し、重みデータをローカルストレージに保存する戦略をとった。これは、特徴マップの計算量が (M × R × C) であるのに対して重みは (M × N × K × K) で、一般的な CNN では重みのほうが計算量が多くなる傾向があり、特徴マップを相互通信するほうが低コストになるからである。

図 6.1 は CONV1 を例に、今回の戦略で畳み込み層を 16 並列化したときのタスク分割の様子を示している。AlexNet の CONV1 は [3, 227, 227] のサイズの入力特徴マップから、[96, 3, 11, 11] のサイズの重みを用いて、[96, 55, 55] のサイズの特徴マップを出力するものである。これを 16 並列化すると、重みが 16 分割されるため、1 ノードあたりの重みは [6, 3, 11, 11] となる。それに伴って、出力特徴マップのサイズも 16 分の 1 になるため、[6, 55, 55] となる。ただし、この 16 分の 1 の出力は次の層のために全対全の相互通信によって共有しなければならない。

図 6.2 は AlexNet を並列化した際に、どのタイミングで特徴マップ共有のための全対全通信が必要になるかを示している。基本的には、ひとつの畳み込み層・識別層に対して 1 回の共有をしなければならない。また、正規化層の前でも共有が必要になるが、これは LRN

の演算で使う前後 2 チャンネル分の特徴マップに対してのみであり、全対全通信と比べて遥かに小さいため、今回は詳しくは扱わない。

以下は、並列化された畳み込み演算を疑似コードで表したものになる。Tm の値によって並列度が変化し、たとえば、M=96 の CONV1 を 16 並列化した場合、Tm=6 となる。

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 tr = 0; tr < R; tr++)
  for (int tc = 0; tc < C; tc++)
    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][tr][tc] +=
              input[ti][S*tr+i][S*tc+j] *
              weight[to][ti][i][j];
```

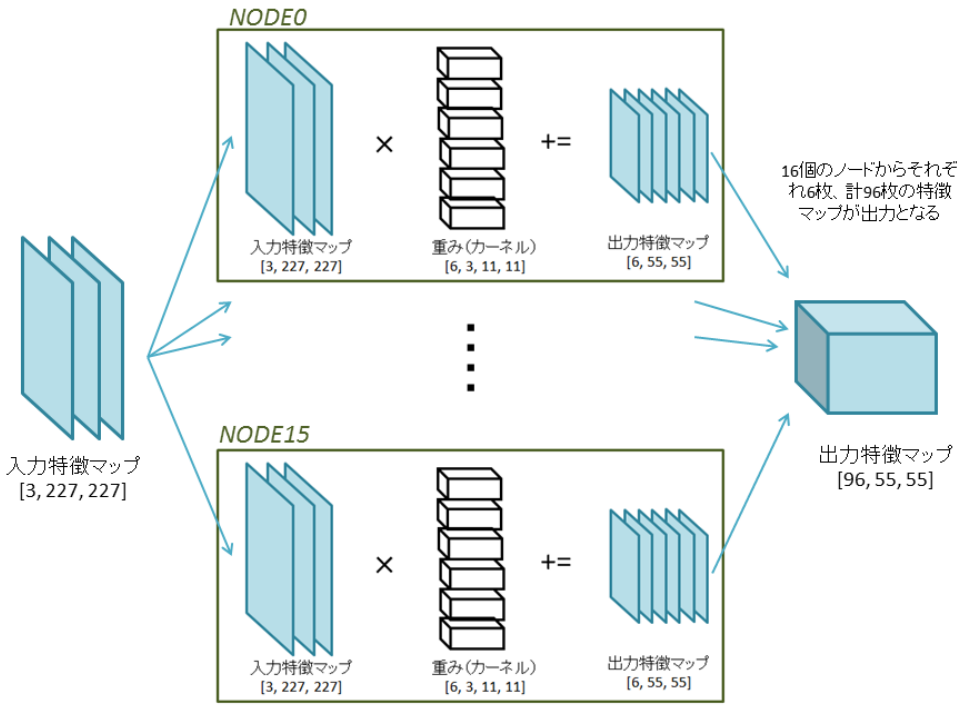


図 6.1: 16 並列化された CONV1 の畳み込み演算の例

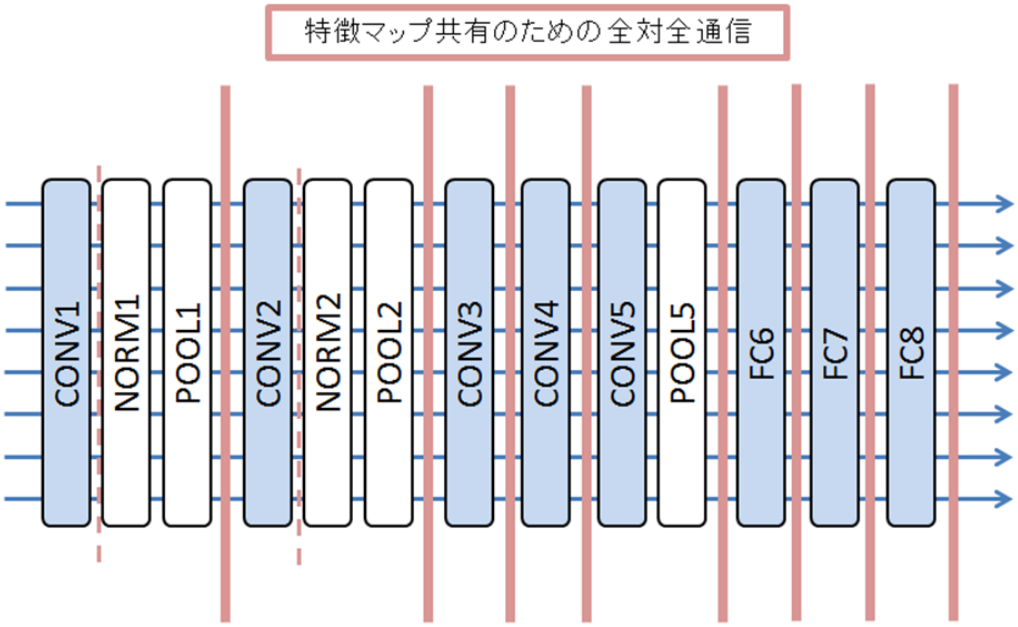


図 6.2: 並列化された AlexNet における特徴マップ共有のタイミング

6.3 マルチ FiC-SW1 システム上での並列化された畳み込み層の扱い

さて、並列畳み込み演算をマルチ FiC-SW1 システム上での処理に置き換えると、1 ノードをひとつの FiC-SW1 ボードに置き換えるのが妥当である。重みは FiC-SW1 上の FPGA のメモリ資源か DRAM をローカルストレージとして用いて保存する。また、入出力特徴マップのためもバッファを設けて、必要に応じてサーキットスイッチネットワークを利用してブロードキャストする形になる。

図 6.3 は FiC-SW1 ボード上での畳み込み層の演算について解説している。まず、演算に必要な入力特徴マップがブロードキャストによって送信されてくるまで待機しなければならない。データが揃ったならローカルストレージの重みを用いて畳み込み演算を行い、その間はサーキットスイッチからは空データを受信し続ける。計算が終わったら、その出力特徴マップをブロードキャストする。すべての FiC-SW1 は基本的には非同期的に動作するが、すべてのノードがブロードキャストし終えないと次の層の処理を始めることができないため、このタイミングで同期が取られる。

さて、表 6.1 に、このシステムで AlexNet を実装した際のひとつの FiC-SW1 あたりの出力特徴マップ数とブロードキャストされるデータ量を算出した結果を示す。並列度が高くなるほど 1 ノードが扱う出力特徴マップ数が減っていることがわかる。しかし、たとえば、CONV1 の結果を見ると、16 並列から 64 並列では並列度は 4 倍になっているのに対して、

出力特徴マップ数は 6 チャンネルから 2 チャンネルと 4 分の 1 にはなっていない。CONV1 の出力特徴マップ数 96 は、64 で割り切れないため、最適なタスク分割を行えていないことが理由である。このように畳み込み層の出力特徴マップ数によっては、過剰な並列化は通信の無駄を増やしてしまうことに注意しなければならない。

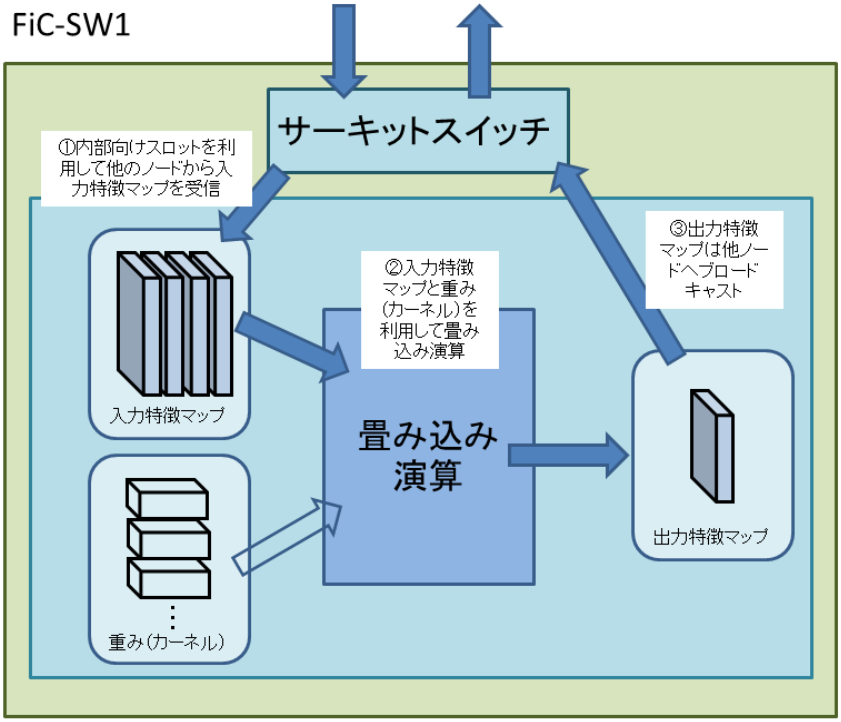


図 6.3: FiC-SW1 上の CNN の計算処理の流れ

6.4 FiC-SW1 ネットワークによる特徴マップ共有

図 6.4 は、表 6.1 を基に、出力特徴マップを FiC-SW1 の時分割多重のサーキットスイッチネットワークによって共有した場合の通信時間の予測を示す。表 6.2 は図 6.4 の詳細をまとめたものである。出力特徴マップのサイズ（R、C）が他の層と比べて大きい CONV1 では 16 並列時で 699.84μsec の通信時間が発生するが、特徴マップのサイズが小さくなるにつれ減少していき、最後の FC8 では 10.08μsec となる。この計算では、FiC-SW1 のサーキットスイッチのスロットひとつに、ひとつの特徴マップの値を格納した場合を仮定している。実際の FiC-SW1 のスロットは 80bit 程度になるので 32bit 浮動小数点数データを 2 つ格納できる可能性が高いが、今回はアクセラレータに合わせて 1 サイクルにつきデータひとつの場合を扱った。

表 6.1: マルチノード CNN の 1 ノードあたりの出力特徴マップ数とそのデータ量

	4 並列		16 並列		64 並列		265 並列	
	fm	(Kb)	fm	(Kb)	fm	(Kb)	fm	(Kb)
CONV1	24	559.9	6	140.0	2	46.6	1	23.3
CONV2	64	346.1	16	86.5	4	21.6	1	5.4
CONV3	96	519.2	24	129.8	6	32.4	2	10.8
CONV4	96	519.2	24	129.8	6	32.4	2	10.8
CONV5	64	73.7	16	18.4	4	4.6	1	1.2
FC6	1024	32.8	256	8.2	64	2.0	16	0.5
FC7	1024	32.8	256	8.2	64	2.0	16	0.5
FC8	250	8.0	63	2.0	16	0.5	4	0.1

時分割多重方式による通信ではノード数が増えるほど通信帯域が狭くなるが、今回のシステムではそのノード数が増えれば 1 ノードあたりの送信量も減るため、結果としてノード数に対して依存しにくい通信時間となる。しかし、表 6.1 に見られるように、過剰な並列度では最適なタスク分割を行うことができず、通信帯域を狭めるだけになってしまい共有にかかる時間コストが増大してしまう。

これらの結果は、すべてのノードが 16Gbps(8Gbps 全二重) の通信路を最大限利用できたときの、理想的状態を仮定したときのものである。また、ネットワークのトポロジ、通信遅延などは考慮していない。これらを含めた通信時間予測は今後の課題とするが、このような要因によって通信時間は今回の結果より大きくなってしまうことが予想される。

表 6.2: 特徴マップ共有の予測通信時間 (μsec)

層	4 並列	16 並列	64 並列	256 並列
CONV1	699.84	699.84	933.12	1866.24
CONV2	432.64	432.649	432.64	432.64
CONV3	648.96	648.96	648.96	865.28
CONV4	648.96	648.96	648.96	865.28
CONV5	92.16	92.16	92.16	92.16
FC6	40.96	40.96	40.96	40.96
FC7	40.96	40.96	40.96	40.96
FC8	10.00	10.08	10.24	10.24

6.5 FiC-SW1 ネットワークの通信帯域改善のための手法

STDM によるサーキットスイッチングネットワークでは、通信の帯域を保証できるというメリットがあるが、ノード数が増えると 1 ノードに与えられる通信帯域が極端に狭くなってしまうという問題が発生する。

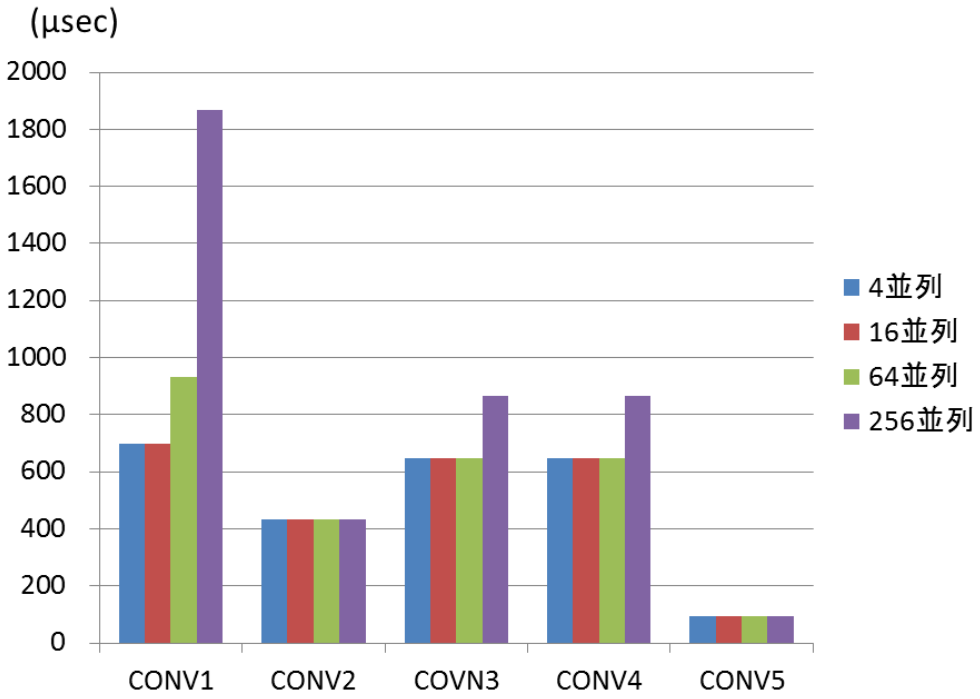


図 6.4: 畳み込み層の特徴マップ共有にかかる通信時間 (μsec) の予測

時分割多重のスロット数を減らすことができれば、1 スロットを複数のノードで共有する方針で、帯域を改善させることができる。その単純な方法としては、データのヘッダ情報として、ラベルを定義する方法がある。スロットを共有しているノードにはそれぞれ固有のラベル番号を振り、送信の際には自身のラベル番号と共にデータを送信するように設計することで、帯域を狭めることなく疑似的にスロット数を増やすことが可能である。ただし、この帯域の改善方法では 1 ノードがスロットを占有している間、そのスロットを共有している他のノードは送信を行うことができない。そのため、各ノードのスロット占有時間が短い場合でないと有効に働かないと考えられる。

第7章

畳み込みアクセラレータの実装

7.1 概要

この章では FiC-SW1 ボード上の FPGA に実装される畳み込みアクセラレータについて解説を行う。この畳み込みアクセラレータは、図 6.3 のような並列畳み込み演算をより効率的に行う目的で実装する。アクセラレータの大まかな設計としては、図 6.3 の処理と同様に、前の層からすべての特徴マップを入力として受け取り、分割された重みデータで並列演算し、次の層へ特徴マップを出力するという方針を取る。

また、アクセラレータ内のすべての特徴マップ・重み・バイアスのデータは 32bit の浮動小数点数で扱うものとする。

FPGA 上の演算・メモリのリソースは有限なため、効率的にリソースを使用して実行サイクル数を減らさなくてはならない。今回の設計では、アクセラレータ間でのリソース共有は行わず、各畳み込み層とアクセラレータモジュールは完全な 1 対 1 の関係になるものとした。リソース共有を行うクロスレイヤーな設計では、並列演算性能とリソース消費をトレードオフにすることができるため、リソース消費を抑えた設計を考える場合には有効である。

7.2 アクセラレータモジュール

図 7.1 に、今回実装したアクセラレータモジュールの概略図を示す。アクセラレータモジュールは大きく分けて、演算モジュール、バッファ、入出力ストリームでの 3 つのパートから構成される。

畳み込み演算の核となる演算モジュールは、パイプライン化された並列積和演算器（PE : Processing Elements）であり、毎サイクル、複数チャンネルの特徴マップを出力できる。この演算モジュールは、優れた計算効率を出した UCLA の Multi-Layer CNN Accelerator[14] の設計を参考にした。

アクセラレータモジュール内のバッファは特徴マップのための入出力バッファと、重みのための重みバッファの 2 種類がある。入出力バッファは、自身の演算に必要な特徴マップを格納するためのストレージで、それぞれのバッファはそれぞれの特徴マップをすべて格納できるサイズを持つ。また、演算モジュールに毎サイクル特徴マップを入出力するために、複数個のバンクに分けることで帯域幅を確保している。重みバッファはカーネルを

格納するためのストレージで、並列化によって分割された重みのすべてを格納できるサイズと、広い帯域幅を持つ。

アクセラレータモジュールの入出力には、Xilinx 社の IP である AXI4-Stream を利用した。AXI4-Stream では、毎サイクル Nbit のデータをストリーム形式で送受信行うことができる。今回は 1 サイクルで 1 個の 32bit 浮動小数点数データを送受信行える、N=32bit でのストリームとした。

[14] では高位合成を用いてスケーラブルに実装可能なアクセラレータの設計を提示している。リソースと実行時間をトレードオフにすることができる設計を用いることで、リソース制約下で畳み込み演算の並列性を最大限活用でき、効率的なハードウェアにすることができると考えられる。

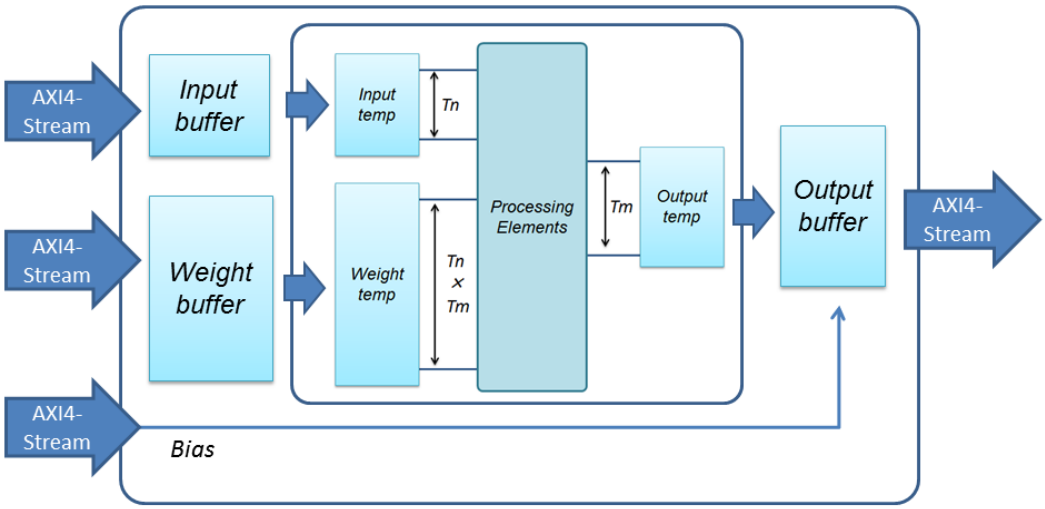


図 7.1: アクセラレータの概略図

7.2.1 演算モジュール

この節では畳み込み演算の並列性に沿って最適化された演算モジュールを設計する。演算モジュールは、実際に計算を行う並列積和演算器 PEs と、そのための一時バッファ(temp buffer) から構成される。

図 7.2 は並列積和演算器 PE の概要である。図 3.2 のように、畳み込み層の特徴のひとつとして、その多くの処理が入力と重みの積和演算から構成されるという点が挙げられる。この処理の高速化には複数個の積和演算器を並列に配置するのが最も単純で効率的な方法だと考えられる。したがって、演算モジュールは Tn 個の入力値から Tn × Tm 個の重みを用いて T 個の出力値を並列に計算できる汎用的な設計となった。さらに、スループット向

上のために PE 内部はパイプライン化されており、毎サイクル Tm 個の値を出力バッファへと出力することができる。

パラメータ Tm、Tn はそれぞれ、PEs が同時に扱う入力データ、出力データの数であり、リソース消費量とトレードオフ関係にあるため実装最適化の対象となる。今回の設計では暫定的に演算モジュールのサイズは（Tm, Tn）=(24, 8) に固定したが、リソース上限によってはこのパラメータを変更することでリソース上限内に消費量を抑えることが必要となる。

PEs の入出力は演算モジュール内の一時バッファ（temp buffer）に保存される。一時バッファは、入出力バッファ・重みバッファと PEs の間に入り、PEs が入力データ Tn 個、重み Tn × Tm 個、出力データ Tm 個を高速に読み書きを行うためのレジスタの役割を持つ。演算モジュール外部から見れば、それぞれ対応する一時バッファのデータを読み書きすることで、計算が行われるという流れになる。

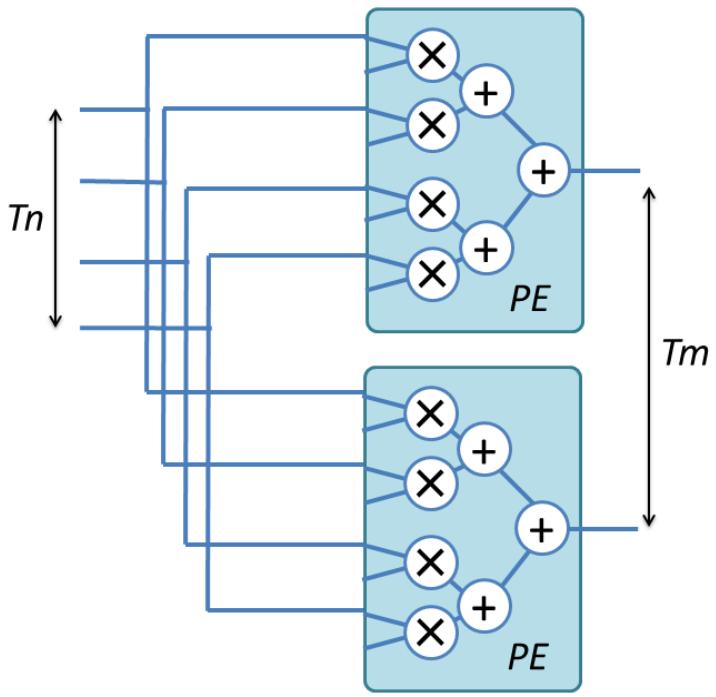


図 7.2: 並列積和演算器 PEs

7.2.2 入出力バッファと重みバッファ

スイッチモジュールから送られてきた入力特徴マップのデータは、まず FPGA 上に確保された入力バッファに蓄えられる。

高速に畳み込み演算を行うためには、演算モジュールは可能な限り広い帯域幅で入力特

徴マップにアクセスできなけらばならない。そこで、入力バッファを特徴マップごとにバンクを分割し並列に読み込むことができるようにした。これにより、入力バッファと演算モジュールの帯域幅は Tn/バンク数となり、帯域幅を改善することができる。同様の手法で、出力バッファや重みバッファも帯域幅を増減させることができる。。

今回の実装では、入力バッファは Tn 個、出力バッファは Tm 個、重みバッファは Tm × Tn 個に分割することで、最大となるような帯域幅を確保した。これにより、演算モジュールが毎サイクル、畳み込み演算を行う効率的な機構を実現した。ただし、この実装を行う場合、バンク数に応じて FPGA リソース消費量が増大するため、過剰なバンク分割によってリソースを浪費することに注意しなければならない。

図 7.3 に、入出力バッファ・重みバッファを演算モジュールと接続した概略図を示す。バッファが特徴マップの次元数によって、入力バッファは Tn 個、出力バッファは Tm 個、重みバッファは Tm × Tn 個のバンクに分割されている様子がわかる。

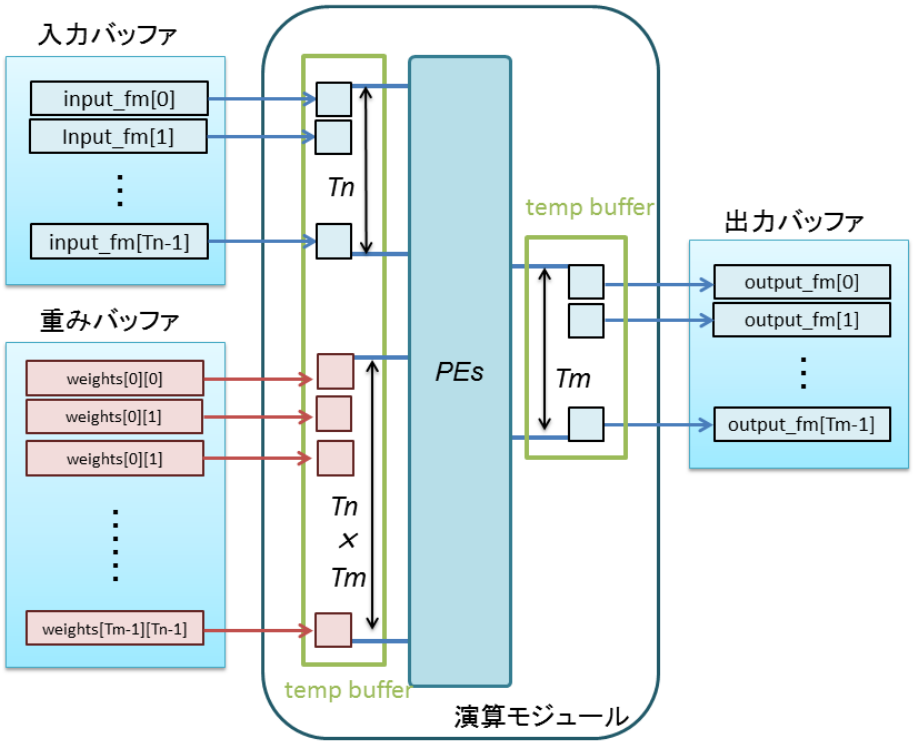


図 7.3: 入出力バッファ・重みバッファと演算モジュールの接続

第8章

アクセラレータの評価

8.1 評価環境

今回実装するアクセラレータは表 8.1 に示す環境で設計を行い、評価を行った。Vivado HLS は Xilinx 社の提供する高位合成ツールで、畳み込みアクセラレータのシミュレーションはこのツール上で行った。評価ボードと FPGA には FiC-SW1 ボードのものと同様に Kintex UltraScale XCKU095 を用いた。

また、比較のために CPU で畳み込み演算のソフトウェア実行を行った。そのときのソフトウェア実行環境を表 8.2 に示す。

表 8.1: CNN アクセラレータの評価環境

ツール	Vivado HLS Version 2016.1 (Xilinx)
評価ボード	Kintex UltraScale XCKU095 (Xilinx)
FPGA	XCKU095-FFVB2104 (Xilinx)

表 8.2: ソフトウェア実行環境

CPU	Intel Core i5-4250U
動作周波数	1.30GHz
コンパイラ	GCC 4.4.7 20120313

8.2 アクセラレータのクロックサイクル数とリソース消費量

演算モジュールの演算器サイズを (Tm, Tn）= (24, 8) としたときの各畳み込み層の計算時間を測定した。Tm は畳み込み層 5 層のなかでの M の最大値である 24 を、Tn は予想されるリソース消費量が上限以下になるように暫定的に決定した。表 6.2 からノード数を 16 より大きくすると通信時間が増えることがわかるため、今回の実装では通信時間が最小の条件下でノード数が最も高い 16 並列時を取り扱う。

表 8.3 は、CONV1 から CONV5 を 16 並列で計算した際のサイクル数の結果となる。特に、入出力特徴マップのサイズが大きく、次元数が低い CONV1 が 508118 サイクルと最もクロックサイクル数が大きい。今回のアクセラレータは入出力の特徴マップの次元数以上に処理を並列化していない設計のため、このような結果となったと考えられる。

また、このときのリソース消費量を図 8.1 に示す。表 8.4 は図 8.1 の詳細な結果である。それぞれの畳み込み層のアクセラレータのリソース消費量は上限に収まっているため、（Tm, Tn）=(24, 8) の演算モジュールのサイズは（最適とはいかないまでも）妥当な値であったと言える。ただし、単純にアクセラレータを 5 層分実装するにはリソースが足りないため、各アクセラレータの（Tm, Tn）を調節して規模を縮小させるか、リソース共有を行うクロスレイヤーな設計を考える必要がある。

表 8.3: 16 並列時の畳み込みアクセラレータの演算サイクル数

	サイクル数
CONV1	508118
CONV2	409746
CONV3	166554
CONV4	245434
CONV5	242506

表 8.4: 16 並列時の畳み込みアクセラレータのリソース消費量

	BRAM 18K	DSP48E	FF	LUT
CONV1	903	32	11388	181047
CONV2	496	66	80886	249112
CONV3	664	98	135466	290533
CONV4	736	98	152868	321012
CONV5	504	66	119670	244851
Avaiable	3360	768	1075200	537600

8.3 FiC-SW1 での畳み込み層の計算時間と通信時間

FiC-SW1 上の Kintex UltraScale KU095 は 100MHz で動作するため、クロックサイクル数から計算時間を計算できる。表 8.3 と表 6.2 から、16 並列時の畳み込み層の計算時間と通信時間の比較を、図 8.2 に算出した。表 8.5 は図 8.2 の詳細な結果である。通信時間に対する計算時間の比率は、約 2 倍から約 10 倍、CONV5 から識別層 FC6 の間では約 26 倍になるという結果になった。CONV5 の比率が高くなっているのは、CONV5 から FC6 の間でプーリング層 POOL5 によって特徴マップのデータが大幅に削減され、特徴アップ共有の通信にかかる時間が短縮されたことが原因である。

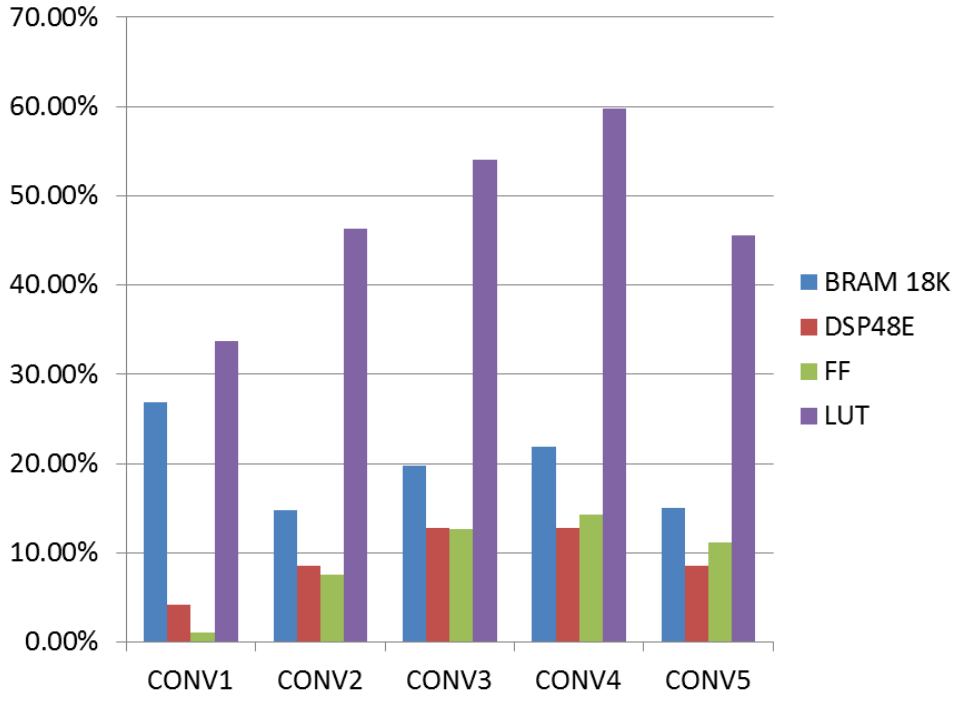


図 8.1: 16 並列時の畳み込みアクセラレータのリソース割合

通信時間の計算には、ネットワークのトポロジ、スイッチの遅延、ヘッダ付与による帯域減少は考慮していないため、実際の通信時間は今回の試算より大きくなると予測される。また、今回のアクセラレータは単純な設計なため、より複雑で高性能な設計（ストリーム機構の見直し、ダブルバッファの導入など）を施すことで実行サイクル数も小さくできる可能性が残っている。それに伴って通信時間に対する計算時間の比率も小さくなると推測される。

計算時間と通信時間の合計が畳み込み演算の実行時間である。図 8.3 では、今回の FiC-SW1 の畳み込みアクセラーレータでの実行時間と、一般的な CPU や UCLA のマルチレイヤー CNN アクセラレータでの実行時間と比較した。表 8.6 は図 8.3 の詳細な結果である。CPU でのソフトウェア実行は表 8.2 に従い、Intel Core i5-4250U(1.60GHz) を使用した。FiC-SW1 の畳み込みアクセラレータは、CPU に比べて 658 倍高速であることがわかる。これは UCLA の設計 [14] と比べて半分程度の性能に留まる。しかし、今回の設計では UCLA が行っているような演算器サイズやバッファリングの最適化は一切行っておらず、これを行うことで性能が大きく向上する可能性が残っている。アクセラレータの最適化は今後の重要な課題とする。

表 8.5: 16 並列時の通信時間に対する計算時間の比率

	計算時間 (μsec)	通信時間 (μsec)	比率
CONV1	5081.18	699.84	7.26
CONV2	4097.46	432.64	9.47
CONV3	1665.54	648.96	2.57
CONV4	2454.34	648.96	3.78
CONV5	2425.06	92.16	26.31

表 8.6: 畳み込み演算の実行時間

	CPU(msec)	UCLA[14](msec)	FiC-SW1(msec)
CONV1	1110 (192.01x)	3.66 (0.63x)	5.78
CONV2	5170 (1141.26x)	2.37 (0.52x)	4.53
CONV3	1520 (656.73x)	1.60 (0.69x)	2.31
CONV4	2530 (815.26x)	1.20 (0.39x)	3.10
CONV5	1670 (663.43x)	0.80 (0.32x)	2.52
Sum	12000 (657.67x)	9.64 (0.53x)	18.25

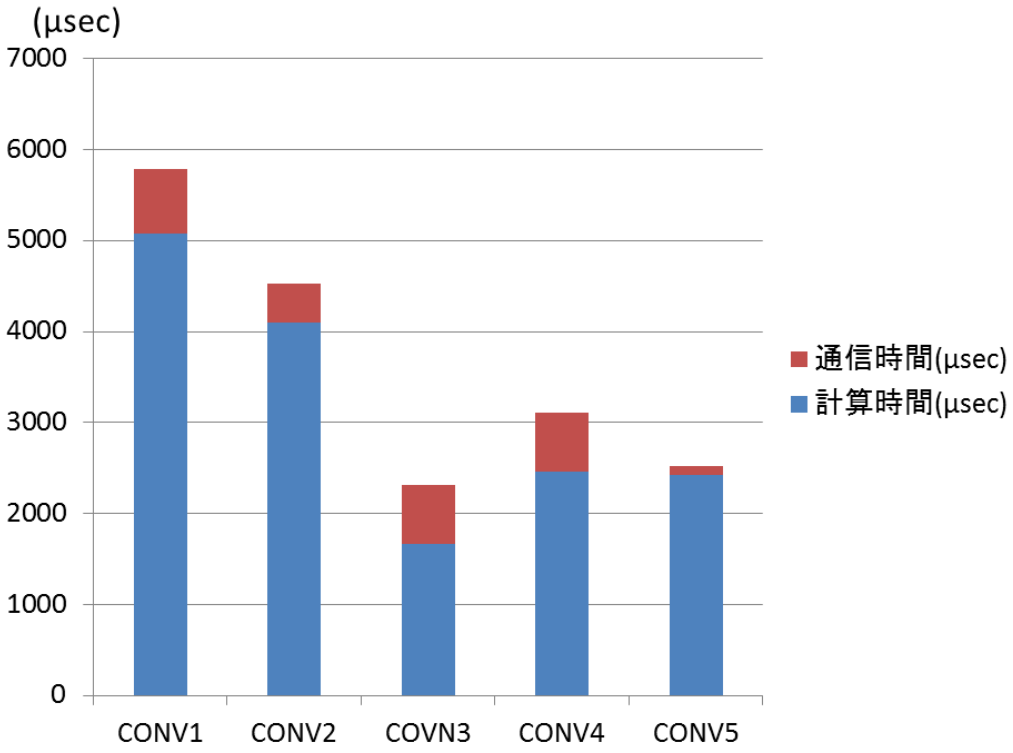


図 8.2: 16 並列時の畳み込み層の計算時間と通信時間の比較

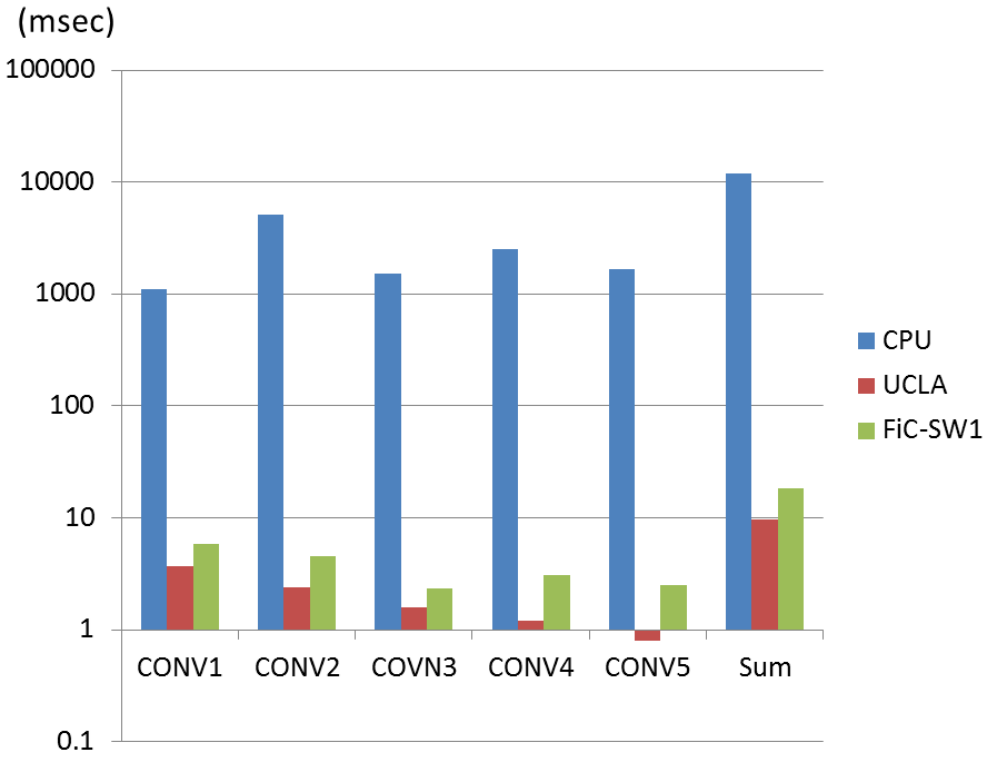


図 8.3: 畳み込み演算の実行時間の比較

第9章

結論

FiC(Flow-in-Cloud) システムは、NEDO 人工知能プロジェクトにおいて、FPGA ノード、GPU ノード、メモリノードなどの異種ノードを多数組み合わせた大規模計算システムである。この FiC システム上で、スイッチノードしての役割を持つ FiC-SW1 は高速リンクを多数接続した FPGA ボードである。本稿では、この FiC システム上でスイッチノードとしての役割と、初期のソフトウェア開発用テストベッドの役割を持つ FiC-SW1 をボードの計算性能と転送性能の予備評価を行う目的で、大規模 CNN の AlexNet をベンチマークとして、畳み込み演算の並列性について検討し、FIC-SW1 上の FPGA に並列に畳み込み演算を行う畳み込みアクセラレータを実装した。

その結果、最適化を施していないアクセラレータでも一般的な CPU に比べて 658 倍高速化することに成功した。また、通信時間に対する計算時間の比率は、2 倍から 10 倍、最大で 26 倍になることがわかった。ただし、通信時間の算出ではヘッダや通信遅延などを考慮していないため、実際の比率はこれより小さくなると予測される。

以下に今後の課題を述べる。本稿では、並列化・高速化の対象を CNN の畳み込み層の畳み込み演算のみに限って検証を行ったが、CNN のプーリング層、正規化層、識別層についてもどのように並列化・高速化を行うか検討しなければならない。また、今回実装した畳み込みアクセラレータは比較的単純な構造をしており、ダブルバッファリングなどの手法によってさらなる低リソース化・高速化できる余地が残っている。アクセラレータを単純に 5 つ実装するにはリソースが足りないため、アクセラレータの改良や FPGA 内でリソース共有を行うことも検討する必要もある。

謝辞

本研究を行うにあたり、慶應義塾大学の天野英晴教授には懇切なご指導ご鞭撻を賜りましたこと心より感謝いたします。天野教授には主査も行っていただきました。

同研究室の志村英樹先輩には副査を行っていただいた他、高位合成ツールの使い方のご教授をはじめ、大変お世話になりました。

また、この成果（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。

プロジェクト内で多岐にわたりご指導いただいた東京大学の工藤知宏教授、国立情報学研究所の鯉渕道紘教授に深くお礼申し上げます。今回、NEDO 人工知能プロジェクトに関わることができ大変光栄に思います。

参考文献

[1] Ilya Sutskever Alex Krizhevsky and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, 2012.

[2] A.Putnam et.al. A reconfigurable fabric for accelerating large-scale datacenter services. *Proc. ISCA2014*, 2014.

[3] Andreas Koch Sascha Muehlbach. A scalable multi-fpga platform for complex networking applications. *Proc. FCCM '11*, 2011.

[4] K.H. Tsoi X.Y. Niu and W.Luk. Reconfiguring distributed applications in fpga accelerated cluster with wireless networking. *Proc. Field Programmable Logic and Applications (FPL)*, 2011.

[5] K.Sano Y.Kono and S. Yamamoto. Scalability analysis of tightly-coupled fpga-cluster for lattice boltzmann computation. *Proc. Field Programmable Logic and Applications (FPL)*, 2012.

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

[7] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition. *Computer Vision and Pattern Recognition (CVPR) 2016*, 2016.

[8] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[9] Tao Luo Yunji Chen and Shaoli Liu. Dadiannao: A machine-learning supercomputer. *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, 2014.

[10] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun. Cnp: An fpga-based processor for convolutional networks. *2009 International Conference on Field Programmable Logic and Applications*, pp. 32–37, Aug 2009.

[11] Srihari Cadambi, Abhinandan Majumdar, Michela Becchi, Srimat Chakradhar, and Hans Peter Graf. A programmable parallel accelerator for learning and classification. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pp. 273–284, New York, NY, USA, 2010. ACM.

[12] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. *SIGARCH Comput. Archit. News*, Vol. 38, No. 3, pp. 247–257, June 2010.

[13] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf. A massively parallel coprocessor for convolutional neural networks. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 53–60, July 2009.

[14] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. *FPGA '15 Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arraysf*, 2015.

[15] Joo-Young Kim Jeremy Fowers Karin Strauss Eric Chung Kalin Ovtcharov, Olatunji Ruwase. Accelerating deep convolutional neural networks using specialized hardware, February 2015.