一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
CPSY2018-116,DC2018-98（2019-03）

マルチFPGAにおける畳み込みニューラルネットワークの実装手法

比留間葵† 山内脩吾† 武者一嵯† 工藤知宏†† 天野 英晴†

† 慶應義塾大学
†† 東京大学情報基盤センター
E-mail: †{hirumaruma,yyamauchi,mushak,hunga}@am.ics.keio.ac.jp, ††kudoh@nc.u-tokyo.ac.jp

あらまし FiC(Flow-in-Cloud)-SW は深層学習アプリケーション向けの FPGA ベースのスイッチボードである。FiC-SW は他のボードと接続するためのシリアルリンクを有し、回線交換式のネットワークを構築している。本稿では、このようなマルチ FPGA システム上に深層学習の学習機構を実装した。実装では各ボードに重みを利用する処理が 1 つずつ存在するようにし、全結合層部分については入出力を並列に行えるよう複数の演算器を用意する形で並列化を行い、実行時間や電力性能についての評価をとった。その結果、並列化した部分の実行性能は 1166FPS となり、ナイーブな実装の 164 倍の性能を達成した。

キーワード FPGA, マルチ FPGA, 深層学習

Implementation of Training Phase of Convolutional Neural Networks on Multi FPGA

Aoi HIRUMA†, Yugo YAMAUCHI†, Kazusa MUSHA†, Tomohiro KUDOH††, and Hideharu AMANO†

† Keio University
†† The University of Tokyo
E-mail: †{hirumaruma,yyamauchi,mushak,hunga}@am.ics.keio.ac.jp, ††kudoh@nc.u-tokyo.ac.jp

1. はじめに

深層学習アプリケーションの実装では、演算コストと電力コストを抑えるため、省電力性と柔軟性に優れる FPGA(Field-Programmable Gate Array) がプラットフォームとしてよく選ばれている。しかし、単一 FPGA で深層学習アプリケーションの膨大なデータとネットワーク機構を搭載する場合、条件を満たすことができる FPGA はハイクラスで高価になってしまう傾向にある。

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) がスタートさせた「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」プロジェクトでは、ミドルクラスの FPGA を複数接続し、1 つの大きな FPGA のように動作させる FiC(Flow-in-Cloud) システムの開発を行っている。FiC システムでは高速リンクによって FPGA を搭載したボードを多数接続し、STDM スイッチを用いた回線交換方式によってネットワークを構築することで、コスト性能比の高い FPGA を用いてスケーラブルに実装を行うことが可能となる。

本稿では、4 枚の FPGA 搭載ボードを接続して形成した FiC のプロトタイプシステムに Lenet-5 の学習機構を実装する手法を提案する。対象アプリケーションを 4 つに分割することで、Prototype-FiC 上に LeNet-5 の学習機構を実装した。また、各層の特徴を検討し、まずボード 2 枚分、主に全結合層の処理に対して並列化を行うことで実行性能の向上を図り、ナイーブな実装と合わせて評価を取った。

本稿の2.節では FiC システムの紹介を行い、3.節において、関連研究について述べる。4.節では畳み込みニューラルネットワークについて述べ、5.節ではその演算に伴う並列性と実装方法について、そして6.節ではその手法を採用した際の評価を行う。最後に7.節で今後の課題と結論について述べる。

2. 背　　景

2.1 Flow-in-Cloud

FiC(Flow-in-Cloud) システムは、FPGA と DRAM を搭載した複数のボードを高速シリアルリンクにより多数接続するこ

とで成立する。このボードのことを FiC-SW と呼び、ボード上に STDM スイッチとユーザの設計したアプリケーションを搭載する。このボード 1 枚を 1 ノードとして、ノード間の接続はボード上の STDM スイッチを介したサーキットスイッチング方式によって構築される。

図は FiC-SW を複数組み合わせた FiC システムの構成例である。

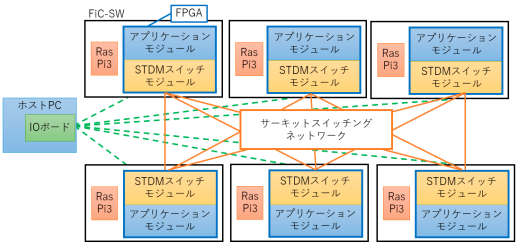

図 1　FiC の構成例

2.2　FiC-SW1

FiC-SW1 は FiC システムのプロトタイプに使用するため製作された試験用ボードである。このボードを高速シリアルリンクによって 4 枚接続し、FiC システム開発用のプラットフォームとした。FiC-SW ボードの FPGA にはユーザによるアプリケーションだけでなくスイッチも搭載する。したがって、大規模アプリケーションの実装においてもアプリケーション部分に十分な資源を確保できるよう、Xillinx 社の Kintex UltraScale XCKU095 を採用している。

図2に FiC-SW1 ボードのハードウェア構成を示す。アプリケーションとスイッチの役割を担う FPGA デバイスの他、メモリとして DDR-4 SDRAM16GB を 2 組持ち、ボード上にデータを格納することが可能となっている。ボード同士の接続を行うため、各ボードが最大 9.9Gbps の双方向高速シリアルリンクを 32 組備えている。また、制御および FPGA のコンフィギュレーションを行うため、RaspberryPi 3 を各ボードに搭載している。Rasberry Pi3 は Ethernet で FPGA と接続しており、ホスト PC から受け取った再構成用データを FPGA に仲介する役割を果たしている。

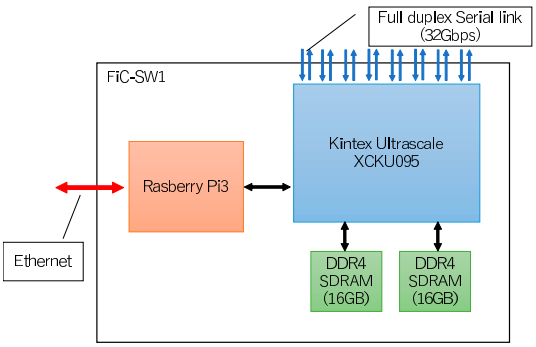

図 2　FiC-SW1 ボードの構成

2.3　STDM スイッチ

FiC システムでは、回線交換式のネットワークを構築するために STDM(Static Time Division Multiplexing) スイッチを採用している。

STDM スイッチを利用する場合、分割したモジュール間の通信における入出力スロット数を最小限に抑えることがシステムの高速化につながる。このため、回線交換方式の転送時間が予測可能であるという特性を利用して、合成時に算出することができる各モジュールの実行サイクル時間とデータ転送量から最適化できるようにした。これにより、転送オーバーヘッドを隠蔽し全体の高速化を行うことができる。

図3は 3 ポート 4 スロットの STDM スイッチの動作例である。各ボードのアプリケーションモジュールから入力された 85bit のデータは入力ポートを通過してスイッチ上で対応するスロットの入力レジスタに格納される。その後、設定されたルーティングテーブルに従って送信先ポートに対応しているレジスタにコピーされる。このように、あらかじめデータの送信先ポートを決定しておくことにより回線交換方式を実現している。

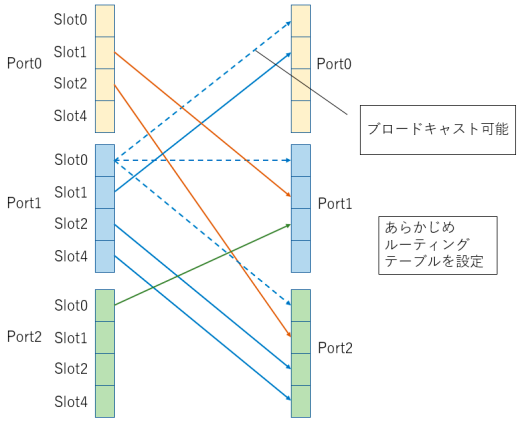

図 3　3 ポート 4 スロット STDM スイッチの動作例

3.　関 連 研 究

3.1　マルチ FPGA システム

著名なマルチ FPGA システムとしては、まず Microsoft 社の Catapult [1] が挙げられる。Catapult はノード間の通信に独自のシリアル I/O を備えているが、この他にもノード間に Ethernet や PCIe を用いるものもあり、例としては Berkeley Univ の BEE3 [2] や東北大学の密結合クラスタ [3] 等がある。

これらのマルチ FPGA システムはすべてパケット交換方式でボード間のネットワークを構築しているため、STDM スイッチを利用して回線交換方式で通信を行う FiC システムとは異なっている。

データセンター等で用いられるマルチ FPGA システム上に深層学習をマッピングする手法の研究 [4] では、深層学習の一つであるアンサンブル学習をテストケースとして、FPGA のノード数を変更した際にノード間通信がボトルネックとなる枚数や、マルチシステム化が有効であるニューロン数について調

査を行っている。

3.2 FPGA での深層学習実装

CNN は大規模なデータを扱うことが多く、また畳み込み演算という特徴的な演算を主体においているため、柔軟性と省電力性を備えた FPGA がしばしばアクセラレータのプラットフォームとして利用される。

そして、特に学習処理では推論処理よりも多くの演算を行い、より多くのデータを扱うため、そのハードウェアリソースの削減は課題となる。学習で扱う小数点の精度に言及している研究 [5] では、指数幅と仮数幅の両方を設定できる浮動小数点を導入し、順伝播処理と逆伝播処理とをそれぞれで適切な精度をカスタマイズして演算を進めるフレームワークを提案している。

推論機構と学習機構を同時に 1 枚のボードに搭載した研究としては [6] があげられる。この研究ではノードの情報をあえて削減することで精度を保ったまま計算効率を向上させるスパースニューラルネットワーク扱っており、あらかじめ削減する部分を決定しておくことで、ハードウェア資源の削減および高速化を行って Artix-71 枚に実装を行っている。

畳み込み処理の重みを離散コサイン変換により圧縮して学習を行う手法を提案している研究 [7] では、ASIC への実装のプロトタイプとして FPGA に LeNet [8] の学習機構を実装している。

4. 畳み込みニューラルネットワーク

4.1 概　　要

CNN(Convolutional Neural Network) は、主に画像認識や音声認識において使用され、特に画像認識における深層学習の分野で普及している手法の一つである。CNN は、特徴マップと合致するパターンを探す畳み込み層と情報を削減するプーリング層、識別を行っている全結合層と全結合層の次に配置する活性化関数、そして最後に結果を確率に変換するソフトマックス関数を用いて全体のネットワークが構成されている。本稿では、畳み込み層 2 つと全結合層 2 つからなる LeNet-5 [9] を実装した。

入力画像を識別する過程を推論と呼び、推論で用いる重みを適切な値に決定する過程を学習と呼ぶ。学習の処理では、学習途中の重みで推論処理を行った結果と教師データとの差分から、誤差逆伝播法によってそれぞれの層における重み誤差を求め、その値を学習途中の重みに加えることで結果を教師データに近づけていく。

誤差逆伝播法では、連鎖律を利用して、それぞれの層における入力に関する局所的な微分を 1 つ前の層に伝播させていくことで、各層における重みに関する損失関数の微分を求める。ここで伝播させていく微分値や重みの更新に用いる微分値を学習処理においては誤差と呼び、またその処理方向から、推論の処理を順伝播と学習の処理を逆伝播と呼ぶ。

以下で、それぞれの層における誤差の求め方について説明する。

4.2 全結合層と活性化関数における誤差

まず、ReLU 層には重みがないため、ここでは上層に渡す誤差のみを求める。ReLU 関数の入力に関する誤差は、順伝播時の入力を x とする時、以下の式によって求められる。

$$\frac{\partial E}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{1}$$

そして、この ReLU 関数の誤差を受け取って、全結合層において重み誤差とバイアス誤差を求め、また上層に伝播させる全結合層の入力の誤差も求める。全結合層の順伝播における入力を X、出力を Y、更新前の順伝播で用いた重みを W とすると、誤差は以下の式によって求められる。

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y} \bullet W^{\mathrm{T}} \tag{2}$$

そして、重みとバイアス B の誤差は、それぞれ以下の式によって求められる。

$$\frac{\partial E}{\partial W} = X^{\mathrm{T}} \bullet \frac{\partial E}{\partial Y} \tag{3}$$

$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y} \tag{4}$$

4.3 畳み込み層及びプーリング層における誤差

畳み込み層では重み誤差、バイアス誤差を求め、その値で重みとバイアスの更新を行う。また入力値に関する畳み込み層の誤差を求め、伝播してきた下層の誤差と積をとって上層に伝播させる。

カーネルサイズ-1 のサイズで伝播してきた誤差のパディングを行った後、順伝播で用いた更新前のカーネルを、次元とチャンネル数はそのままにマップのみ上下左右反転させたもので畳み込むことで、畳み込み層において逆伝播させる誤差を求めることができる。

畳み込み層の重み誤差は、下層から逆伝播してきた誤差を用いて順伝播時の入力を畳み込む形で重みの誤差を形成していく。バイアス誤差に関しては、入力の和をチャンネル毎にとることで求めることができる。畳み込み層での演算をまとめると図4のようになる。

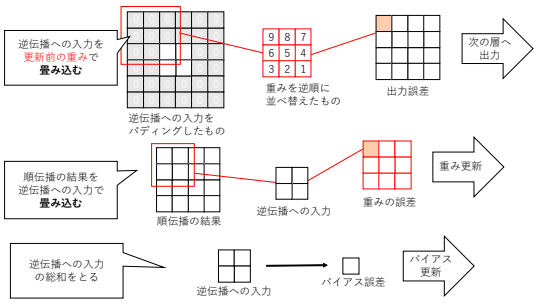

図 4　畳み込み層の逆伝播演算

そして、プーリング層では重みやバイアスが存在しないため、出力誤差のみを求める。プーリング層における出力誤差は、順伝播の際にどの位置のピクセルの情報が選択されたかを探索、同じ位置のピクセルに伝播してきた誤差を配置し、そのほかのピクセルは 0 とすることで求められる。

5. アプリケーションの実装と並列化

5.1 設計概要

Prototype-FiC システムが 4 枚の FiC-SW1 ボードから形成されるため、本稿では、LeNet-5 の学習部分を 2 層ごと、4 つに分け搭載する。逆伝播処理では重み及びバイアスの誤差も求め、またその値によって更新を行うが、更新時の読み出しおよび書き込みの処理は学習機構においてボトルネックとなりやすい。このため、重みは FPGA 上にローカルストレージとして保存する設計とした。従って、各ボード重みを用いる層とその層に付随する処理一つがセットになるように LeNet-5 学習機構の分割を行った。図5に、実装の概要を示す。各ボードは逆伝播の処理の出力結果のみを STDM スイッチを通して次のボードに送信し、重みについては他ボードで使用されることがないため、ボード外に送信せず各自のバッファの更新を行うのみにとどめる。

ボードに搭載するモジュールは主に、積和演算器、入出力バッファ、重みバイアスバッファ、順伝播の入出力バッファ、そして ReLu やソフトマックス、またプーリング等付随する処理を行う演算器から構成される。モジュールは全て高位合成言語 (HLS) によって記述し、32bit 浮動小数点数を用いて演算を進める。各重みの初期値及び、学習時に用いる順伝播機構への入力は FiC-SW1 ボード上の Rasberry Pi3 から送信、このモジュール内に作成したバッファに保存することで逆伝播処理時に使用可能となるようにした。

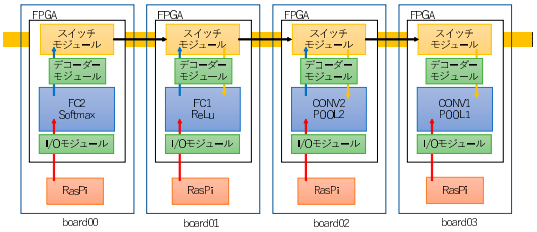

図 5 実装概要

これらの設計を実装する第一段階として、まず LeNet-5 の学習機構を高位合成可能にするため、オリジナルのアルゴリズムをナイーブな形で c 言語にて記述を行った。その後、並列度を全結合層に導入し、並列化を行った。

5.2 並列化の流れ

並列化を行うに当たり、入力並列度 M と出力並列度 N を上記プログラムに導入した。すべてのノードについて並列化を行うとリソースが不足してしまうため、このパラメータに従い、各モジュールについて入力を M 個並列に読み込み、出力を N 個並列に出力することで並列化を実現する。パラメータを利用することで、対象アプリケーションや FPGA の総資源量に合わせてモジュールの規模を変更可能な設計となるため、この実装を行ったあとパラメータの値を変更しながらモジュールのインターバルが小さくなる組み合わせについて調査した。

実際に M 個並列入力、N 個並列出力を実現するために行った事項は以下の通りである。

まず、$M \times N$ の演算を同時に行うためには、その数だけ演算器が必要である。従って、演算器に対してループ展開を導入し、また演算器同士の競合を防ぐためのバッファを必要数用意した。

そして次に、演算器をパイプライン化して動作させた。この処理がなければ、$\frac{I}{N} \times \frac{O}{M}$ 個の演算器が順に起動するのみとなり、数を用意した意味がなくなってしまう。従って、パイプライン化を行うことで、すべての演算器が並列に動作するようになるため、全体のスループットを上昇させることができる。

最後に、バッファを分解して内部帯域幅を確保した。通常一つの配列に対するアクセスは 1 サイクルにつき 1 つに限定されるが、バッファを配列の値一つずつ分割しておくことで、バッファのすべてのインデックスに同時にアクセスすることが可能となる。

以上の処理を行うことにより、N 個同時読み込み及び M 個同時書き込み可能な $M \times N$ の演算器が、それぞれで並列に動作するような設計とした。

また、全結合層の出力を M 個に分割しているため、各全結合層に付随する Softmax 関数や ReLU 関数への入力も M 個に分割される形となり、これらの関数も M 個の演算器によって並列実行が可能となる実装とした。

5.3 並列度の決定

実際に利用する M と N の値を決定するため、パラメータを変化させながらインターバルが最短となる組み合わせを探索した。パラメータ M、N はそれぞれ入力ノード数と出力ノード数の約数である必要がある。今回の設計では、M × N 個の演算器を用意するため、その積が大きければ大きいほど必要なリソースの数も多くなり、また実行性能も向上するはずである。

まず DSP 使用量が 100% を超えない積の値を探し、その積を導くすべての組み合わせについて DSP 使用量をそれぞれ求め、またその取りうる積を小さくしていくことで最適な並列度を探索した。並列度決定の指針に DSP 使用量を用いたのは、本実装の並列化が DSP 使用量によって制限されるからである。

まず、ボード 00 の全結合層の入力ノード数は 10、出力ノード数は 500 であるので、ボード 00 における入力並列度 M と出力並列度 N の値はそれぞれこの二つの約数でなければならない。M と N の積を小さくしていって検証したところ、$M \times N = 25$ の場合は DSP 使用量が 100% を超え、$M \times N = 20$ の時は容量を超える場合と超えない場合があると判明したため、その二つの値ともう一段回小さい値である $M \times N = 10$ の場合についてそれぞれインターバルを調べた。その結果、入力並列度 2、出力並列度 10 の場合において DSP 使用量が 90%、インターバルが 1590 となり、DSP 容量を満たす組み合わせの中でインターバルが最小となることが分かったため、この値によって並列化を行った。

そしてボード 01 に関しては、全結合層の入力ノード数 500、出力ノード数 800 であるので、M と N の値はそれぞれこの二つの約数でなければならない。ボード 00 の場合と同様に M と N の積を小さくしていって検証したところ、表2のような結果となった。従って入力並列度 2、出力並列度 10 の場合におい

表 1　ボード 00 の CNN モジュールにおけるリソース使用量

$M \times N$	入力並列度	出力並列度	DSP	使用量 (%)	インターバル
25	5	5	867	112	1884
20	1	20	678	88	1615
20	2	10	696	90	1590
20	10	2	820	106	1590
10	10	1	490	63	3090
10	1	10	348	45	2616

て、DSP 使用量が 83%、インターバルが 81093 となり、DSP 容量を満たす組み合わせの中でインターバルが最小となることが分かったため、この値によって並列化を行った。

表 2　ボード 01 の CNN モジュールにおけるリソース使用量

$M \times N$	入力並列度	出力並列度	DSP	使用量 (%)	インターバル
32	4	8	860	111	51254
20	1	20	514	66	81957
20	2	10	528	68	81461
20	4	5	556	72	81919
20	5	4	574	74	81173
20	10	2	644	83	81093
20	20	1	784	102	81783
16	1	16	416	54	101957
16	2	8	432	56	101461

6. 評　　価

6.1 評価環境

設計したアプリケーションモジュールについて、リソース消費量及び実行性能、電力性能に関して評価する。各設計環境は表3の通りである。実験環境である Prototype-FiC は FiC-SW ボード 4 枚を高速リンクで相互に接続したものである。

表 3　設 計 環 境

デザインツール	Vivado HLS Version 2017.2 (Xilinx)
実装	Vivado Version 2017.2 (Xilinx)
FPGA	Kintex UltraScale XCKU095-FFVB2104 (Xilinx)
周波数	100MHz

6.2 実装でのリソース消費

以上により決定した並列度で実装を行った結果、スイッチ等 CNN モジュール以外の使用モジュールもすべて FiC-SW 上の FPGA に搭載した状態でのリソース消費は以下の表6の通りとなった。並列化により演算器を増設しているため、DSP 数はボード 00 で 90.6%、ボード 01 で 83.9% となり、他の資源よりも多く消費されていることが分かる。

また、畳み込み層を含む後半ボード 2 枚についてのリソース消費量について全体的にまだ余裕があり、並列化を行うことでこの余剰分を更に活用できると考えられる。

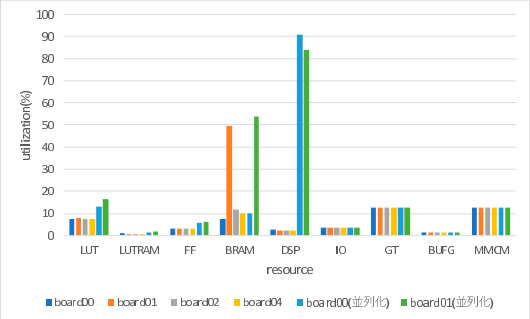

図 6　各ボードでのリソース消費割合

6.3 実行性能と電力性能についての評価

4 枚の FiC-SW ボードでナイーブな実装の LeNet の学習機構を動作させた場合と CPU でオリジナルのアルゴリズムを記述したプログラムを実行した場合について演算性能を測定し、比較した。並列化を行った 2 枚のボードの演算性能についても算出し、CPU 実行の場合とボード 2 枚分をナイーブな実装で動作させた場合との結果と比較を行った。比較対象である CPU には Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz を用い、ナイーブな HLS プログラムを制作した際に使用したプログラムとほぼ同様の C プログラムを動作させた際の実行時間を 1000 回測定し平均を算出したところ、結果は表4の通りとなった。

並列化を行ったボード前半 2 枚分については、ナイーブな実装を動作させた際の実行性能は 7.13FPS(Flame Per Second: 1 秒間に処理できる画像枚数) となり CPU 実行の 4% 程度であったが、並列化を行った場合は 1166FPS となり、CPU 実行の 6.98 倍・ナイーブな実装の 164 倍を達成した。今回ボード前半の全結合層で行列演算に対して行った入出力の並列化は畳み込み演算に関しても適用できる [10] ため、同様に並列化を行えば畳み込み層を含めた全体に関しても CPU 実行を上回ることが可能であると考えられる。

電力性能に関しては、演算実行中の FiC-SW1 ボード 1 枚あたりの消費電力を測定して算出を行った。評価ボードの演算中平均消費電力は 17.89 W であり、2 枚分の並列化実装についてはこれを 2 倍したものを消費電力として電力性能を求めた。結果は表5の通りである。並列化した実装は FC 層 1 つのみの場合において 0.0648 GLOPS/W、2 枚合わせた場合で 0.0924 GLOPS/W という結果となった。並列化した実装とナイーブな実装を比べたとき、FC 層 1 枚分では 59.5 倍、2 枚合わせた実装では 164 倍を達成した。

各ボードはパイプライン上に動作するため、今回の 2 枚動作の場合においては実行性能が低いボード 01 に、4 枚動作の場合はナイーブな実装かつ入出力ノード数が大きいボード 02 に合わせて実行性能が落ちているということができる。このような場合、より性能を向上させるためにはボードそれぞれの実行時間が均一になるように設計する必要がある。従って、ボード 00 の並列化を最大限行いリソースを消費するよりも、ボード 01 で行う処理を分割してボード 00 に一部移行させることでボード 00 のリソースを消費する方がより高い実行性能を達成する

ことができると考えられる。

表6には、関連研究と本稿の結果をまとめた。本稿の結果として掲載している 3.31GFLOPS は並列化を行ったボード 2 枚分の性能である。本稿での実行性能はこれらの先行研究には及ばないため、今後畳み込み層の並列化及びボード全体での処理最適化を行うことで、これらへ性能を近づけていくことが必要となる。

表 4　実行性能 (FPS)

	CPU	ナイーブ実装	並列化実装 (CPU 性能比)
FC2(Softmax)		556	33070
FC2+FC1(Softmax+ReLu)	167	7.13	1166 (×6.98)
LeNet-5 Full Size	40	1.78	

表 5　電力性能 (GFLOPS/W)

	ナイーブ実装	並列化実装
FC2(Softmax)	0.00109	0.0648
FC2+FC1(Softmax+ReLu)	0.000505	0.0924

表 6　関連研究との比較

	GOPS(GFLOPS)
Sparce NN (Artix-7) [6]	96.9 GOPS
LeNet(Virtex-5) [7]	33.7 GOPS
本稿	3.31 GFLOPS

7. 結論及び今後の課題

本稿では、マルチ FPGA システムである FiC(Flow-in-Cloud) システムのプロトタイプである Prototype-FiC において、LeNet-5 の学習機構を実装し、全結合層と付随する処理について並列化処理を導入した。重みを用いる処理とそれに付随する処理とを 1 セットとして全体を 4 分割にし、重みの更新は他ボードと通信することなくボード内で完結する設計とした。並列化に関しては、入出力をそれぞれ分割するパラメータを利用して、その積の数だけ演算器を用意する形式で行い、FPGA の資源上限を上回らないもののうち最短のインターバルを達成するパラメータを設定した。

結果として、並列化を行ったボード 2 枚分については 1166FPS を達成し、これはナイーブな実装を動作させた際のボード 2 枚分の実行性能の 164 倍、CPU で同様のプログラムを実行した際の性能の 6.98 倍となった。並列化を行わなかった後半 2 枚分のボードも含め 4 枚のボードを動作させた結果としては、実行性能が 1.78FPS となった。電力性能に関しては、並列化を行った場合がそうでない場合の 59.5 倍を達成し、0.0648GFLOPS という結果を得た。

今後の課題としては、残り 2 枚の畳み込み層を含むボードの並列化、ボードに搭載するモジュールの量を調整することによるシステム全体の最適化、STDM スイッチのスロットを利用した通信性能の改善等があげられる。また、FiC システムで用いる評価ボードは資源容量が拡大する予定のため、これらの項目を検討したうえで実装を行えば、より実行性能を向上させることができると考えられる。

謝　　辞

この成果は、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。

文　　献

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

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

[3] Yoshiaki Kono, Kentaro Sano, and Satoru Yamamoto. Scalability analysis of tightly-coupled FPGA-cluster for lattice Boltzmann computation. *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 120–127, aug 2012.

[4] M. Owaida and G. Alonso. Application partitioning on fpga clusters: Inference over decision tree ensembles. *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 295–2955, Aug 2018.

[5] Roberto DiCecco. Caffeinated FPGAs: FPGA Framework for Training and Inference of Convolutional Neural Networks With Reduced Precision Floating-Point Arithmetic. *ProQuest Dissertations and Theses*, p. 87, 2018.

[6] Sourya Dey, Diandian Chen, Zongyang Li, Souvik Kundu, Kuan-Wen Huang, Keith M. Chugg, and Peter A. Beerel. A highly parallel FPGA implementation of sparse neural network training. *CoRR*, Vol. abs/1806.01087, , 2018.

[7] Jong Hwan Ko, Burhan Mudassar, Taesik Na, and Saibal Mukhopadhyay. Design of an Energy-Efficient Accelerator for Training of Convolutional Neural Networks using Frequency-Domain Computation. *Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC '17*, pp. 1–6, 2017.

[8] Léon Bottou Yann LeCun, Patrick Haffner and Yoshua Bengio. Gradient-based learning applied to document. *PROC. OF THE IEEE*, 1998.

[9] Léon Bottou Yann LeCun, Patrick Haffner and Yoshua Bengio. Object recognition with gradient-based learning. *Advances in Neural Information Processing Systems 25*, 1999.

[10] Huimin Li, Xitian Fan, Li Jiao, Wei Cao, Xuegong Zhou, and Lingli Wang. A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks. *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*.

[11] 工藤知宏, 高野了成, 天野英晴, 鯉渕道紘, 松谷宏紀, 塙敏博, 池上努, 須崎有康, 田中哲, 赤沼領大, 並木周, 田浦健次朗. データの流れに着目した異種エンジン統合クラウドシステム flow in cloud. 信学技報　vol. 117　no. 153　CPSY2017-16, pp. 1–5, 2017.

[12] Gan Feng, Zuyi Hu, Song Chen, and Feng Wu. Energy-efficient and high-throughput fpga-based accelerator for convolutional neural networks. *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 624–626, Oct 2016.