Vol.2018-HPC-163 No.16
2018/3/1

CNN の学習におけるチャネル方向並列化の提案

赤沼 領大[a,b] 高野 了成[b] 工藤 知宏[c]

概要： CNN は現在画像認識をはじめとしたさまざまな処理に用いられているが、その学習には膨大な時間が必要である。そこで処理をデータ並列で並列化による CNN の学習高速化が一般に行われている。しかしこの方式では利用できる並列度がグローバルミニバッチサイズに制限されてしまう。この制限を拡張するために本研究ではデータ並列と CNN のチャネル方向の並列化を導入したモデル並列を併用したハイブリッド並列を提案する。演算時間の予想モデルを構築しデータ並列とハイブリッド並列の比較を行った。

キーワード：CNN 学習，並列計算機，データ並列，モデル並列

Proposal of a channel-wise parallelization scheme for training of CNN

Erio AKANUMA[a,b] Ryousei TAKANO[b] Tomohiro KUDOH[c]

1. はじめに

現在、CNN（畳み込みニューラルネットワーク）は画像認識をはじめとする広い分野で用いられている。しかし、パラメータや演算量が膨大なために CNN の学習には長時間を要する。CNN の学習を高速化する手法の 1 つとして並列化があり、CNN の学習をデータ並列で演算を高速化させる手法が現在一般的に用いられている。しかし、この方法では一度に行う演算器の利用率効率を高めたり、演算の並列度を上げたりするためにはミニバッチのサイズを大きくする必要がある。しかし、ミニバッチのサイズを上げすぎると学習の収束性が悪くなることが知られている。

データ並列以外の演算の並列化としてモデル並列がある。モデル並列は 1 つの CNN を複数のノードに分割して並列化する手法である。本報告では CNN の各層のチャネル方向への並列化を導入し、データ並列にとモデル並列を組み合わせたハイブリッド並列を行う、既存手法のよりも並列度が高い CNN 学習の実行手法を提案する。またこれらの実行時間の予想モデルを立てて、データ並列とハイブリッド並列を比較した。

最後に GPU と FPGA で CNN の畳み込み層のチャネル方向並列化を行ったときの演算性能を評価した。

2. 研究背景

2.1 CNN

ニューラルネットは高度な情報処理を目的として脳の特性を計算機上でモデル化した物であり、その中で畳み込み層を持つものを CNN と呼ぶ。図 1 の様に CNN は複数の層を持ち、各層は固有のパラメータを持つ。各層は入力されたデータ（特徴マップと呼ばれる）にその層のパラメータを使って処理を行い、結果を次の層へ出力する。画像のクラス分類を行う CNN であれば初めの入力は画像になり、各層で適宜処理が行われて、最終層の出力にはその画像がどのクラスに属すのかを示す確率ベクトルが出力される。

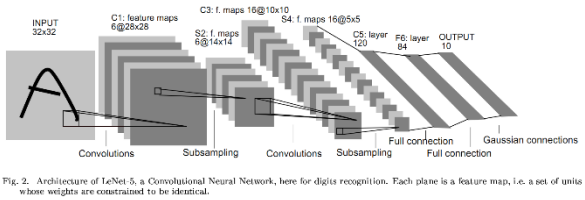

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

図 1 CNN の流れ [1]より

2.2 CNN の学習

CNN の各層はパラメータを持ち、そのパラメータを用いて推論処理を行う。CNN が目的とする推論を行えるようにするためにこれらのパラメータの値を適切に設定する必要があり、大量のデータセットを用いてパラメータの値を逐

a 東京大学大学院 工学系研究科
Graduate School of Engineering, The University of Tokyo
b 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
c 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo

次更新する。この過程を CNN の学習という。学習は図 2 の様に 3 つのステップからなる。

①推論

推論はデータの画像を入力に入れ、入力側から出力側へ向けて、各層が特徴マップを伝播させる過程である。最終的な出力として確率ベクトルが出力される。

②バックプロパゲーション

十分に学習が行われていない場合は、推論で出力された学率ベクトルが正しいものとは限らない、そこでデータセットのデータの正解ラベルと推論の出力の差を出力側から入力側に伝播させる。この処理で伝播されるものを「勾配」と呼ぶ。この処理をバックプロパゲーションと呼ぶ

③パラメータ更新

推論で計算した特徴マップとバックプロパゲーションで計算した勾配を元に各層のパラメータの更新値を得る。

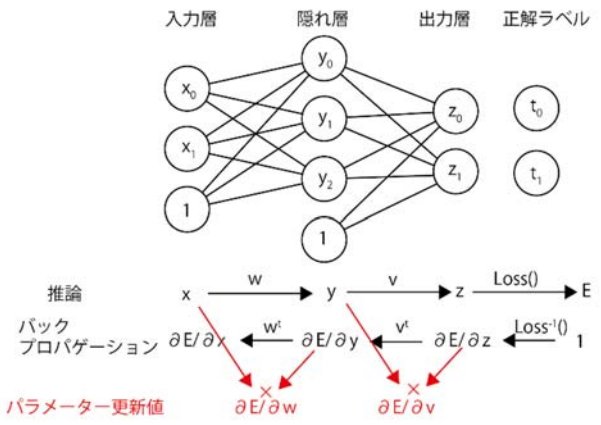

図 2　NN の学習の流れ

学習のために用意したデータセットのすべてのデータを用いて CNN の学習を行うことが CNN の学習の 1 つの単位であり epoch と呼ぶ。通常、学習が完了するまでに数十 epoch ほどかかる。

2.3 ミニバッチ学習

前述の様にデータセットの 1 つのデータごとに推論、バックプロパゲーション、パラメータの更新を行う方式をオンライン学習と呼ぶ。それに対して、複数のデータの推論・バックプロパゲーションをまとめて行い、その後にパラメータ更新を 1 度だけ行う方式をミニバッチ学習と呼ぶ。またこの時にまとめて処理を行うデータ数をミニバッチサイズと呼ぶ。ミニバッチサイズを大きくとると推論・バックプロパゲーションの処理を大きいデータサイズで行うことが出来るので処理の効率が良くなる。また、各データへの推論バックプロパゲーション処理は独立に行え、後述する CNN 学習のデータ並列ではミニバッチ学習の 1 つのミニバッチの処理を並列に行っている。一方、ミニバッチサイズを大きくとりすぎると学習の収束に影響が出て、最終的な CNN の画像認識精度が悪くなることが知られている [2]。

2.4 CNN の並列化

CNN の学習は膨大な量のデータセットへの演算を何十回も繰り返すために時間がかかる。この演算を高速化するために並列計算機を用いたデータ並列での並列化が行われている。データ並列では初めに CNN を並列演算器の各ノードにコピーする。各ノードでそれぞれミニバッチ内のいくつかのデータの学習を独立に行う（図 3 の Forward、Backward に相当）。各ノードで計算したパラメータ更新量をノード間通信で交換し（この通信は AllReduce で表せる）、その平均を取り各ノードに分配する（図 3 の All-Reduce に相当）。各ノードは得られた平均値でパラメータを更新する（図 3 の Optimize に相当）。

この時にシステム全体でのミニバッチサイズ（これをグローバルミニバッチサイズと呼ぶ）は、各ノードでのミニバッチサイズ（これをローカルミニバッチサイズと呼ぶ）を N、ノード数を P と置くと PN になる。グローバルミニバッチサイズがアルゴリズム上のミニバッチサイズに当たり、前述のとおりこれがあまりにも大きくなると学習の収束率に影響を与えるためにこの値には上限がある。そのためにデータ並列で利用可能な並列度にも上限が生じる。

Akiba らのグループはアルゴリズムの改良により、32K のグローバルミニバッチサイズを達成し、1024 個の GPU を用いたデータ並列での CNN 学習の並列計算を実現している [3]。

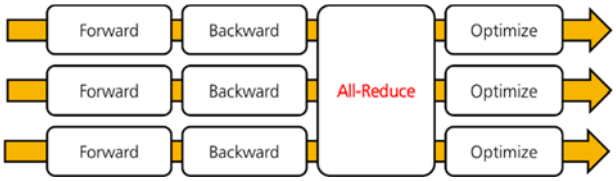

図 3　CNN のデータ並列 [4]より

3. ハイブリッド並列化

前述したように CNN のデータ並列ではミニバッチサイズの制限に起因して並列度に上限が存在する。これが今後さらなる高速化を図るために並列度を上げていく際の障害となる。本節では既存のデータ並列にモデル並列を導入したハイブリッド並列を行い、この制限を拡張する手法について述べる。

3.1 モデル並列化

CNN のモデル並列化とは 1 つの CNN を分割し、複数ノードで協調して演算を行うことである。先行研究として CNN を層単位で分割してモデル並列を行った例がある [5]。CNN の演算は層ごとに分かれていて層の間で特徴マップ

や勾配をやり取りしながら演算が行われている。そこに注目して先行研究では層ごとにノードを割り当てて図 4 の様に CNN の処理をパイプライン化している。

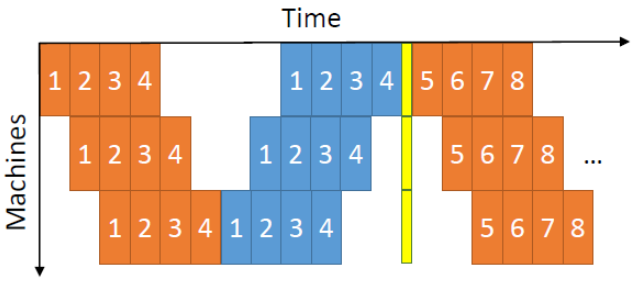

図 4　CNN の層単位でのモデル並列 [5]より

3.2 チャネル方向の並列化

ハイブリッド並列化を行うにあたっては上記の CNN 学習の層ごとの並列化のみでは不十分である。まず全体の演算量のほとんどを占める畳み込み層については、一般的な CNN では畳み込み層によって演算量は異なる。これを無視して層ごとに並列化をするとノードによって演算量にばらつきが生じ、これが性能低下の原因となってしまう。また、一般的な CNN では全体のほとんどのパラメータを 1 つの全結合層が持つ。そのために層ごとの並列化ではパラメータが一部の層に手中し、データ並列において必要になるパラメータ交換の速度向上は望めない。

そこで本報告では上記の 2 つの点に考慮して層ごとのモデル並列化に加え、チャネル方向での層の並列化を導入した。

3.2.1 畳み込み層のチャネル方向分割

1 チャネル入力（in[0:F][0:F]）と 1 つ畳み込みフィルタ（w[0:K][0:K]）から 1 チャネルの出力（out[0:F][0:F]）を得る畳み込み演算自体は以下の Code 1 のように表せる。

```
for(row=0;row<F;row++){
  for(col=0;col<F;col++){
    for(ky=0;ky<K;ky++){
      for(kx=0;kx<K;kx++){
        out[row][col]+=w[ky][kx]*in[row+ky][col+kx]
}}}}}
```

Code 1　1 チャネル入力 1 チャネル出力の畳み込み演算

この演算を$\mathrm{out} = \mathrm{w} \otimes \mathrm{in}$とすると。n チャネルの入力（$\mathrm{In}_0$～$\mathrm{In}_{n-1}$）を受けっとって m チャネルの出力（$\mathrm{Out}_0$～$\mathrm{Out}_{m-1}$）を行う畳み込み層の演算は

$$\mathrm{Out}_j = \sum_{i=0}^{n} W_{j,i} \otimes In_i \quad (0 \le j < m)$$

と表せる。ここではパラメータ j に従って畳み込み層の演算を分割することを出力チャネル方向の分割と呼ぶ。例えば畳み込み層を出力チャネル方向で 2 つのノードに分割した場合、演算量は 2 つのノードに均等に分配出来る。ここで入力データは両方のノードに bcast する必要がある。2 つのノードが出力したデータを gather したものが元の層の出力となる。

3.2.2 全結合層のチャネル方向分割

全結合層の演算は単純な行列-ベクトル積となる。つまり n 要素の入力（in_0～in_{n-1}）を受けとって m 要素の出力（out_0～out_{m-1}）を行う全結合層の演算は

$$out_j = \sum_{i=0}^{n} w_{j,i} in_i \quad (0 \le j < m)$$

と表せる。この時、パラメータ j に従って全結合層の出力を分割することを出力チャネル方向での分割と呼ぶ。全結合層を分割することで、パラメータを複数のノードに分散させることが出来る。これによってノードあたりの交換すべきパラメータ量が減るためにパラメータ交換の処理を高速化することが出来る。

3.3 ハイブリッド並列化

データ並列では 1 つ 1 つのノードが CNN 全体の学習の計算を行うが、これにモデル並列を導入することでモデル並列・データ並列を混合したハイブリッド並列が行える。1 つの CNN をモデル並列により分担して学習をする複数のノードの集合を「セット」と呼ぶ。このセットを複数用意してそれらでデータ並列を行う。

4. 実行時間予想モデル

本節では CNN の学習に掛かる処理時間をモデル化して同じ CNN の学習をデータ並列、ハイブリッド並列で実行した時の各々の処理時間を算出し、ハイブリッド並列の性能についての比較を行う。

4.1 想定環境

VGG16 [6]を元に評価用に一部を単純化した、以下の図 5 に示す CNN を今回の評価の対象とした。サイズが 224x224 の画像が 64 チャネルある入力を受け取り、4 つの畳み込み層と 1 つの全結合層を経て 4096 次元の確率ベクトルを出力する。また各層での演算量、パラメータ量、入出力データ量は以下の表 1 のようになる。全結合層 cv1～cv3 では演算量が等しいが、cv4 の演算量はその半分になる。また全体の 97%ものパラメータが fc1 に集中している。

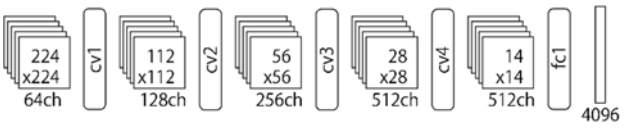

図 5　評価用 CNN

表 1　評価用 CNN の処理量

	演算量（flop）	Param（B）	Input（B）	Output (B)
cv1(L1)	7.40G	0.29M	12.8M	6.42M
cv2(L2)	7.40G	1.18M	6.42M	3.21M
cv3(L3)	7.40G	4.72M	3.21M	1.60M
cv4(L4)	3.70G	9.43M	1.60M	0.40M
fc1(L5)	0.21G	411M	0.40M	4096

またこの CNN の学習の並列化を実行する並列計算機については以下の性能を仮定した。

表 2　想定する並列計算機の性能

ノードあたり演算性能	4T FLOPS
ノードあたり メモリアクセス性能	400G B/s
ネットワーク通信性能	4GB/s
ネットワーク接続	Fat tree

4.2 並列化シナリオ

比較条件としてデータ並列と、3 種類のモデル並列を用いたハイブリッド並列の計 4 通りのシナリオを今回考慮した

① データ並列

② ハイブリッド並列 1（5 ノード）
1 セットは 5 つのノードからなる。CNN を層単位で 5 つに分割した。

③ ハイブリッド並列 2（6 ノード）
1 セットは 6 つのノードからなる。全結合層である fc1 のみチャネル方向に 2 分割し、全結合層は層ごとの分割のみを行った。

④ ハイブリッド並列 3（9 ノード）
1 セットは 9 つのノードからなる。畳み込み層の内 cv1～3 をチャネル方向に 2 分割し、cv4 は層ごとの分割化のみ、全結合層 fc1 もチャネル方向に 2 分割した。

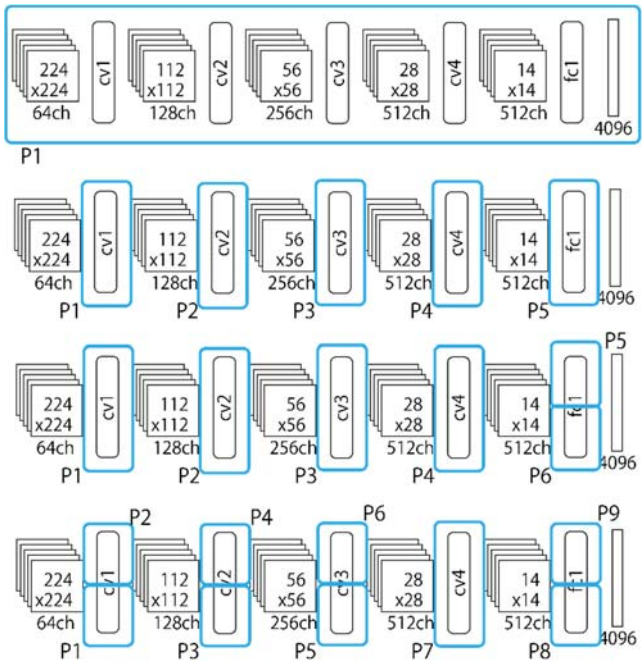

図 6　並列化シナリオ　各シナリオでの「セット」の構成
上からデータ並列、ハイブリッド並列 1、ハイブリッド並列 2、ハイブリッド並列 3

4.3 データ並列の実行時間モデル

データ並列の処理は先述した図 3 にあるように 3 つの部分に分かれるため各々の実行時間をモデル化してその和をデータ並列における実行時間とした。以下ローカルミニバッチサイズを N として 1 回のミニバッチ処理を行う時間をモデル化した。

まず、各層での処理の時間だが、畳み込み層では演算が律速となるので層 Li における処理時間 T_{Li} は層の演算量 $Copm_{Li}$ とノードの演算性能 Pnode を用いて $T_{Li} = \frac{Comp_{Li}}{Pnode}$ と表せる。一方畳み込み層ではメモリ律速となる。よって層 Li における処理時間 Ti は層のパラメータ量を $Param_{Li}$、ノードのメモリアクセス性能を Mnode と置くと $T_{Li} = \frac{Param_{Li}}{Mnode}$ と表せる。

データ並列で各ノードが独立に学習を行う時間(T_1)では N 回の推論と N 回のバックプロパゲーションを行う。推論とバックプロパゲーションの演算量は同じなので以下の式で表される。

$$T_1 = (T_{cv1} + T_{cv2} + T_{cv3} + T_{cv4} + T_{cv5}) \times 2N$$

パラメータ交換を行う部分(T_2)は CNN の全体のパラメータのサイズとネットワーク転送性能(NWnode とおく)から以下の式の様に求められる。

$$T_2 = \frac{Param_{cv1} + Param_{cv2} + Param_{cv3} + Param_{cv4} + Param_{cv5}}{NWnode}$$

最後のパラメータ更新の時間(T_3)はパラメータ更新の演算量は推論やバックプロパゲーションと同じであるので以下

の式で表される。

$$T_3 = (T_{cv1} + T_{cv2} + T_{cv3} + T_{cv4} + T_{cv5})$$

4.4 ハイブリッド並列の実行時間モデル

ハイブリッド並列でも図 7 に示すようにモデル並列と同様に 3 つの部分に分けることが出来る。しかし推論とバックプロパゲーションの処理をノード間でパイプライン的に処理を行うので実行時間の計算は複雑になる。

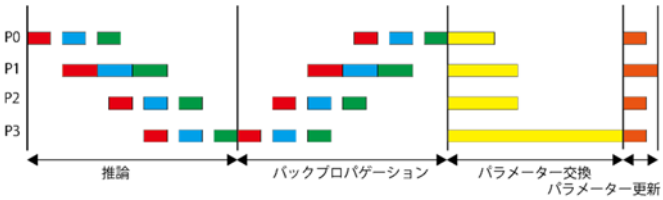

図 7　ハイブリッド並列でのタイミングチャート

推論とバックプロパゲーションは対称な演算なのでここでは推論部分のみに注目する。まずノード 1 で 1 番目のデータの L1 での処理を行う、処理が終わったら出力を次の層を担当するノードに送って、転送終了後ノード 2 でデータ 1 についての L2 でのデータの処理を行い、終わり次第出力を次のノードに送信する。またノード 1 ではデータ 1 の処理が終わり次第、速やかにデータ 2 の処理に移る。この時に重要になる指標として interval と latency がある。interval はデータ 1 の処理を始めてから次のデータであるデータ 2 の処理を開始するための間隔でスループットの逆数である。これは以下の式で表せる。

$$\text{interval} = \max(T_{cv1}, T_{cv2}, T_{cv3}, T_{cv4}, T_{cv1})$$

latency はノード 1 がデータ 1 の処理を始めてから最後のノードがデータ 1 の処理を終えるまでの時間である。層 i から層 j への通信時間を U_{ij} とすると

$$\text{latency} = T_{cv1} + T_{cv2} + T_{cv3} + T_{cv4} + T_{cv1} + U_{12} + U_{23} + U_{34} + U_{45}$$

と表せる。チャネル方向に層を並列化した場合は層の処理時間である T_{Li} との時間が並列化した分だけ小さくなるそのために interval と latency の両方が小さくなる影響を及ぼす。この 2 つの値を用いてハイブリッド並列で各セットが独立にミニバッチ学習を行う時間（T_1）は

$$T_1 = (\text{interval} \times (\text{N} - 1) + \text{latency}) \times 2$$

と表せる。

図 8　推論のパイプライン処理
（上）は層単位のみの分割
（下）第 2 層にチャネル単位の分割を導入

パラメータ交換を行う時間（T_2）はセットの中の各ノードが別々にパラメータ交換を行うので

$$T_2 = \frac{\max(Param_{cv1}, Param_{cv2}, Param_{cv3}, Param_{cv4}, Param_{fc1})}{NWnode}$$

と表される。

最後のパラメータのアップデートを行う時間は

$$T_3 = \max(T_{cv1}, T_{cv2}, T_{cv3}, T_{cv4}, T_{fc1})$$

と表せる。

以下の図 9（上）は 5 ノードでのモデル並列化の模式図となる。例えばこの時にボトルネックとなっている P2 で実行されている層を図 9（下）の様に分割すると interval が短くなって処理が高速化する。

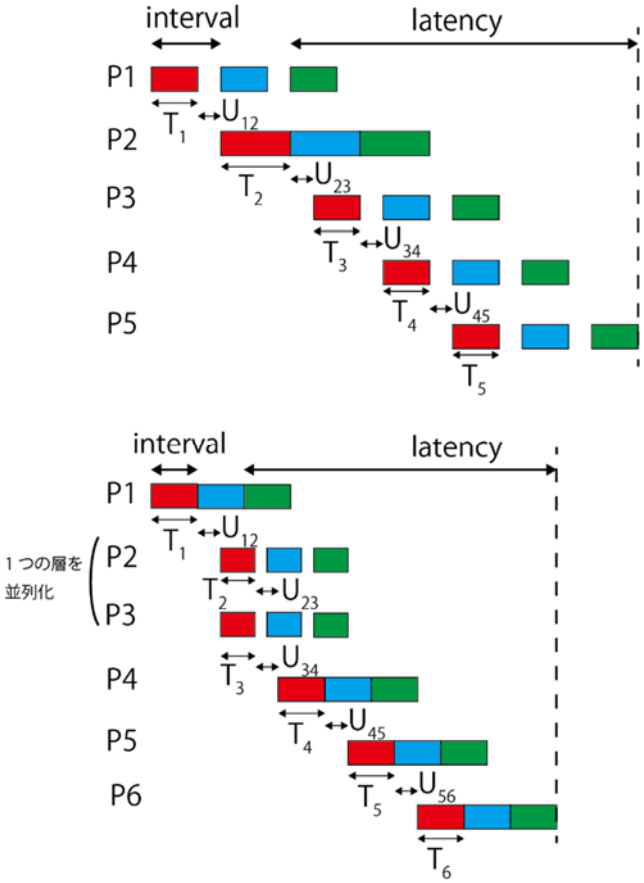

図 9 パイプライン部分のタイミングチャートと配列化による高速化

4.5 比較

1 ノードあたりのグローバルミニバッチサイズを変更して各方式での処理時間の理論値を求めた。この値が小さいほど、ミニバッチサイズが小さく、並列度が高い（使用しているノード数が多い）状況を表している。その結果を図 10 に示す。ハイブリッド 1 方式（hv1:5 nodes）は層単位のモデル並列化のみを使用している方式だが、グラフの全域においてデータ並列（data）よりも時間がかかっている。ハイブリッド並列 3（hv3:0 nodes）はグラフの領域においてはデータ並列よりも処理時間が短く高速に動作している。
特に 1 ノードあたりのグローバルミニバッチサイズが小さい値でデータ並列との処理時間の差が大きい、これは現在以上の大規模な並列度で実行しようとすればするほど、提案方式のハイブリッド並列に優位性があることを示している。

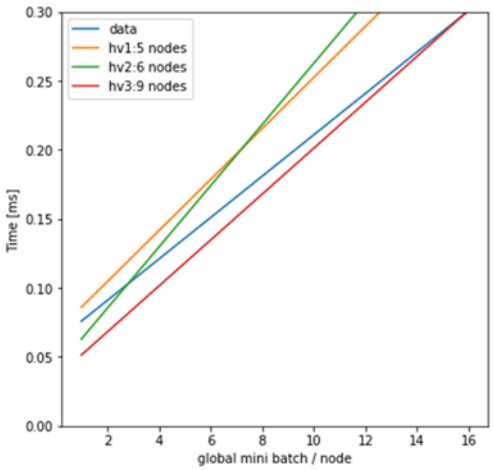

図 10　各方式の処理時間の理論値

5. 評価実験

5.1 GPU 上での演算性能評価

GPU 上で CNN の畳み込み層の入力、出力チャネル数を変えて実行して、各々の条件での GPU の演算性能を測定した。前述の各方式での処理時間のモデルでは畳み込み層の演算のサイズに関わらず同じ演算性能であることが前提となっている。しかし現在 CNN の学習で広く使われている GPU は内部で演算器を多数並列に並べた構造を取っていて、その並列度で高い処理性能を達成している。畳み込み層をチャネル方向で分割すると一度に実行する演算のサイズが小さくなり、GPU での演算性能が低下することが予想される。評価条件を以下に示す。各条件で演算を 10 回行い、後半の 5 回の平均値を測定値とした。

表 3　実験条件

計算機	GPU NVIDIA P100
CNN フレームワーク	Chainer
実行演算	Convolution2D
特徴マップサイズ	62x62
入力チャネルサイズ	1~4096
出力チャネルサイズ	1~4096

各条件の時の入力チャネルと出力チャネル積を総チャネル数とここでは呼ぶ。総チャネル数と演算性能をプロットした物を図 11 に示す。このように GPU での演算では演算サイズによって処理性能が大きく変わる結果が得られた。現在使われている CNN の畳み込み層の総チャネル数は約 10,000~300,000 程度である。この領域では畳み込み演算のサイズによって GPU の演算性能は大きく変化する、つまり現在の GPU ではハイブリッド並列を有効に活用できないということが分かる。

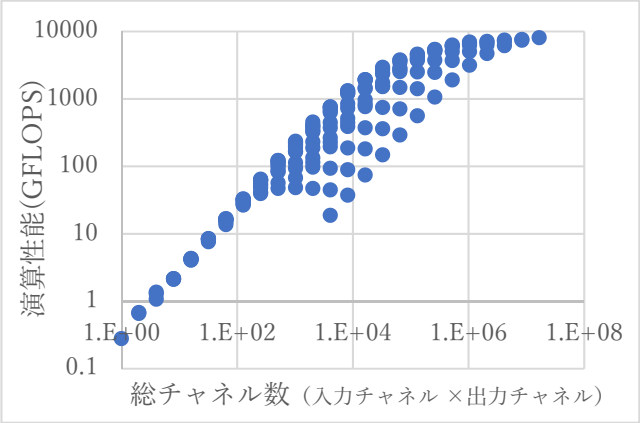

図 11　チャネル総数と GPU の演算性能（両軸共に対数）

5.2 FPGA 上での演算性能評価

FPGA はコンフィギュレーションによって内部に専用回路を生成できる素子である。専用回路により、高スループットで低遅延な処理を行うことが出来る。GPU と違い目的の処理に応じた専用回路を構成するために、チャネル総数の小さい場合でも、演算性能が極端に低下することなく実行可能である。

FPGA 同士を多数接続した並列計算機としては先行研究[7]があるが、そのノード数は数～十数と限られる。

Flow-in-Cloud (FiC)は NEDO「IoT 推進のための横断技術開発プロジェクト」の、「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」で開発している並列計算機であり、多数の GPU、FPGA 等の異種ノードとメモリノードを高速低コストの光ネットワークで接続し、専用ソフトにより統合制御する。この大規模システムでは処理内容に応じて資源を柔軟に接続することが出来る。このシステム上で CNN を実装する開発が現在進められている [8]。

FiC では数十～数百の FPGA 同士を接続して並列計算を行うことが可能である。そのために本提案手法のハイブリッド並列での CNN 実行に適していると言える。

本節では FPGA 上に高位合成で畳み込み演算を実装し、実行間隔を変えた時に必要となる FPGA 上の資源量の変化を観測した。実験条件を以下に示す。

表 4　実験条件

高位合成ツール	Vivado HLS 2016.4
対称 FPGA	Xilinx UVC108(XCVU095)
実行周波数	100MHz
実行演算	畳み込み演算
入力チャネルサイズ	8
出力チャネルサイズ	8
実行間隔	1 clock/data～18 clock/data

実験の結果を以下の図 12 に示す。実行間隔(clock/data)

の値は演算性能(flops)の値の逆数に比例する。実験の結果が示すように実行間隔を上げるとそれに従って各資源の消費量も低くなる。使用する資源量が少なくなると、1 つの FPGA に今回実装した畳み込み演算回路を多数搭載できるようになるため、大きい総チャンネル数の演算が行える。

今回の実装では DSP の使用率が律速となるので、DSP の使用率に注目して使用率と削減率を表 5 に示す。削減率は実行間隔が 1 の値と比べて、どれだけ資源の使用量が減ったかを表す。この表から今回実験した範囲では資源の消費の削減率は実行間隔とほぼ同じ値になることが分かる。つまり FPGA 上に畳み込み演算の演算器を構成した時にでも FPGA 上の回路の演算性能と必要な資源量はほぼ比例する。

今回の実験から実行間隔を変えて資源の消費量を減らすことで大きな総チャンネル数に対応できるように回路を変更しても、演算性能はほぼ変化しないということが分かった。

このことより本報告で提案したハイブリッド並列化は FPGA をプロセッサとして採用することにより効果を発揮出来ると言える。

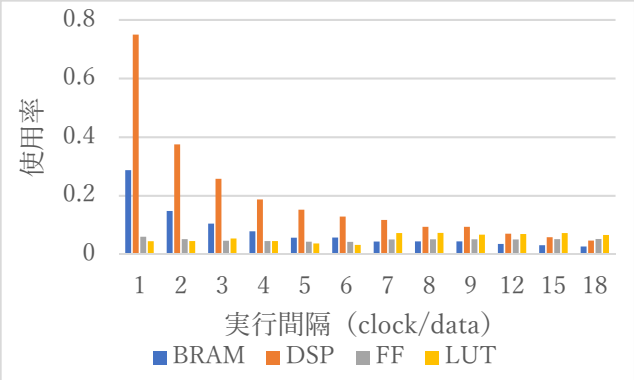

図 12　演算性能と各資源の使用率
（演算性能は右に行くほど低くなる）

表 5　DSP の使用率と削減率

実行間隔（ckock/data）	使用率	削減率
1	0.750	1.000
2	0.375	2.000
3	0.258	2.909
4	0.188	4.000
5	0.152	4.923
6	0.129	5.818
7	0.117	6.400
8	0.094	8.000
9	0.094	8.000
12	0.070	10.667
15	0.059	12.800
18	0.047	16.000

6. 結論

本報告では CNN 学習の並列計算機上での並列所に関して、まずデータ並列について紹介しその限界について述べた。この欠点を拡張する手法としてデータ並列に層とチャネル方向で並列化するモデル並列を導入したハイブリッド並列を提案した。またハイブリッド並列について処理時間をモデル化してデータ並列と比較することで、優位性を検討した。

次に GPU 上と FPGA 上で CNN の畳み込み演算を実装したが、GPU では本手法であるチャネル方向の分割によって極端に性能が低下することが分かった。
本手法の様に演算を細かく分割する手法は FPGA での実装に適している。そのために今後 FiC のような FPGA を用いた並列計算機での実装による効果が期待される。

謝辞　本研究の一部は JSPS 科研費 16H02793 の助成を受けたものです。

参照文献

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[2] Yang You, Igor Gitman, Boris Ginsburg, “Large Batch Training of Convolutional Networks,” 2017. https://arxiv.org/abs/1708.03888.

[3] T. Akiba, S. Suzuki, K. Fukuda, “Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes,” 2017. https://arxiv.org/abs/1711.04325.

[4] Preferred Networks, Inc., "分散深層学習パッケージ ChainerMN 公開," https://research.preferred.jp/.

[5] Alexander L. Gaunt, Matthew A. Johnson, Maik Riechert, Daniel Tarlow, Ryota Tomioka, Dimitrios Vytiniotis, Sam Webster, "AMPNet: AsynchronousModel-Parallel Training for Dynamic Neural Networks," https://arxiv.org/abs/1705.09786.

[6] Simonyan, K., & Zisserman, A., “Very Deep Convolutional Networks for Large-Scale Image Recoginition,” 2015.

[7] Sano, Kentaro, Yoshiaki Hatsuda, and Satoru Yamamoto, "Multi-FPGA accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems,* vol. 25, no. 3, pp. 695-705, 2014.

[8] 武者千嵯、工藤知宏、鯉渕道紘、天野英晴, “マルチ FPGA 上での CNN の実装,” *信学技報, vol.* 116, no. 417, 2017.