Vol.2016-ARC-221 No.38
2016/8/10

ニューラルネットワークアクセラレータにおけるコア間通信量最小化のためのタスク配置手法

進藤 智司[1]　大場 百香[2]　津邑 公暁[1]　三輪 忍[2]

概要：画像認識や音声認識，自然言語処理等において，ニューラルネットワークを用いた機械学習が注目されている．近年の機械学習では認識率を向上させるためにニューラルネットワークの規模を拡大させることが主流であり，これに伴い，シミュレーションに必要な計算時間が増大している．そこで，計算時間の短縮と消費電力の削減を目指し，ニューラルネットワーク専用のハードウェアアクセラレータが盛んに研究されている．このようなハードウェアアクセラレータでは，ソフトウェア実行の際には考慮する必要のなかった点が性能に影響を及ぼす可能性がある．そこで本稿では，高性能なハードウェアアクセラレータの実現のためのひとつの着眼点として，複数コアへのタスク配置について検討する．各コアへのタスク配置がハードウェアアクセラレータの実行性能および電力効率に及ぼす影響を調査し，コア間通信量最小化のための適切な配置について考察する．シミュレーションによる評価の結果，タスク配置が実行サイクル数に影響を与え，層の規模によって異なるタスク配置が有効であることを確認した．

1. はじめに

画像認識や音声認識，自然言語処理などにおいて，**ニューラルネットワーク**を用いた機械学習が注目されている．近年の機械学習は，**Deep Neural Network**（**DNN**）と呼ばれる多層構造を持ったニューラルネットワークを用いることで高い認識率を達成している．例えば，画像認識の国際コンテスト ILSVRC において，2014 年に最も高い認識率を示したニューラルネットワークは 20 層で構成されている [1]．このような多くの層を持つニューラルネットワークを用いた機械学習は**ディープラーニング**と呼ばれ，さらなる認識率の向上を目指した研究が行われている．しかし，ニューラルネットワークの規模の拡大に伴い，そのシミュレーションに必要な計算量が増大している．そこで，シミュレーションに要する計算時間の短縮と電力効率の向上を目指し，ニューラルネットワークのシミュレーションに特化したハードウェアアクセラレータが盛んに研究・開発されている．このようなハードウェアアクセラレータでは，ソフトウェア実行の際には考慮する必要のなかった点が性能に影響を及ぼす可能性がある．本稿では，高性能なハードウェアアクセラレータの実現のためのひとつの着眼点として，複数コアへのタスク配置について検討する．各コアへのタスク配置がハードウェアアクセラレータの実行性能および電力効率に及ぼす影響を調査し，コア間通信量最小化のための適切な配置について考察する．

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 電気通信大学
UEC

2. 研究背景

本章では，ニューラルネットワークの概要，およびニューラルネットワークアクセラレータに関する既存研究について述べる．

2.1 ニューラルネットワーク

ニューラルネットワークとは，生物の脳神経系を模倣した数理モデルであり，**ニューロン**と呼ばれる多数の計算ユニットと**シナプス**と呼ばれるニューロン同士の結合から成る．最も基本的なニューラルネットワークは層状に並んだニューロンが隣接した層のみと結合した構造を持つ．このニューラルネットワークの構造を**図 1** に示す．

図のネットワークは入力層，中間層，出力層の 3 層からなり，入力層へデータが入力されると入力層に位置するニューロンはシナプスを介して中間層のニューロンへ計算結果を出力し，入力層のニューロンから入力を受け取った中間層のニューロンは出力層のニューロンへ計算結果を出力する．この例のように，前層に位置する全てのニューロンから入力が与えられるように隣接層間が結合した層構造を全結合層と呼ぶ．またシナプスはそれぞれ**重み**と呼ばれる値 w_i $(i = 1, ..., n)$ を持っており，1 ニューロンへの入

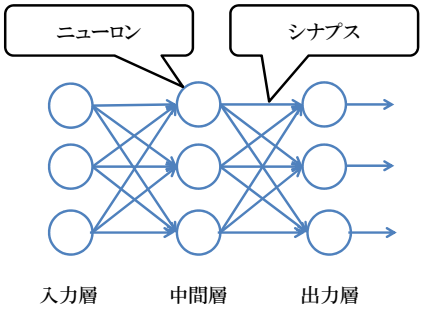


図 1　ニューラルネットワークの構造

力を x_i $(i = 1, ..., n)$ とすると，出力 y は次の式で計算される．

$$u = \sum_{i=1}^{n} w_i x_i + b \tag{1}$$

$$y = f(u) \tag{2}$$

ここで，b はバイアスと呼ばれるニューロンごとに異なる定数である．また f は**活性化関数**と呼ばれる神経細胞の性質をモデル化した関数である．特に近年のディープラーニングでは ReLU[2] と呼ばれる次の活性化関数が主に使用される．

$$f(x) = max(0, x) \tag{3}$$

これらの式からも分かるように，1 ニューロンの出力を求める計算は，入力とそれに対応したシナプス重みの積和計算，および，その積和計算結果に対する活性化関数の適用から構成される．本稿では，1 ニューロンの出力を求めるために必要なこの計算全体を**タスク**と定義する．

ニューラルネットワークでは入力層から出力層に向けて順に，各層のニューロンの出力をこの計算によって求めていく．このようにニューラルネットワークの入力層にデータを入力し，それに対する出力を得ることを**推論**と呼ぶ．さらに，ニューラルネットワークから期待する出力を得るためには，シナプス重みの値を適切に決定する必要がある．適切な重みの値を求めるために，様々な入力を与え重みの値を更新しながら推論を繰り返すことを**学習**と呼ぶ．

2.2　関連研究

近年ニューラルネットワークのシミュレーションに特化した専用アクセラレータが研究されている．そのようなアクセラレータには，大きく分けて 2 つのタイプが存在する．

1 つ目のタイプは，1 ニューロンを 1 つの演算ユニットに割り当て，ニューラルネットワーク全体をハードウェアで再現した構造を持つアクセラレータである．例えば，IBM の Merolla ら [3] は，256 個の演算ユニットに対して 1 つずつニューロンを割り当てることで，256 個のニューロンからなるニューラルネットワークをシミュレーション可能なアクセラレータを開発した．また，Temam ら [4] は機械学習で主に使用されてきたニューラルネットワークおよびニューロンモデルを対象に，シナプス重みを格納したバッファを介して，各ニューロンに対応する演算ユニット同士を接続したアクセラレータを提案している．これらのアクセラレータでは，各演算ユニットは入力を受け取ると同時に計算を開始し，次に接続した演算ユニットに計算結果を伝えるという単純な動作のみを行うため，シミュレーションを実現するために制御用の回路や命令を必要としない．さらに，ニューロンの出力を計算する演算ユニット付近のバッファに，必要なシナプス重みが格納されるため，処理中の演算ユニットまでのデータの移動距離を短くすることが可能である．しかし，これらのアクセラレータではニューラルネットワークのニューロン数や層数，ニューロンモデルは，ハードウェアの構成に制約を受けるため，ニューラルネットワークの構造を変更することが困難である．また，事前に記憶領域に格納したシナプス重みを用いた推論しかできないため，学習によってシナプス重みの値を更新することができない．

2 つ目のタイプは，1 つの演算ユニットが複数のニューロンの出力を計算することで，様々なニューラルネットワークに柔軟に対応可能なアクセラレータである．例えば，Chen ら [5] は複数ニューロンの出力を並行して計算可能なコアを設計し専用命令を実装することで，ニューラルネットワークを高速にシミュレーションするアクセラレータを提案している．さらに同氏らはこのコアを複数接続し，全てのコアから読み書きが可能な記憶領域を用意することで，さらなる高速化を実現している [6]．これらのアクセラレータでは，専用命令によってニューラルネットワークの構造を定義することができ，ニューラルネットワークの規模の変化にも柔軟に対応することができる．さらに，命令セットを拡張することで，推論のみならず，学習も実現できると考えられる．

そこで，我々は様々なニューラルネットワークにおいて推論および学習を高速に行うことを目的として，後者のタイプのアクセラレータを開発している．我々が開発しているアクセラレータは複数のコアから構成されるが，このように複数のコアを持つアクセラレータでは，各コアがタスクを処理する際には入力データを他のコアから受け取る必要があるため，タスク配置の仕方によってコア間のデータ通信量が変化し，アクセラレータの性能に影響を与えると考えられる．そこで本稿では，コア間のデータ通信量の最小化を目指し，各コアへの適切なタスク配置を検討する．

3.　ニューラルネットワークアクセラレータ

本章では，我々が開発しているニューラルネットワークアクセラレータの構成と動作について述べる．

3.1　アクセラレータの構成

我々はニューラルネットワークの複雑な構造や規模の変

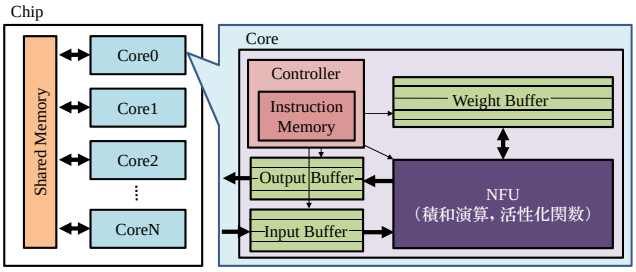


図 2　アクセラレータの構造

化に柔軟に対応することができるアクセラレータを開発している．このアクセラレータの構造を**図 2** に示す．アクセラレータは Shared Memory と複数のコアから構成され，各コアが Shared Memory に対してデータを読み書きすることで，全てのコアでデータを共有することができる．

各コアは，前層から与えられる入力を格納するためのバッファ（Input Buffer），シナプス重みを格納するためのバッファ（Weight Buffer），ニューロンの出力を格納するためのバッファ（Output Buffer）と，これらのバッファに接続した，ニューロンの出力計算用の演算ユニットから構成される．本稿では，この演算ユニットを NFU（Neural Functional Unit）と呼ぶ．NFU は積和演算器と活性化関数用の演算器から構成される．また，これらのバッファおよび NFU は Controller によって制御される．

このアクセラレータでは，コアが Shared Memory からデータを読み出す際，Shared Memory から読み出したデータは一旦 Input Buffer に格納され，NFU は Input Buffer に格納されたデータを読み出す．また NFU での計算結果は，一旦 Output Buffer に格納した後，Shared Memory に書き込む．このように Shared Memory に対するデータの入出力の際，Input Buffer を介することで Shared Memory からのデータの先読みを可能にし，Output Buffer を介することで Shared Memory への書き込み待ちを抑制している．

このような動作によってデータを入出力するため，Input Buffer と Output Buffer が NFU と接続しているネットワークのバンド幅によって，NFU が一度に読み出し可能な入力値の数および出力計算が可能なニューロンの数が制限される．本稿では，一度に読み出し可能な入力値の数を N_i，一度に出力計算が可能なニューロンの数を N_o とする．なお，アクセラレータ中では N_i および N_o の単位でデータを管理し，このデータのかたまりを**チャンク**と呼ぶ．

ここで，アクセラレータ中でニューラルネットワークの入出力およびシナプス重みがどのように格納されるかを**図 3** を用いて説明する．この例では，8 個の入力値から成るベクトルと 12 行 8 列の重み行列，および，それらの乗算によって求められる 12 個のニューロンの出力値から成るベクトルを表しており，$N_i = N_o = 4$ としている．入力値は I_1〜I_8 であり，これらと重み行列の i 行目を乗算し活性化関数を適用することで出力値 O_i を求める．アクセラレータ内では，これらの行列およびベクトルをチャンクに分割して各バッファに格納する．入力と出力はそれぞれ N_i，N_o 個の要素を 1 つのチャンクとし，シナプス重みは N_o 行 N_i 列の行列を 1 つのチャンクとして格納する．

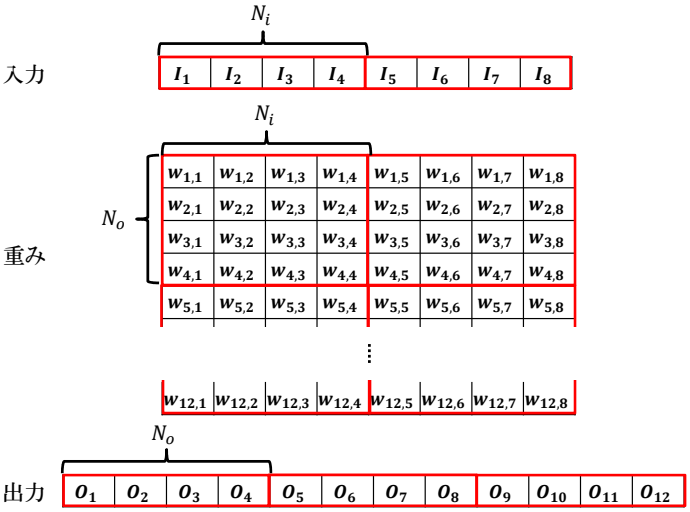
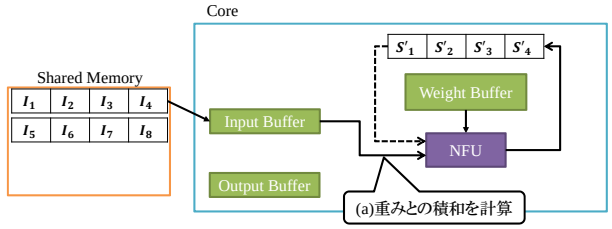


図 3　アクセラレータ内でのデータ格納形式

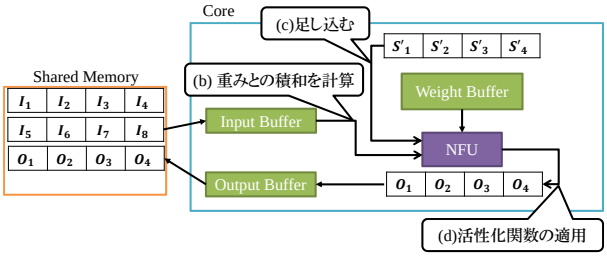
Input Buffer と Output Buffer は複数のエントリを持ち，1 エントリに 1 つのチャンクを格納する．さらに Input Buffer に格納されているデータは再利用することが可能である．つまり，一度 Shared Memory からデータを読み出した後，再び同じデータを読み出す場合，Input Buffer からそのデータを読み出すことで Shared Memory からの読み出しを省略することができる．なお，エントリの数を超えて新たにデータを格納しようとした場合，Input Buffer 上のデータの一部が追い出されるため，その追い出されたデータは以降再利用できなくなる．

3.2　アクセラレータの動作例

前節で述べたように我々が開発中のアクセラレータは，複数の入力から複数の出力を同時に計算可能であるが，入出力の数が N_i および N_o をそれぞれ上回る場合は，繰り返し NFU を用いることで計算する．ここで，N_i を超えた数の入力値を受け取るニューロンの出力を計算する際の，NFU 内での計算，および，コアと Shared Memory の間のデータ移動の様子について**図 4** を用いて説明する．この例では，出力を求めるニューロンの数は 4 であり，$N_o = 4$ と仮定すると 1 つのチャンクとして格納される．また，前層のニューロンから与えられる入力の要素数は 8 であるため，$N_i = 4$ と仮定すると入力は 2 つのチャンクに分けられて Shared Memory に格納される．そのため，2 回に分けて入力を読み出し，積和を計算する．はじめに，1 つ目のチャンクに対する計算を考える (i)．まず NFU は Shared Memory から入力値 I_1〜I_4 を読み出し，対応したシナプス重みとの積和を求める．この際，1 チャンク分に相当する，$N_o = 4$ と同数の積和 S'_1〜S'_4 が計算される (a)．その

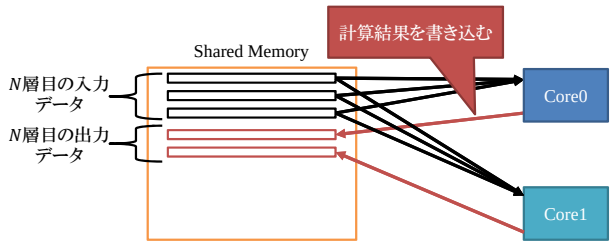


(i) 1 つ目の入力チャンクに対する積和計算

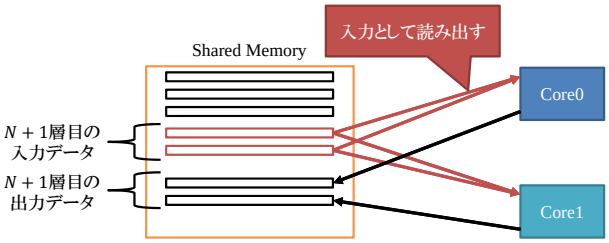


(ii) 2 つ目の入力チャンクに対する積和計算

図 4 ニューロンの出力計算のための計算とデータ通信



(i) N 層のニューロンの出力計算時のデータ移動



(ii) $N+1$ 層のニューロンの出力計算時のデータ移動

図 5 コア間のデータ受け渡し

後，2 つ目のチャンクに対する積和が計算される際 (ii)，同様に読み出した入力値 I_5〜I_8 と対応するシナプス重みの積和を求め (b)，NFU 出力からバイパスされた 1 チャンク目の結果 S'_1〜S'_4 を足し込む (c)．これにより，I_1〜I_8 の入力に対する 4 つの積和を求めることができる．この積和に活性化関数を適用し，ニューロンの出力 O_1〜O_4 を求め (d)，Output Buffer を介してこれらを Shared Memory に格納する．ここまでの動作により，前層の 8 ニューロンから入力を受け取った 4 ニューロンの出力を計算することができる．また N_o を超えた数のニューロンの出力を求める際は，残りのニューロンの出力についても同様の動作によって求める．

以上で述べた動作を繰り返し，ニューラルネットワーク内のニューロンの出力を計算していく．このとき，ニューラルネットワークは層間にデータ依存があるが，層内にはないため，同層に含まれるニューロンは複数コアで並列に出力計算することが可能である．また，ニューロンの出力計算には，前層のニューロンの出力が入力として必要となるため，全てのコアで一層ずつ順にニューロンの出力を計算していくことになるが，この際，ニューロンの出力を全コアで共有する必要がある．我々のアクセラレータでは，Shared Memory に格納することで全てのコアでニューロンの出力を共有する．ここで，第 N 層のニューロンの出力計算から第 $N+1$ 層のニューロンの出力計算に移る際の，各コアと Shared Memory の間のデータ移動の様子を，**図 5** を用いて説明する．まず第 N 層のニューロンの出力を計算する際 (i)，各コアは計算結果を Shared Memory へ書き込む．その後，第 $N+1$ 層のニューロンの出力を計算する際 (ii) には，先ほど第 N 層のニューロンの出力として Shared Memory に格納したデータを第 $N+1$ 層の入力として読み出す．このように Shared Memory を介してコア間でデータを移動することで，ニューロンの出力計算に必要な入力データを全てのコアで共有している．

4. 複数コアへのタスク配置

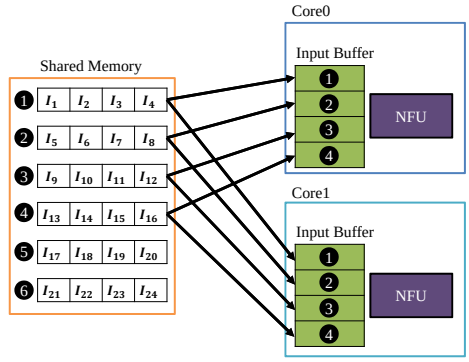
本章では，本稿で検討する 2 つのタスク配置方法の概要を述べる．

4.1 タスク配置の方針

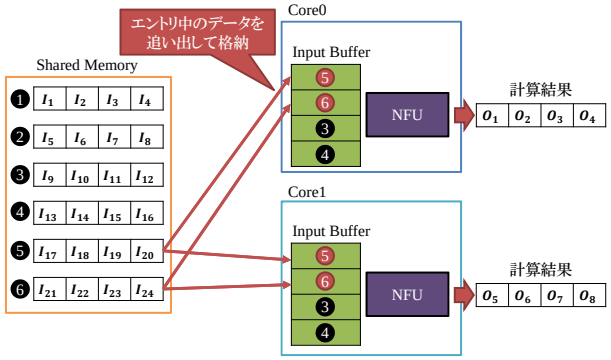
前章で述べたように，ある層のニューロンの出力計算に必要となる入力を読み出す際，および，計算結果として得られた出力を書き戻す際に，Shared Memory を介したデータの移動が発生する．このデータ移動は，Shared Memory を介したコア間通信と捉えることができる．このコア間通信の発生頻度がアクセラレータの性能に影響を与えると考えられるため，本稿ではコア間通信量を最小化することを目指す．コア間通信量は，各コアが前層から与えられる入力を受け取る回数に比例して増加するため，各コアの入力の受け取り方に着目して，2 つのタスク配置の方針を考える．1 つ目はニューロンをグループ化し，各グループに属するニューロンの出力計算を個別のコアに割り当てる方針である．2 つ目は前層から与えられる入力をグループ化し，それぞれのグループに属する入力に対応した計算を個別のコアに割り当てる方針である．次節以降では，この 2 つの方針に沿ってタスクを配置した際の，アクセラレータの動作を説明する．

4.2 ニューロンのグループ化によるタスク配置

本節では，1 つ目の方針である，ニューロンをグループ化したうえで各グループに属するニューロンの出力計算を個別のコアに割り当てる方針について説明する．この方針では，単一のコアに複数のタスクを配置するため，前層から

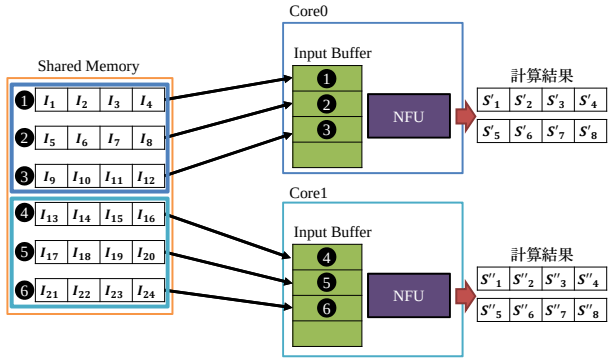


(i) 最初の 4 つのチャンクを Input Buffer に読み出す際の動作

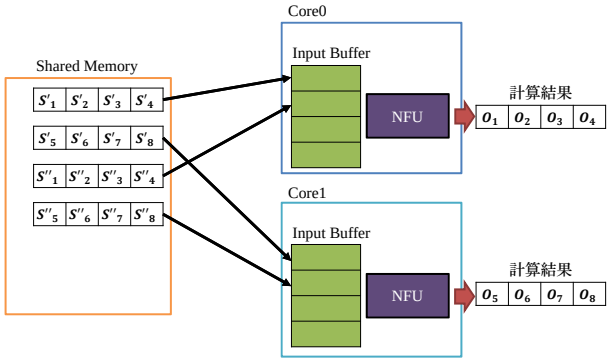


(ii) 残り 2 つのチャンクを Input Buffer に読み出す際の動作

図 6 ニューロンのグループ化によるタスク配置



(i) グループに属する入力に対応した積和計算



(ii) 積和の統合

図 7 入力のグループ化によるタスク配置

与えられる全ての入力を受け取るための通信が必要となるが，各ニューロンの出力計算をそれぞれのコアで完結して行うことができる．24 個の入力値を受け取る 8 つのニューロンの出力をこのタスク配置の方針に沿って計算する際の，データ移動および計算の様子を**図 6** に示す．この例では $N_o = N_i = 4$ としており，前層から与えられる入力 I_1〜I_{24} は 6 つのチャンクに分割されて Shared Memory に格納されている．また，8 つのニューロンを 2 つのグループに分け，各グループに属する 4 つのニューロンの出力計算を Core0 および Core1 に割り当てている．まず各コアは，Shared Memory に格納された 1 つ目の入力チャンク I_1〜I_4 を読み出し，割り当てられたグループに属するニューロンの出力計算に必要となる部分的な積和を計算する．次に，2 つ目の入力チャンク I_5〜I_8 を Shared Memory から読み出し，同様に積和を計算する．その後，計算によって得られた積和を 1 つ目の入力チャンクに対する積和に加算する．この積和計算を前層から与えられる全ての入力に対して行い，入力値 I_1〜I_{24} に対する積和を求める．これに活性化関数を適用することで，Core0 は出力値 O_1〜O_4，Core1 は出力値 O_5〜O_8 をそれぞれ計算する．このようにこの方針では，各コアが前層から与えられる全ての入力を読み出す必要があるが，各ニューロンの出力計算をコア内に閉じて行うことができ，1 つのニューロンの出力を求める途中で Shared Memory にデータを書き戻す必要がない．一方で前層から与えられる全ての入力を順番に Shared Memory から Input Buffer に読み出していくため，Input Buffer 中のデータが追い出され再利用できなくなる可能性が高い．この例では，Input Buffer のエントリ数を 4 と仮定しているため，最初の 4 つのチャンクを Shared Memory から Input Buffer に読み出した時点で，Input Buffer の全てのエントリが埋まる (i)．その状態で，5 つ目および 6 つ目のチャンクを読み出そうとすると (ii)，Input Buffer のエントリ数を超えてしまうため，Input Buffer 中のデータが追い出される．このように，入力チャンクの数が Input Buffer のエントリ数よりも多い場合，入力を順番に読み出すと Input Buffer からデータが追い出されてしまう．

4.3 入力のグループ化によるタスク配置

次に，2 つ目の方針である，前層から与えられる入力をグループ化して各コアにタスクを配置する方針について説明する．この方針では，それぞれのグループに属する入力に対応した積和計算を個別のコアに割り当てる．その後，各コアが求めた積和を統合し活性化関数を適用することでニューロンの最終的な出力を求める．図 6 と同じく，24 個の入力値を受け取る 8 つのニューロンの出力をこのタスク配置の方針に沿って計算する際の，データ移動および計算の様子を**図 7** に示す．このタスク配置では，まず Shared Memory 中の入力データをグループ化する (i)．この例では入力データは Shared Memory 中で 6 つのチャンクに分かれており，それをコア数と同数の 2 つのグループに分

割している．これらのグループを各コアへ割り当て，グループに属する 3 つの入力チャンクに対応した積和を各コアで計算する．Core0 は入力値 I_1〜I_{12} に対する積和 S'_1〜S'_8 を計算し，Core1 も同様に入力値 I_{13}〜I_{24} に対する積和 S''_1〜S''_8 を計算する．そして，これらの積和を一度 Shared Memory へ格納する．その後，積和を統合する処理を行う (ii)．Core0 では，積和 S'_1〜S'_4 と S''_1〜S''_4 の対応する各要素の和をとり，活性化関数を適用することで 4 つのニューロンの最終的な出力値 O_1〜O_4 を計算する．また Core1 でも同様にして 4 つのニューロンの最終的な出力値 O_5〜O_8 を計算する．このようにタスクを配置することで，全てのニューロンに共通して必要な積和計算の各部分計算が単一のコアで行われる．

なお，この例では $N_o = 4$ と仮定しているため，各コアで計算される 8 つの積和 S'_1〜S'_8 および S''_1〜S''_8 は 2 回に分けて計算されるが，Shared Memory からの各入力チャンクの読み出しはそれぞれ 1 回に抑えられる．まず各コアが積和 S'_1〜S'_4 および S''_1〜S''_4 を計算する際，Core0 は入力値 I_1〜I_{12}，Core1 は入力値 I_{13}〜I_{24} をそれぞれ 3 つの入力チャンクとして Shared Memory から読み出すが，この例では Input Buffer のエントリ数を 4 と仮定しているため，3 つの入力チャンクは Input Buffer から溢れることなく全て格納される．次に各コアが積和 S'_5〜S'_8 および S''_5〜S''_8 を計算する際，先程と同じ入力値を Shared Memory から読み出そうとするが，そのデータは Input Buffer にも格納されているため，Shared Memory からではなく Input Buffer から読み出される．これにより，各入力チャンクの 2 回目以降の Shared Memory からの読み出しを省略することができる．つまり，入力をグループ化した際の各グループに属する入力チャンクの数が Input Buffer のエントリ数よりも小さい場合は，Shared Memory から読み出した全ての入力チャンクを Input Buffer に格納することができるため，Shared Memory からの読み出しをそれぞれ 1 回に抑えることが可能となる．したがってこの方針では，各コアで求めた積和を統合するための通信および処理が別途必要となるが，Input Buffer 中のデータが再利用される確率が向上する．

5. 評価

本章では，タスク配置方法が性能へ与える影響を調査するために行ったシミュレーション結果を示し，その結果について考察する．

5.1 評価環境

前章で述べた 2 つのタスク配置方法を，現在我々が開発中のニューラルネットワークアクセラレータ上に実装しシミュレーションにより評価した．ニューラルネットワークアクセラレータの構成は 16 コアとし，コア内の Input Buffer と Output Buffer のエントリ数，および N_i と N_o には Chen ら [6] のアクセラレータと同様の値を設定した．**表 1** に詳細なシミュレーション環境を示す．評価対象として，多層ニューラルネットワークのうちの一層の全結合層の処理に要する実行サイクル数を計測した．この際，層の規模が結果に与える影響を確認するため，ニューロン数および前層から与えられる入力数を，**表 2** に示す 3 つのパラメータセット FC1〜FC3 に設定した場合でそれぞれ評価した．2 つのタスク配置方法において，ニューロンのグループ化によるタスク配置では，表中のニューロン数をコア数と同数の 16 個のグループに分ける．また，入力のグループ化によるタスク配置の場合も同様に，入力を 16 個のグループに分ける．このようなグループ分けにより，FC1〜FC3 の各パラメータに基づいた規模の全ての場合において，どちらのタスク配置方法でも全てのコアに均等にタスクを配置することができる．さらに，それぞれのタスク配置方法について，Input Buffer 中のデータを再利用する場合としない場合に分けて測定することで，Input Buffer 中のデータを再利用することがアクセラレータの性能に与える影響もあわせて確認した．なお，アクセラレータが計算を開始する際，前層から与えられる入力値は，事前に Shared Memory に格納されているものとした．

表 1 シミュレータ諸元

Chip		
cores	16	cores
clock	606	MHz
non-memory IPC	1	
network latency	1	cycle
network band width	100	GB/s
N_i	16	neurons
N_o	16	neurons
Core		
Input Buffer	16	entries
Output Buffer	16	entries
Memory		
latency	1	cycle
port	1	

表 2 層の規模に関するパラメータ

Layer	Neuron	Input
FC1	512	512
FC2	1280	1280
FC3	2560	2560

5.2 評価結果

評価結果を**図 8** および**表 3** に示す．図 8 のグラフは，5 つの異なる評価モデル毎の，前節で述べた全結合層の出力計算に要した実行サイクル数を，表 2 の 3 つのパラメータセットに対応する規模別にまとめて示したものである．各

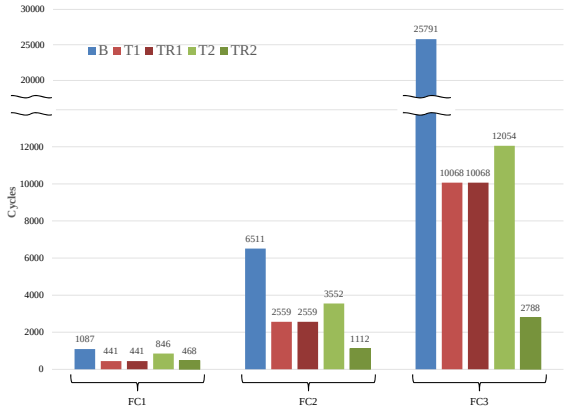


図 8 全結合層の出力計算に要した実行サイクル数

表 3 Shared Memory に対する読み書き回数

	FC1		FC2		FC3	
	Read	Write	Read	Write	Read	Write
(B)	1056	32	6480	80	25760	160
(T1)	66	2	405	5	1610	10
(TR1)	66	2	405	5	1610	10
(T2)	98	34	485	85	1770	170
(TR2)	36	34	90	85	180	170

パラメータセットにおける 5 本のグラフは，左から順に

(**B**) 1 コアで計算するモデル（ベースライン）
(**T1**) ニューロンをグループ化してタスクを配置するモデル
(**TR1**) ニューロンをグループ化してタスクを配置し，さらに Input Buffer 中のデータを再利用するモデル
(**T2**) 入力をグループ化してタスク配置をするモデル
(**TR2**) 入力をグループ化してタスクを配置し，さらに Input Buffer 中のデータを再利用するモデル

にそれぞれ対応している．また表 3 は，各モデルで要した Shared Memory に対する読み出しおよび書き込み回数を示している．なお，表中の値は全て 1 コアあたりの Shared Memory に対する読み書き回数を表しているが，複数コアを用いる評価モデルでは，タスクが均等に配置されることから全てのコアで同数の読み書きが発生する．

評価の結果，全ての規模において 1 コアで計算するモデル (B) と比べ，16 コアを用いて並列に計算するモデルは実行サイクル数が少なくなっている．個別に見ると，FC1 では (T1) の実行サイクル数が (T2) および (TR2) と比べて少なくなっている一方，FC2 および FC3 では (TR2) が最も少ない実行サイクル数となっている．また全ての層において，(TR2) の実行サイクル数が (T2) と比べて少なくなっている．このことから，タスク配置が実行サイクル数に少なからず影響を与えること，また，層に含まれるニューロンの数および前層から与えられる入力の数によって有効なタスク配置が異なることを確認した．

5.3 考察

まず全ての規模の層において，(T1) と (TR1) を比較すると，それぞれの実行サイクル数が同一となっている．また，Shared Memory からの読み出し回数も同一となっている．これは，(TR1) では Input Buffer 中のデータを一度も再利用できていないためだと考えられる．今回評価に使用した中で最も入力の数が少ない FC1 の場合でも入力は $512/N_i = 32$ 個のチャンクに分けられるため，Input Buffer のエントリ数である 16 を超えてしまう．したがって，Input Buffer 中に再利用できるデータが無く，Shared Memory からの読み出しを省略できなかったと考えられる．一方で (T2) と (TR2) を比較すると，(TR2) は (T2) と比べて実行サイクル数，および Shared Memory からの読み出し回数を低減できていることが分かる．例えば今回評価に使用した中で最も入力の数が多い FC3 において，前層から与えられる入力をコア数と同数の 16 個のグループに分けると，各グループには 160 個の入力が属する．よって各コアが，あるグループの入力に対して積和を計算していく際，入力は $160/N_i = 10$ 個のチャンクに分けられる．これは Input Buffer のエントリ数 16 よりも少ないため，全ての入力チャンクを Input Buffer に格納することができる．これにより，各入力チャンクの Shared Memory からの読み出しをそれぞれ 1 回のみに抑えることができる．

次に層の規模ごとに見ると，FC1 では (T1) よりも (TR2) の実行サイクル数が多くなっている．これは，入力のグループ化によるタスク配置をとった場合に必要となる，部分的な積和の書き戻し処理の影響が大きいためだと考えられる．各コアの Shared Memory への書き込み回数を見ると，(T1) が 2 回であるのに対し，(TR2) は 34 回となっている．このように Shared Memory への書き込み回数が増加したことで，Input Buffer 中のデータ再利用による実行サイクル数削減の効果が打ち消されてしまったと考えられる．一方で，FC2 および FC3 を見ると，(T1) よりも (TR2) の実行サイクル数が少なくなっている．FC2 の場合，Shared Memory への書き込み回数は (T1) が 5 回であるのに対し，(TR2) は 85 回となっている．一方で，Shared Memory からの読み出し回数は (T1) が 405 回であるのに対し，(TR2) が 90 回となっている．つまり，(TR2) は (T1) よりも Shared Memory への書き込み回数が多い一方で，Shared Memory からの読み出し回数が大幅に少ないことで，実行サイクル数が低く抑えられていると考えられる．これらのことから，Shared Memory を介したコア間通信量が最小になるようにタスクを配置することで，実行サイクル数を削減できることが分かる．

なお，今回は Input Buffer 中のデータのみが再利用可能な場合を想定したが，今後は Output Buffer 中のデータを再利用することがアクセラレータの性能に与える影響も検討する必要がある．今回検討した 2 つのタスク配置

に共通して，ある層に含まれるニューロンの出力計算から次の層のニューロンの出力計算に移る際，各コアは計算結果として得られた出力を一度 Shared Memory に格納し，自身が書き込んだデータを Shared Memory からすぐに読み出すという動作が必要である．また，入力のグループ化によるタスク配置において，Shared Memory を介して全てのコアで部分的な積和を共有する際にも同様の動作が必要である．このとき，Output Buffer を介して Shared Memory に書き込んだデータを，Output Buffer 中に保持しておき，NFU が Shared Memory に書き込んだデータを読み出す際に，Output Buffer から同じデータを読み出すことで Shared Memory からの読み出しを省略できる．このように，各コアが Output Buffer 中のデータを再利用して Shared Memory に書き込んだデータをすぐに読み出す動作を抑制することで，Shared Memory からの読み出し回数を減らすことができると考えられる．

また，Output Buffer 中のデータが再利用可能になることで，隣接層間でのニューロンの接続関係を考慮したタスク配置が有効になると考えられる．ディープラーニングに用いられるニューラルネットワークは全結合層だけではなく，畳み込み層 [7] と呼ばれる隣接層間のニューロン同士が部分的にのみ接続した層構造を持つものが多い．さらに，近年は学習後のニューラルネットワークからシナプスを間引くことで計算量の削減を図る研究が盛んに行われている [8], [9]．このような隣接層間のニューロンが部分的にのみ接続しているネットワークの出力をアクセラレータで計算する場合を考えると，ある層のニューロンの出力計算には，前層の一部のニューロンの出力のみが入力として必要となり，各コアが計算したニューロンの出力は全てのコアで共有する必要がない．このような場合，隣接層間で接続しているニューロンの出力計算が，できるだけ同一のコアで行われるようにタスクを配置することで，ニューロンの出力を共有するためのコア間通信を削減することができる．これは，各コアは計算結果として得られたニューロンの出力を Shared Memory へ書き込まず，Output Buffer に一時的に格納しておき，次の層のニューロンの出力を計算する際に Output Buffer から読み出すことで実現できると考えられる．

6. おわりに

本稿では，高性能なハードウェアアクセラレータの実現に向けて，複数コアへのタスク配置に着目し，タスク配置がハードウェアアクセラレータの性能に及ぼす影響を調査した．またその結果から，コア間通信量最小化のための適切な配置について考察した．2 つのタスク配置方法について 3 種類の規模の全結合層を用いて評価を行った結果，層の規模によって有効なタスク配置が異なることを確認した．しかし，今回検討したタスク配置方法は Input Buffer 中のデータのみが再利用可能な場合を想定しており，Output Buffer 中のデータを再利用することでさらにコア間通信量を削減できると考えられる．したがって今後，Output Buffer 中のデータを再利用することを考慮したタスク配置が，アクセラレータの性能に与える影響を検討していきたい．

参考文献

[1] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper With Convolutions, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9 (2015).
[2] Nair, V. and Hinton, G. E.: Rectified linear units improve restricted boltzmann machines, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814 (2010).
[3] Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R. and Modha, D. S.: A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm, *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, IEEE, pp. 1–4 (2011).
[4] Temam, O.: A defect-tolerant accelerator for emerging high-performance applications, *ACM SIGARCH Computer Architecture News*, Vol. 40, No. 3, pp. 356–367 (2012).
[5] Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y. and Temam, O.: Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning, *ACM Sigplan Notices*, Vol. 49, No. 4, ACM, pp. 269–284 (2014).
[6] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N. et al.: Dadiannao: A machine-learning supercomputer, *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, pp. 609–622 (2014).
[7] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P.: Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324 (1998).
[8] Han, S., Pool, J., Tran, J. and Dally, W.: Learning both Weights and Connections for Efficient Neural Network, *Advances in Neural Information Processing Systems*, pp. 1135–1143 (2015).
[9] Polyak, A. and Wolf, L.: Channel-Level Acceleration of Deep Face Representations, *Access, IEEE*, Vol. 3, pp. 2163–2175 (2015).