

ベクトルプロセッサを用いた Direct Sparse Convolution の最適化

大野 善之^{1,a)} 石坂 一久¹

概要：Convolutional Neural Network (CNN) は、高い認識精度を有する一方で、Convolution 演算にかかる演算量が多いという問題がある。演算量が多い問題に対し、CNN の精度を落とさずにネットワークをスパース化し演算量を削減することで高速化できることが分かっている。しかし、ネットワークをスパース化することにより、データアクセスがランダムになるため、データアクセスを効率化しないと高い演算性能が得られなくなるという問題がある。そこで、本稿ではスパースなネットワークを用いた Convolution 演算を高効率に実行する方式を新たに提案する。提案方式では、以下の 2 点で入出力にかかるメモリアクセスを削減する。(1) $K \times P \times Q$ の 3 次元の出力のうち、 K 方向の同一直線上にある K 個の座標をまとめて計算する。このとき、 K 個の座標が計算完了するまでレジスタに保持しておくことで、出力に対するメモリアクセスを削減する。(2) $K \times C \times R \times S$ の 4 次元のスパースなフィルタについて、 $C \times R \times S$ の 3 次元上の座標それぞれに対して、 K 方向にある非零の重みをリスト化して保持する。フィルタ演算時には、1 回の入力データのメモリロードに対して、リスト化しておいた全ての非零の重みと積算させることで、入力データのメモリアクセスを削減する。本方式を、SX-ACE のベクトルアーキテクチャを用いて評価したところ、1 コアの場合 88.7%、ソケット全体 (4 コア) の場合 84.5% と、高い効率で演算できることを確認した。

Optimization of Direct Sparse Convolution on Vector Processor

YOSHIYUKI OHNO^{1,a)} KAZUHISA ISHIZAKA¹

1. はじめに

近年、認識や検出、判断などを計算機で行う機械学習手法の 1 つである Deep Learning が注目をあびている。Deep Learning では、生物の神経回路網を模倣した Neural Network を用いることで、高い認識性能を得ている。Neural Network は、入力層、出力層、隠れ層 (中間層とも言う) から構成され、層と層の間には、ニューロン同士のつながりの強さを示す重みがある。入力層にデータを与えることで、入力層から隠れ層、隠れ層から出力層へと、データに重みが掛けあわされて伝搬していき、出力層の信号が定まり、結果を得るという構造である。

特に、画像や動画などは、領域の一部の (局所的な) 情

報だけを抽出して伝搬させることが有効であり、その機能を有する層が Convolutional Layer である。Convolutional Layer を持つ Neural Network である Convolutional Neural Network (CNN) は、画像の認識に対して高い性能が得られている [1] [2]。しかしながら、CNN は高い認識性能を有する一方で、Convolutional Layer の Convolution 演算の演算量が多く、計算時間を要するという問題がある。

CNN に関して、近年の研究により、ネットワークの全ての重みが必須というわけではなく、精度を落とすことなくネットワークの重みの多くを零にできることがわかっている [3]。ネットワークの重みを零にすることは、ネットワークのスパース化と呼ばれる。ネットワークのスパース化の目的は、ネットワークの重みにかかるデータ量を削減することにあつた。CNN が多くの計算時間を要するという問題に対して、ネットワークのスパース化を利用するこ

¹ 日本電気株式会社 システムプラットフォーム研究所
System Platform Research Laboratories, NEC Corporation

^{a)} y-ohno@ji.jp.nec.com

とで, CNN の演算量自体を削減し高速化ができることが分かっている [4] [5]. しかしながら, ネットワークをスパース化すればするほど演算量は減り高速化できると考えられるが, 先行研究においては, スパース化率に比例した性能が得られていない. 例えば, Li ら [5] の報告では, ネットワークの非零要素率が 9% で, 3.1 倍から 7.3 倍の性能向上という結果である. 理想的には, 演算量が 9% になるので 11 倍の向上が得られるはずだが, 理論性能に対して 28% から 66% ほどの性能しか得られていない.

そこで, 本稿では スパースなネットワークを用いた Convolution 演算 (Sparse Convolution) を高効率に演算する方式を新たに提案する. 我々は, Sparse Convolution は, 単純な演算である一方で, データがスパースになることにより, データアクセスが複雑になるため, 演算に対してデータアクセスコストが増加し, 演算性能が低くなると考えた. 我々の提案方式では, 以下の 2 点により, データアクセスの効率化を行う.

- 1 回の入力データのメモリロードに対して, 複数のフィルタを同時に適用する. そのために, 4 次元で表現されるフィルタ (k, c, r, s) について, K 方向に同一位置にある座標 (c, r, s) ごとに, 非零の重みをリスト化し保持する.
- 複数の出力チャネルをまとめて演算する. まとめて演算する出力は, 結果が確定するまでレジスタに保持し続ける.

本方式においては, 大容量のデータレジスタを有する方が有利となるため, 大容量のベクトルレジスタを有する SX-ACE 向けに実装した. 本稿では, その評価結果についても報告する.

本稿の章構成は以下のとおりである. まず, 2 章で, CNN および, そのスパースネットワーク版である Sparse CNN について説明する. 次に, 3 章で, 本稿で提案する Sparse Convolution の演算方式を述べる. 4 章では, Sparse CNN に関する先行研究について述べる. 5 章で, 我々の方式の性能評価結果, および, 先行研究との比較を行う. そして, 最後の 6 章でまとめを述べる.

2. Sparse Convolutional Neural Network

本章では, Sparse Convolutional Neural Network (Sparse CNN) および, その基本となる Convolutional Neural Network (CNN) について述べる.

2.1 Convolutional Neural Network

近年, 認識や検出, 判断などを計算機で行う機械学習手法の 1 つである Deep Learning が注目をあびている. Deep Learning は, 生物の神経回路網を模倣した Neural Network (NN) を用いた機械学習手法である. NN は, 入

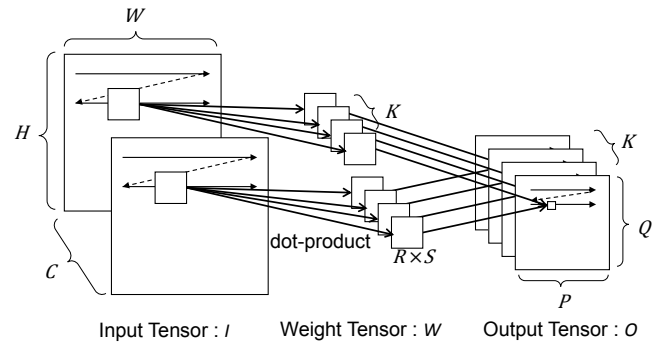


図 1 Convolutional Layer の概要

力層, 出力層, 隠れ層から構成され, 層と層の間には, 各ニューロン同士のつながりの強さを示す多数の重みがある. 入力層にデータを与えることで, 入力層から隠れ層, 隠れ層から出力層へと, データに重みが掛けあわされて伝搬していき, 出力層の信号が定まり, 結果を得るという構造である. Deep Learning は, 隠れ層が何層にもつながっていること, つまり, 層が深くなっていることが特徴であり, それが名前の由縁となっている.

Deep Learning の中でも, 画像や動画などの認識に有効であるのが, Convolutional Neural Network である [1] [2]. 画像や動画などは, 領域の一部の (局所的な) 情報だけを抽出して伝搬させることが有効であり, その機能を有するのが Convolutional Layer である. CNN は, Convolutional Layer を有する NN である. しかしながら, CNN は高い認識性能を有する一方で, Convolutional Layer の Convolution 演算 の演算量が多く, 計算時間を要するという問題がある.

図 1 は, Convolutional Layer で行う Convolution 演算の概要を示す図である. Convolutional Layer では, 幅 W × 高さ H の C 枚の入力に対し, それぞれ幅 R × 高さ S の K 枚のフィルタをスライドさせながら適用し, 幅 P × 幅 Q の K 枚の出力を作成する. 入力を $C \times W \times H$ の 3 次元配列 I , フィルタを $K \times C \times R \times S$ の 4 次元配列 W , 出力を $K \times P \times Q$ の 3 次元配列 O と表したとき, 出力 1 ピクセルを計算する式は,

$$O(k, p, q) = \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} W(k, c, r, s) \times I(c, p+r, q+s)$$

と表現できる. 出力の全ピクセルを計算するには, $K \times P \times Q \times C \times R \times S$ 回の積和演算が必要となり, 大きな計算量となる.

Convolution 演算を高速に演算するための方法として, 行列行列積の形式に変換するという方法がある. この方法の利点は, 多くのアーキテクチャで実装・最適化されている行列行列積のライブラリを使うことで, 簡単に高効率に演算できるという点である. しかし, 3 次元や 4 次元のデータを, 行列行列積の形式に合わせて, データレイアウト変換しなければならず, このレイアウト変換がオーバーヘッ

```

for k = 0 to K-1
  for p = 0 to P-1
    for q = 0 to Q-1
      for c = 0 to C-1
        for r = 0 to R-1
          for s = 0 to S-1
            O(k,p,q) += W(k,c,r,s) * I(c,p+r,q+s)

```

図 2 DirectConvolution の 6 重ループ

ドとなる。また、Convolution 演算は、上述の式をそのままコード化した、図 2 のような 6 重のループで演算することでも実現できる。これは、直接 Convolution 演算を施すため、Direct Convolution と呼ばれる。ちなみに、図 2 の 6 重ループは、ループの順序が入れ替え可能であり、図 2 は Direct Convolution のループ構造の一例である。Direct Convolution は、行列積に変換する方式と違い、データレイアウトの変更は必要がない。しかし、演算するアーキテクチャに応じて性能が出るように、ループインターチェンジやブロッキング等を駆使して最適化する必要がある。

2.2 Sparse Convolutional Neural Network

CNN のネットワークのためのメモリサイズの削減のため、ネットワークの重みを削減（ネットワークをスパース化）するという研究がある [3]。

Han ら [3] は、学習時にネットワークの重みを枝刈りしながら学習し、学習後には、学習した重みを量子化しハフマン符号化することで、精度をおとすことなく、ネットワークを保持するためのメモリサイズを削減している。その削減量は、AlexNet[1] で 35 倍、VGG-16[2] で 59 倍である。メモリ削減量の多くは、Fully Connected Layer の重みを枝刈りして圧縮したことによるものだが、Convolutional Layer に限っても、枝刈りにより AlexNet で 44%、VGG-16 で 33% の重みの削減ができています。

この重みの削減は、メモリ量を削減するだけでなく、演算量の削減にもつながる。前節のとおり、Convolution 演算は、フィルタの重みと入力データの積を出力に足し込む演算であるため、重みが 0 の場合は演算の必要がない。したがって、重みを削減した分だけ演算量を削減することができる。

しかし、演算量を削減することができたとしても、演算の高速化につながるわけではない。Convolution 演算の演算方式である行列積形式にしたとしても、密行列演算向けの行列積ライブラリを利用すれば、演算量の削減効果を得ることができない。図 3 で表すような、非零の重みだけを用いるような Direct Convolution で演算することで、演算量の削減効果を得ることができる。

ただし、図 3 で示すとおり、基本構造は 3 変数を入力として 1 回の積和演算を行うという構造であるため、変数が単精度の場合、Byte/Flop (B/F) = 6 という、高いメモリバンド幅が必要とされる。したがって、データアクセスを

```

for p = 0 to P-1
  for q = 0 to Q-1
    for each NonZeroWeight W(k,c,r,s) in all W
      O(k,p,q) += W(k,c,r,s) * I(c,p+r,q+s)

```

図 3 Direct Sparse Convolution の基本構造

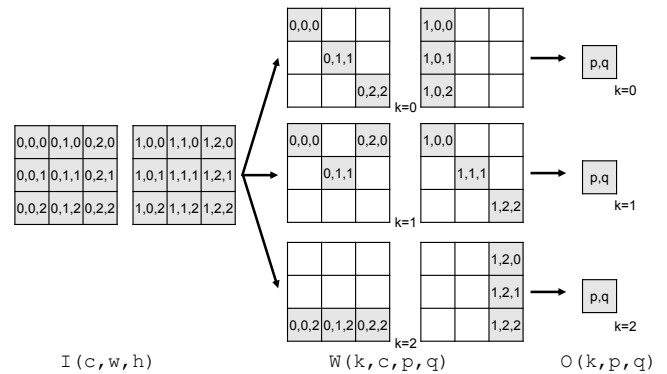


図 4 入出力の位置を固定した場合の一例

座標 (フィルタ番号, 重み) のリスト

(c, r, s) : (k, W(k, c, r, s)), ...

(0, 0, 0)	: (0, W(0, 0, 0, 0)), (1, W(1, 0, 0, 0))
(0, 2, 0)	: (0, W(0, 0, 2, 0))
(0, 1, 1)	: (0, W(0, 0, 1, 1)), (1, W(1, 0, 1, 1))
(0, 0, 2)	: (2, W(2, 0, 0, 2))
(0, 1, 2)	: (2, W(2, 0, 1, 2))
(0, 2, 2)	: (0, W(0, 0, 2, 2)), (2, W(2, 0, 2, 2))
(1, 0, 0)	: (0, W(0, 1, 0, 0)), (1, W(1, 1, 0, 0))
(1, 0, 1)	: (1, W(1, 1, 0, 1))
(1, 1, 1)	: (1, W(1, 1, 1, 1))
(1, 2, 1)	: (2, W(2, 1, 2, 1))
(1, 0, 2)	: (0, W(0, 1, 0, 2))
(1, 2, 2)	: (1, W(1, 1, 2, 2)), (2, W(2, 1, 2, 2))

図 5 フィルタの座標ごとに、フィルタ番号と非零の重みのペアのリスト化 (図 4 のフィルタの場合)

効率化して、B/F を下げる必要がある。図 3 の構造も、通常の Direct Convolution と同様に、ループの演算順序は入れ替え自由である。したがって、データの再利用性が効くように、最適化を施すことが可能である。本稿では、スパースなネットワークモデルを用いた Direct Convolution (Direct Sparse Convolution) の、データアクセスを効率化した演算方式を提案する。

なお、本稿では、ネットワークの重みを決定する学習フェーズは対象とせず、ネットワークの重みが決定した後、重みを使うだけの推論フェーズを対象とする。

3. 提案方式

本章では、提案の Sparse Convolution の演算方式について説明する。

2 章で述べたとおり、Sparse Convolution の基本構造は、B/F が 6 と非常に大きい。したがって、データを再利用してメモリアクセスを効率化することが重要となる。本提案方式の特徴を以下にまとめる。

- $K \times P \times Q$ の 3 次元の出力のうち、 K 方向の同一直線

Algorithm 1 Vector Sparse Direct Convolution

```

1: for all Output Coordinate (p, q) do
2:   (i, j) = Input Coordinate corresponding Output (p, q)
3:   for k = 0 to K - 1 do
4:     Initialize Register(Ok)
5:   end for
6:   for c = 0 to C - 1 do
7:     for r = 0 to R - 1 do
8:       for s = 0 to S - 1 do
9:         if nonZeroWeights(c, r, s) isn't empty then
10:          load I(c, i+r, j+s) to Register(In)
11:          for all pair(k, w) in nonZeroWeights(c, r, s) do
12:            Register(Ok) += w × Register(In)
13:          end for
14:        end if
15:      end for
16:    end for
17:  end for
18:  for k = 0 to K - 1 do
19:    Store Register(Ok) to O(k, p, q)
20:  end for
21: end for

```

上にある K 個の座標をまとめて計算する。このとき、 K 個の座標が計算完了するまでレジスタに保持しておくことで、出力に対するメモリアクセスを削減する。

- $K \times C \times R \times S$ の 4 次元のスパースなフィルタについて、 $C \times R \times S$ の 3 次元上の座標それぞれに対して、 K 方向にある非零の重みをリスト化して保持する。フィルタ演算時には、1 回の入力データのメモリロードに対して、リスト化しておいた全ての非零の重みと積算させることで、入力データのメモリアクセスを削減する。

図 3 で示すとおり、 p と q を固定した場合、すなわち、入力に対するフィルタ適用位置や出力ピクセルが固定された場合、 K 方向の同一直線上にある K 個の出力ピクセルに値を足しこみ続ける演算になる。その間は、出力結果はメモリにおいておくのではなく、演算器に最も近いレジスタに保持し続けるほうがよい。また、図 4 は、入出力の位置を固定した場合の一例を示している。例えば、入力 $I(0, 0, 0)$ は、フィルタ番号（出力の観点からは、出力チャンネル番号） $k=0$ および $k=1$ で共通して利用される。このように、複数のフィルタで共通して利用可能な入力データは、都度メモリロードするのではなく、1 回のメモリロードで、複数のフィルタに対して適用するほうがよい。そのために、図 5 のように、 $K \times C \times R \times S$ の 4 次元で表現されるフィルタ (k, c, r, s) について、3 次元座標 (c, r, s) ごとに、フィルタ番号と非零の重みのペアをリスト化したもの (`nonZeroWeights`) をあらかじめ作成して保持しておき、Convolution の計算時に利用するようにし、入力をロードしたときに適用すべきフィルタを特定できるようにしておく。

Algorithm 1 は、非零の重みのリスト (`nonZeroWeights`)

```

for k = 0 to K-1
  for j in [W.rowptr(k), W.rowptr(k+1)]
    off = W.colidx(j) ; w = W.value(j)
    for p = 0 to P-1
      for q = 0 to Q-1
        O(k, p, q) += w * I(f(off, p, q))

```

図 6 Li ら [5] の sparse convolution の擬似コード

を作成した後の、本提案方式の Sparse Convolution の計算本体の擬似コードである。1 行目から始まる最外ループでは、1 チャンネルの出力サイズ $P \times Q$ ピクセル分を走査しており、その内部で K チャンネル分の出力を 1 度に計算している。計算の本体は、6 行目から 17 行目の 4 重ループであり、本体ループの前に K チャンネル分の結果保持レジスタを初期化し (3 行目から 5 行目)、本体ループの後に K チャンネル分の結果をメモリにストアする (18 行目から 20 行目) という構造である。本体ループでは、レジスタに対する演算であるため、結果に関するデータアクセスが発生しない。6 行目からの 3 重ループはフィルタの座標を走査するループであり、それぞれのフィルタの座標において非零の重みがある場合 (9 行目) のみ、対応する入力データをメモリロードし (10 行目)、必要とする出力にのみ重みとの積を足し込むという構造になっている。

本方式は、SIMD 演算させる (ベクトル化する) ことができる。最外の出力座標間には、全く依存性がないので、ベクトル化が可能である。また、複数の入力セットを同時に計算する (バッチ処理) の場合に、バッチ方向でのベクトル化も可能である。ベクトル化した場合、ベクトル演算になるのは、4 行目のレジスタ初期化、10 行目の入力データのメモリロードや、12 行目の積和演算、19 行目のメモリストアである。それ以外はスカラ処理となる。11 行目の `nonZeroWeights` の要素数が n である場合、1 回のベクトルロードに対して、 n 回のベクトル積和演算であるため、ベクトル演算の B/F は $4/(2n) = 2/n$ となり、メモリアクセスが効率化されたことがわかる。ベクトル演算以外にも、スカラ命令によるフィルタの重みに関わるメモリアクセスが発生するが、ベクトル長が長ければスカラ命令のコスト比率は小さくなる。

4. 関連研究

Liu ら [4] は、Sparse Convolution の演算方法として、3 次元の入力データと 4 次元のスパースなネットワークを、それぞれ、2 次元の密行列と 2 次元の疎行列に変換し、密行列と疎行列の積の形式に変換し、密行列疎行列の積の高効率な演算方法を提案している。この方法は、あらかじめ行列行列積の形式にデータ変換をしなければならず、オーバーヘッドとなる。一方我々は、入力データのデータ変換を行わずに、直接 Convolution 演算を行う方法を提案している。

Li ら [5] は、我々と同様にスパースなネットワークモデルに対する、Direct Convolution の方式を提案している。本方式は、SkimCaffe という名前で ソースコードが公開されている [6]。図 6 が Li らの方式の sparse convolution の基本構造を表す擬似コードである。Sparse なネットワークを compressed sparse row(CSR) 方式で保持しておき(図中 w), K 個のフィルタそれぞれ順番に適用して K 枚の出力を順番に計算している。Li らの方式では、最内の p と q の 2 重ループを ブロッキングでき、入力 が キャッシュに収まるようにブロッキングすることで、メモリアクセスを削減している。一方で、我々の方式では、入力データ 1 回のメモリロードで K 個のフィルタ間で共有し、メモリアクセスを削減している。

5. 性能評価

本章では、性能評価について述べる。

5.1 SX-ACE のベクトルアーキテクチャ

我々の提案する Direct Sparse Convolution は、レジスタが大きく、また、ベクトル演算可能でベクトル長の長いアーキテクチャで有効である。そこで、そのようなアーキテクチャであるベクトルプロセッサを搭載した SX-ACE を用いて、提案方式を実装した。ここでは、SX-ACE のプロセッサアーキテクチャについて述べる。

SX-ACE は、2013 年に発売された NEC 製のベクトルプロセッサ搭載コンピュータである。表 1 は SX-ACE のプロセッサ諸元であり、図 7 は SX-ACE のプロセッサの構成を示している。SX-ACE は、4 個の CPU コアから構成される。各コアは、ベクトル演算を行う Vector Processing Unit (VPU)、スカラ演算を行う Scalar Processing Unit (SPU)、そして、ソフトウェア制御可能なキャッシュである Assignable Data Buffer (ADB) を 1MB 有している。VPU では、72 本のベクトルレジスタを有しており、ベクトルレジスタそれぞれが、最大 256 要素(1 要素の最大は 64bit)のデータを保持できる。VPU では、ベクトルレジスタを入出力とする 最大ベクトル長 256 のベクトル演算を施すことができ、コアあたりの最大演算性能は 64 GFLOPS である。また、VPU と ADB の間は 256 GB/s のバンド幅を有する。各コアとメモリとのバンド幅は、最大 256 GB/s であるが、CPU 全体でのメモリバンド幅は 256 GB/s であるため、4 コアが均等にメモリアクセスした場合は、各コア最大 64 GB/s となる。これにより、各コアの B/F は 1.0 ~ 4.0 となる。

次に、SX-ACE 向けの実装について述べる。Algorithm 1 の擬似コードのとおり、出力用のレジスタ K 個、入力用のレジスタ 1 個 が必要である。SX-ACE では、積和命令がないため積算と加算に分ける必要がある。そのため、積の一時保持用にレジスタが追加で 1 個必要であり、合

表 1 SX-ACE プロセッサ諸元

Core	
Performance	64 GFLOPS
ADB Size	1 MB
ADB Bandwidth	256 GB/s
Memory Bandwidth	64 GB/s - 256 GB/s
Byte/Flop	1.0 - 4.0
CPU	
Cores	4
Performance	256 GFLOPS
Memory Bandwidth	256 GB/s
Byte/Flop	1.0

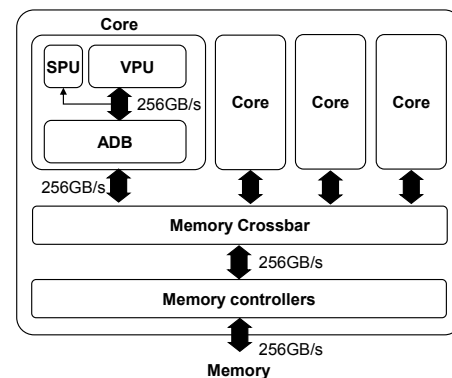


図 7 SX-ACE プロセッサ構成

計 $K + 2$ 個のレジスタが必要である。SX-ACE には、72 個のベクトルレジスタがある。我々の実装においては、64 個のレジスタを出力用のバッファに用いることとし、出力チャネル数が 64 より大きい場合は、64 チャネルずつ分割して計算するようにした。

5.2 ランダムなスパースモデルにおける性能評価

我々の提案方式の有効性を示すために、いくつかのネットワークを用い演算効率を測定した。用いたネットワークは、LeNet-5[7], AlexNet[1], VGG-16[2] である。これらのネットワークに対して、文献 [3] で達成したスパース化率であるような、ランダムなスパースモデルを作成した。表 2 が、ネットワークのパラメータ一覧である。

ベクトル化は、3 章のとおり、出力座標方向やバッチ方向でできるが、本評価ではバッチサイズ方向で行うものとし、バッチサイズを 256 とした。上記のネットワークそれぞれに対して、256 セットの入力データを作成し、SX-ACE 1 コア、および、4 コアでの演算効率を測定した。4 コアの場合は、タスクの総量は変更せず、スレッド並列により出力座標方向でタスクを分割した。

図 8 が、本評価結果のグラフである。横軸の Layer ID は、表 2 の ID に対応している。それぞれ、左側の棒が 1 コアの演算性能、右側の棒が 4 コアの演算性能を示している。ここで言う演算性能は、演算機の理論ピーク演算性能

表 2 評価に用いたネットワークパラメータ

ID	Layer	C	H,W	K	R,S	u,v	Sparcity%
1	Lenet-5 conv1	1	28	20	5	1	34
2	Lenet-5 conv2	20	12	50	5	1	88
3	AlexNet conv1	3	227	96	11	4	16
4	AlexNet conv2	96	27	256	5	1	62
5	AlexNet conv3	256	13	384	3	1	65
6	AlexNet conv4	384	13	384	3	1	63
7	AlexNet conv5	384	13	256	3	1	63
8	VGG-16 conv1.1	3	224	64	3	1	42
9	VGG-16 conv1.2	64	224	64	3	1	78
10	VGG-16 conv2.1	64	112	128	3	1	66
11	VGG-16 conv2.2	128	112	128	3	1	64
12	VGG-16 conv3.1	128	56	256	3	1	47
13	VGG-16 conv3.2	256	56	256	3	1	76
14	VGG-16 conv3.3	256	56	256	3	1	58
15	VGG-16 conv4.1	256	28	512	3	1	68
16	VGG-16 conv4.2	512	28	512	3	1	73
17	VGG-16 conv4.3	512	28	512	3	1	66
18	VGG-16 conv5.1	512	14	512	3	1	65
19	VGG-16 conv5.2	512	14	512	3	1	71
20	VGG-16 conv5.3	512	14	512	3	1	64

との比である。この値が高いほど、性能が良いことを示している。図 8 のとおり、並列化の有無に関わらず、概ね 8 割から 9 割の演算性能を達成していることが分かる。20 Layer の演算性能の平均は、1 コアの場合 88.7%、4 コアの場合 84.5% を達成した。一部の場合で、並列化により、大きく性能が定価して場合がある。これは、並列化によりコア間のインバランスや、コア間のメモリバンクコンフリクトが顕在化したのではないかと考えられる。

5.3 実スパースモデルにおける性能評価

先の評価は、ランダムに作成した Sparse Network での評価であったが、公開されている、実際に高い認識性能を有する Sparse Network を用いた評価も行った。用いたのは、文献 [3] により作成された、AlexNet のスパース版である。これは、WEB 上で公開されている [8]。図 9 は、先述のランダムなスパースモデルの場合と、実際に作成されたスパースモデルの場合の結果を図示している。図 9 のとおり、ランダムなモデルと実際のモデルの性能に差異はなく、ランダムなモデルの評価で、性能の傾向を見ても問題ないといえる。

5.4 他方式 (SkimCaffe) との比較

最後に、他の方式との比較として、Intel Skim Caffe [6] との比較を行った。Skim Caffe は、Xeon や Xeon Phi 向けに Sparse Convolution を最適化している。ただし、全てのネットワークに対して最適化しているわけではなく、AlexNet や Google Net 等、特定のネットワークに限定している。先述の、実スパースモデルにおける性能評価で用いた AlexNet のスパースモデルを SkimCaffe を用いて性能測定した。評価には、Xeon E5-2660 v3 を 2 ソケット搭載したサーバを用い、1 コアのみを使って実行した場合と、1 ソケット全体 (12 コア) を使って実行した場合の演

算効率を測定した。図 10 は、我々の提案方式を SX-ACE で実行した場合と、SkimCaffe を Xeon で実行した場合の比較グラフである。SkimCaffe(Xeon) において、Conv1 の性能が極端に低いのは、Conv1 向けのスパース向け最適化が行われていないためである。Conv 1 以外の層は、4 割前後の演算性能である。一方で、我々の方式は 9 割弱の演算性能を達成している。SkimCaffe では、入力データをキャッシュブロッキングしてメモリからのロードを削減している。キャッシュブロッキングでは、メモリからのロードをキャッシュからのロードにすることで、データアクセス時間を短縮できるが、あくまでもメモリロード命令の実行である。したがって、1 回の積和演算に対して、ロード命令のためのアドレス計算が必要であったり、キャッシュに対するレイテンシであったりと、オーバーヘッドが見えてしまう。一方で我々の方式は、複数回の積和演算に対して、ロード命令が 1 回でよい。この点が演算効率の差に出ていると考えられる。

6. おわりに

本稿では、Direct Sparse Convolution の最適化方式を提案した。

Sparse Convolution は、非零のフィルタ値と、対応する入力データをロードし、それらの積を出力に足しこむという演算の連続である。ナイーブにはメモリアクセスネックになるが、本稿の提案方式では、以下の 2 点により Sparse Convolution にかかるメモリロードの削減を行った。(1) あらかじめ、 $K \times C \times R \times S$ の 4 次元で表現されるフィルタについて、 $C \times R \times S$ の 3 次元上の座標それぞれに対して、 K 方向の同一直線上にある非零値をリスト化しておき、Convolution の演算時には、入力データの 1 回のメモリロードで、あらかじめリスト化しておいた非零値の全てと演算することで、入力データのロード回数を削減する。(2) (1) の演算は、 K 個のフィルタをまとめて適用すること、 K 個の出力チャネルをまとめて演算することを意味し、それら K 個の出力チャネルを演算する間は、演算結果をレジスタに保持し続けることで、メモリアクセスを削減する。

提案方式を SX-ACE で評価したところ、1 コアの場合 88.7%、ソケット全体 (4 コア) の場合 84.5% と、高い演算効率であることを確認した。しかしながら、4 コア並列の場合、コア間のタスクのインバランスや、コア間のメモリアクセスによるバンクコンフリクト等が顕在化する場合もあり、改良が必要である。また、本稿では CNN で最も演算量が多い Convolutional Layer をターゲットとしたが、Fully-Connected Layer もネットワークがスパースにできることが分かっており、Fully-Connected Layer がスパースな場合の効率的な実装の検討も今後の課題である。

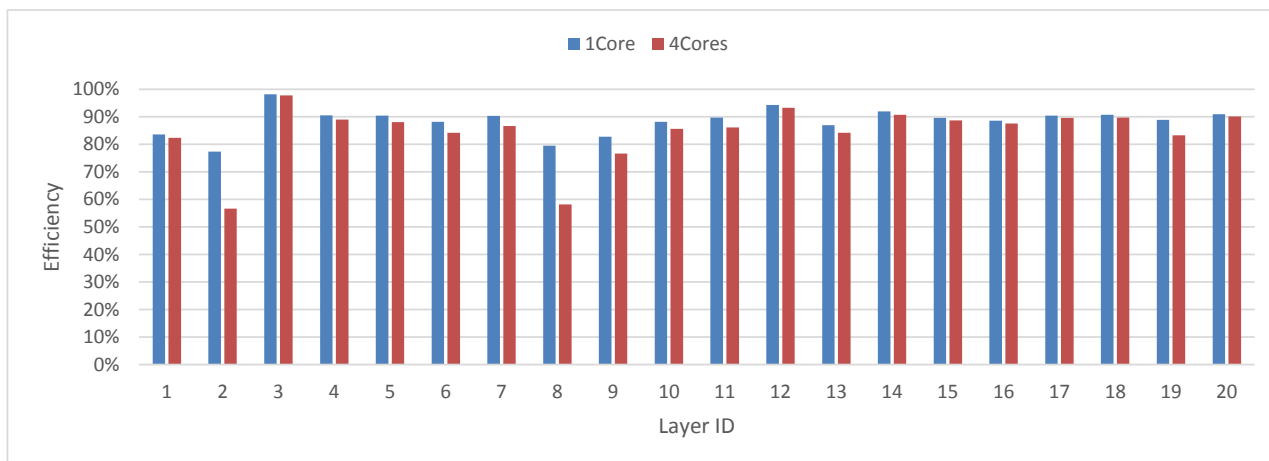


図 8 ランダムな Sparse Network を用いた場合の演算性能

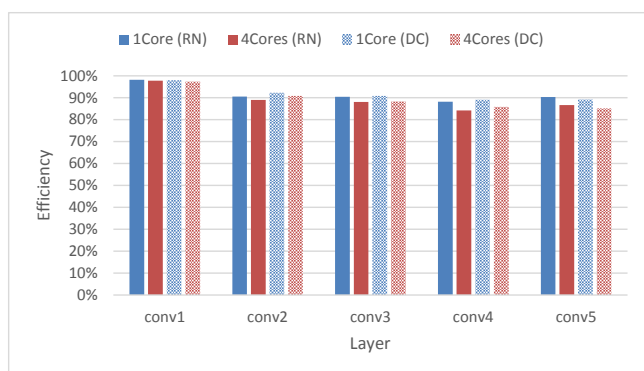


図 9 Deep Compression AlexNet を用いた評価
RN : Random, DC : Deep Compression

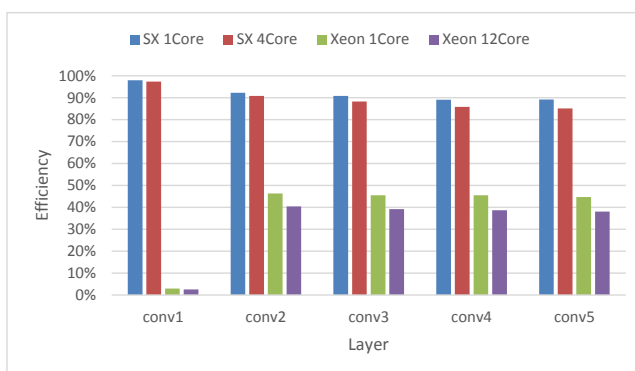


図 10 Intel Skim Caffe との比較
SX : SX-ACE, Xeon : Xeon E5-2680 v3

参考文献

- [1] Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, *Proceedings of the 25th International Conference on Neural Information Processing Systems, NIPS'12*, pp. 1097–1105 (2012).
- [2] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *ICLR 2015*, (online), available from <http://arxiv.org/abs/1409.1556v6> (2015).
- [3] Han, S., Mao, H. and Dally, W. J.: Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding, *ICLR'16*, (online), available from <http://arxiv.org/abs/1510.00149v5> (2016).
- [4] Liu, B., Wang, M., Foroosh, H., Tappen, M. and Penksy, M.: Sparse Convolutional Neural Networks, *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 806–814 (2015).
- [5] Park, J., Li, S., Wen, W., Tang, P. T. P., Li, H., Chen, Y. and Dubey, P.: Faster CNNs with Direct Sparse Convolutions and Guided Pruning, *ICLR 2017*, (online), available from <https://openreview.net/forum?id=rJPcZ3ttxx> (2017).
- [6] IntelLabs: Caffe for Sparse Convolutional Neural Network, <https://github.com/IntelLabs/SkimCaffe>.
- [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.: Deep Compression on Alexnet, <http://songhan.github.io/Deep-Compression-AlexNet/>.