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
CPSY2018-107,DC2018-89（2019-03）

組込みプロセッサ用AI処理向けベクトルユニットの設計

井出　陽介† 　鈴木　宏海† 　森　　祐樹† 　山﨑　信行†

† 慶應義塾大学理工学部　〒212-0032 神奈川県川崎市幸区新川崎 7-1
E-mail: †{ide,suzuki,mori,yamasaki}@ny.ics.keio.ac.jp

あらまし　近年，AIアプリケーションが幅広い分野で用いられるようになり，その学習や認識に用いられるニューラルネットワーク (NN) の高速化手法が盛んに研究されている．High-Performance Computing(HPC) 向けの研究ではGPUやFPGA，特定NN特化のASICを用いた手法等が提案されている．しかしながら，これらの手法は電力消費や面積の観点から組込み向けに適用するのは難しい．一方で組込みプロセッサの中にはマルチメディア処理向けにベクトルユニットを備えているものがある．本研究では組込みプロセッサである Responsive Multithreaded Processor(RMTP) のベクトルユニットを拡張し，畳み込みニューラルネットワークで頻繁に実行される畳み込み演算の性能向上を図った．ベクトル演算は長いベクトルに対する性能向上の効果が大きい一方で，二次元状でサイズの小さいカーネルを用いる畳み込み演算に対しては非効率となってしまう．そこで，ロード時に一次元のベクトルになるようデータ整形を行うことによって畳み込み演算の効率向上を図った．さらに，ベクトルユニットに低精度な演算を用いたSIMD演算を導入し，複数のカーネルに対する畳み込み演算を並列に行うことで，計算精度とトレードオフに処理の高速化を行った．

キーワード　組込みプロセッサ ベクトルユニット 畳み込みニューラルネットワーク AI

Design of Vector Unit for AI Acceleration in Embedded Processor

Yosuke IDE†, Hiromi SUZUKI†, Yuki MORI†, and Nobuyuki YAMASAKI†

† Faculty of Science and Technology, Keio University
E-mail: †{ide,suzuki,mori,yamasaki}@ny.ics.keio.ac.jp

Abstract　In recent years, AI is applied in wide range of fields. Its learning and recognition are based on Neural Network (NN), which are actively studied. Although for High-Performance Computing (HPC), GPU, FPGA or ASIC specialized to certain NN is proposed, it is not easy to apply them to embedded applications because of power consumption and area constraints. On the other hand, some embedded processors adopt vector units for multimedia application. In this study, extended vector load function and lower precision SIMD operation are added to vector units to accelerate convolution, which is executed in Convolutional Neural Network.

Key words　Embedded Processor, Vector Unit, Convolutional Neural Network, AI

1. はじめに

近年，機械学習の分野に属する深層学習が目覚ましい発展をとげ，医療画像の診断 [15] や芸術作品の生成 [8] といったものから，スマートフォン上での顔認証など日用的なものまで，多岐にわたる用途へAIの応用が進められている．

畳み込みニューラルネットワーク (CNN, Convolutional Neural Network) は深層学習に用いられるニューラルネットワークモデルの一つで，畳み込み演算を繰り返し行い入力データから特徴量を抽出し識別を行う．このCNNの処理を高速化する手法として，GPU [11] やFPGA [16]，ニューラルネットワーク計算に特化したASIC [3], [4] を用いた手法等が提案されている．

リアルタイム性の求められる組込み用途では，遠隔地にあるサーバへアクセスする通信や処理による遅延の大きいクラウドコンピューティングを行うことが困難なため，組込みプロセッサ上での処理が求められる．しかしながら，GPUやFPGA，ASICを用いた手法はHigh Performace Computing(HPC) 向けであり，組込みシステムにおいて要求される省電力性やコスト，実装面積といった制約を満たすことができず，高速化の手法として用いることは困難である．組込みプロセッサの中にはマルチメディア処理のように高いDLP(Data Level Parallelism)

を有するアプリケーション向けにベクトルユニットを備えるものがある．ベクトルアーキテクチャは電力効率の高さや設計のシンプルさから，研究用途では Responsive Multithreaded Processor(RMTP) [13]，商用では ARM [1] のような組込みプロセッサに用いられている．本研究では，RMTP に搭載されたベクトルユニットの拡張により，畳み込み演算の高速化を行った．3 × 3 のような小さなカーネルを用いることが多い畳み込み演算に対し，ベクトルロードへの機能拡張により演算効率を向上させる手法と，低精度なデータ型を用いることで処理の並列度を向上させる手法を提案する．

2. 背　　景

2.1 RMTP

Responsive multithreaded processor(RMTP) [13] は，8way の優先付き Simultaneous Multithreading(SMT) を行うリアルタイム処理向け組込みプロセッサである．RMTP Unit のブロック図は図 1 に示したとおりである．本研究ではこのプロセッサのベクトルユニット及び Data Cache 内のベクトルユニット用メモリコントローラに対して拡張を行った．

RMTP にはマルチメディア処理のような，高い演算性能を必要とする処理をサポートするために，ベクトルユニットが実装されている．ベクトルユニットは図 1 のように Vector Floating Point(VFP) Unit と Vector Integer(VINT) Unit の 2 つで構成される．それぞれが独立した Reservation Station に接続され Out of Order 実行が可能となっている．図 2 は RMTP のベクトルユニットのブロック図を示す．2 中の Vector Control Unit には VINT と VFP それぞれに制御レジスタを保有し，ベクトル長やストライド，Reservation Station の無効化など，専用のコプロセッサ命令による設定を行うことが可能である [17].

ベクトルレジスタは VINT，VFP ともに 512 エントリ持ち，各エントリの幅は VINT が 32bit，VFP が 64bit となっている．ベクトル演算を行うのに先立ち，ベクトルレジスタを確保する必要がある．その際，使用するベクトル長に適したモードを選択することができる．確保するエントリが 128 の場合，使用可能なベクトルレジスタの長さや本数が制限されるものの，最大で 4 スレッドでのベクトル演算をサポートする．一方で，1 スレッドからのみのアクセスに限定されるが，最大 64 エントリのベクトルレジスタを 8 本使用可能とすることも可能である．

VINT では 32bit のレジスタを 16bit×2，8bit×4，4bit×8 で共有し，VFP では 64bit のレジスタを 32bit×2，16bit×4 で共有する SIMD 演算が可能である．図 7 は VFP の SIMD ユニットの構成示したものである．この演算方式は，RMTP において二次元ベクトル演算 [17] と呼ばれている．各レーンに VINT では 32bit，VFP では 64bit のデータが入力され，指定した SIMD モードにより演算を行う．

2.2 畳み込みニューラルネットワーク

畳み込みニューラルネットワーク (Convolutional Neural Network: CNN) は，多層ニューラルネットワークの一つである．CNN は学習により取得したパラメータを用いて畳み込み演算を行い特徴量を抽出する．この処理を繰り返し行うことで識別を行う．主な層として畳み込み層，プーリング層，全結合層がある．CNN は数層の畳み込み層とプーリング層を積み上げたトポロジになっており，state-of-the-art なニューラルネットワークでは 30 層以上の深いネットワークも存在する．代表的な CNN として，GoogLeNet [14] や AlexNet [6]，VGG [12]，ResNet [5] 等がある．例えば VGG は畳み込み層とプーリング層を繰り返し通し，出力層付近に全結合層が配置される．VGG11 では全体が 21 層で構成さている中で，8 層が畳み込み層で実装されている．各畳み込み層では 3 × 3 のカーネルを用いた畳み込み演算を大量に行う．他の CNN アーキテクチャでも同様に畳み込み層が深く，各層のカーネル数も数十から数百にも上るため，畳み込み演算の高速化が重要になる．そのため，本研究では主に畳み込み演算の高速化に焦点を当てた．

次に畳み込み演算に関して説明する．二次元データに対する畳み込み層での演算の動作を図 3 に表した．畳み込み層では，図 3 で示したように入力データに対して $k \times k$ の行列で表されるカーネル K を用いて畳み込みを行う．二次元の入力データ A に対する畳み込みは式 1 で表せるように，カーネルをスライドさせながら畳み込みを行うことで，入力データ全体から特徴

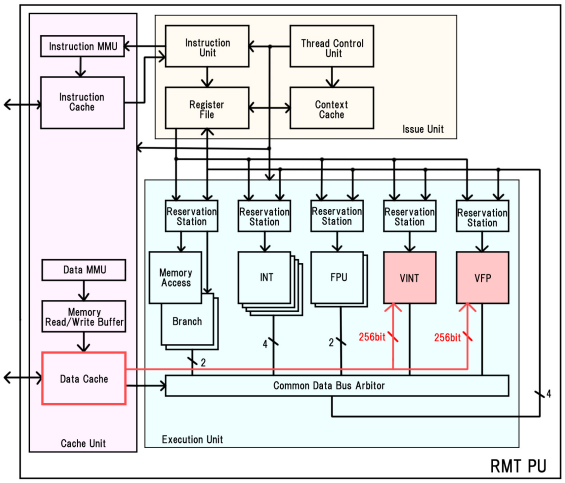

図 1　RMTP Unit のブロック図
Fig. 1　Block diagram of RMTP

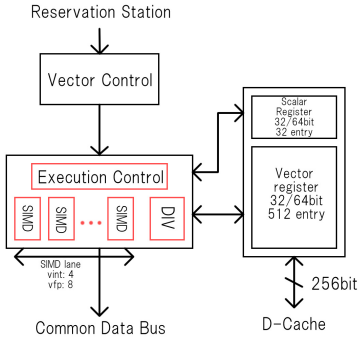

図 2　ベクトルユニットのブロック図
Fig. 2　Block diagram of vector unit

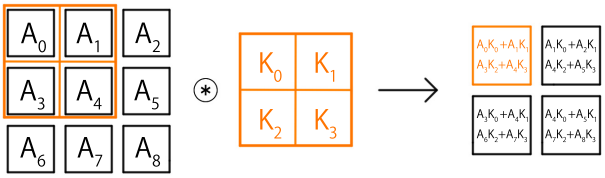

図 3　畳み込み演算
Fig. 3　Convolution

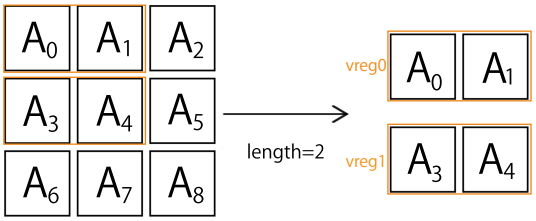

図 4　従来のベクトルロード
Fig. 4　Conventional vector load

量の抽出を行う．

$$B_{i,j} = \sum_{x,y=0}^{k} A_{i+x,j+y} K_{x,y} \tag{1}$$

3. 関 連 研 究

3.1 ニューラルネットワーク計算の高速化

High performance computing(HPC) 用途でのニューラルネットワークの高速化において，大部分のトランジスタを演算器へと使用し計算の並列度を飛躍的に向上させている．その反面，大量のデータを消費するため，広い帯域幅のメモリが必要となり，メモリアクセスが高性能化のボトルネックとなりやすい (memory wall)．そのため，メモリアクセスの低減や効率化，広帯域化といった手法が多く提案されている [3], [4], [11], [16]．その一例は DaDianNao [3] である．DaDianNao [3] ではコンフィギュレーション可能なパイプラインにより構成された Neural Function Unit(NFU) をアクセラレータとして用いる．NFU の直近に 4 バンクの eDRAM を 2MB 集積することで，メモリアクセスのレイテンシを低減し，メモリ帯域を広げる手法を提案している．

HPC 用途とは対照的に，組込みプロセッサでニューラルネットワーク計算を高速化する場合には，メモリアクセス以上に計算スループットがボトルネックと考えられる．その要因として複数の点が考えられる．まず，HPC に使用されるプロセッサの動作周波数が GHz オーダである一方で，組込みプロセッサでは電力消費を抑えるために，周波数が数十 ∼ 数百 MHz である点である．また，限られた面積のダイ上に SPI や I2C，PWM など制御用 IO を多数組み込む必要があり，演算器に費やすことのできるリソースには限界がある．

マルチメディア処理のように演算スループットの必要なアプリケーションにに対応するために，RMTP や Arm [1] の NEON などベクトル演算をサポートするものも存在する．CMSIS-NN [7] では NEON を搭載した Arm Cortex-M シリーズを用いた畳み込みニューラルネットワーク用のソフトウェアの設計を行っている．しかしながら，畳み込み演算を行う際にソフトウェアを用いたデータ整形により演算を最適化するため，メモリや実行時間のオーバヘッドが大きくなる．

3.2 低精度なデータ型

ニューラルネットワークにおいて，演算の精度を落とすことにより演算の並列度やメモリ密度の向上を行う手法は非常に有効である．そのため，低精度な演算を用いた高速化はニューラルネットワークにおいて頻繁に使用されている [3], [4], [9], [10]．例えば，XNOR-Net [10] は重みとアクティベーションを ±1 の二値で表現するが，ResNet-18 を用いた評価では Full-Precision のネットワークの 80% 以上の認識精度を維持した．一方で，CPU 上での実装では 32bit 浮動小数点演算に比べ 58 倍の高速化を実現した．このようにニューラルネットワークの識別精度が計算精度に受ける影響は小さい．

GPU や CPU では低精度演算として 16bit 浮動小数点や 4bit 整数をサポートしていることが多い．例えば，NVIDIA Turing [2] では，ディープラーニングによる推論の高速化を目的とした Tensor Core に 16bit 浮動小数点や 4bit が追加された．また，ARMv8 [1] では SIMD 拡張の命令セットである NEON において，16bit 浮動小数点のサポートを行うことにより，演算スループットの改善を図っている．

4. 提 案 手 法

4.1 設 計 方 針

ベクトル演算を用いて畳み込み演算を行う際のボトルネックとなるを解析するために予備評価を行った．CNN に用いられるカーネルのサイズは一般的に小さいため，カーネルサイズ $k = 3, 5, 7$ に絞った．また入力データサイズは 28×28 と MNIST に準拠したデータサイズを採用した．ベクトル演算を用いた畳み込み演算は Algorithm1 のように表される．ロードや演算を 1 行ごとに行うことから，カーネルサイズごとに処理を分けなくてはならない．更に 4 のように 2 次元の行列を k 回に分け長さ k のベクトルごとにロードを行い，k 個の演算命令を発行する必要がある．擬似コード中では省略したが，$k = 5$ の場合は 17 行，$k = 7$ の場合は 23 行のコードが必要となる．

Algorithm1 に従い，ベクトル演算を用い畳み込みを行う API をアセンブラにより記述し，スカラとベクトルについて倍精度浮動小数点を用いた演算を行った．実行サイクル数を測定した結果は図 5 に示した．カーネルサイズが小さいほどベクトル演算のスカラ演算に対する相対性能が下がるという結果が得られた．これは処理を行うベクトル長が短いだけでなく，計算に必要な命令数が増加するのに伴い，ベクトルユニットの使用率が低いことが原因だと考えられる．従って，本研究では畳み込みに用いるベクトル長を伸張することにより，演算の効率を向上することを設計の方針とする．

4.2 ベクトルロード時のデータ整形

$k \times k$ のベクトル演算に対して畳み込み演算を行う際には，

Algorithm 1 ベクトル演算を用いた畳み込み演算

$VectorLength \leftarrow k$
if $k = 3$ **then**
  $v0 \leftarrow \{K[0], K[1], K[2]\}$ // load vector to vreg0
  $v1 \leftarrow \{K[3], K[4], K[5]\}$
  $v2 \leftarrow \{K[6], K[7], K[8]\}$
  $v3 \leftarrow \{A[adr - w_{in} - 1], A[adr - w_{in}], A[adr - w_{in} + 1]\}$
  $v4 \leftarrow \{A[adr - 1], A[adr], A[adr + 1]\}$
  $v5 \leftarrow \{A[adr + w_{in} - 1], A[adr + w_{in}], A[adr + w_{in} + 1]\}$
  $v0 \leftarrow v0 \times v3$ //Vecotr Multiply
  $v0 \leftarrow v0 + v1 \times v4$ //Vector Multiply and Add
  $v0 \leftarrow v0 + v2 \times v5$
  $s0 \leftarrow \sum_{i=0}^{k-1} v0[i]$ //Vector Accumulate
  $out \leftarrow s0$
else if $k = 5$ **then**
  ...
else if $k = 7$ **then**
  ...
end if

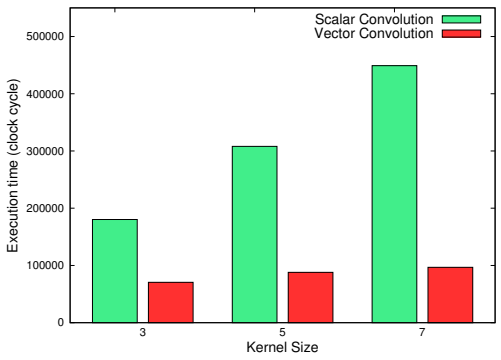

図 5　畳み込み演算実行サイクル数
Fig. 5　Execution cycle of convolution

図 6 のように入力データを一次元に整形してベクトルレジスタへのロードを行うようにハードウェアの設計を行った．ベクトルユニット内部の制御レジスタにカーネルサイズ $ksize = k$，ベクトル長 $length = k^2$，ストライド $stride = w_{in}$(入力データの幅) を設定し，既存のベクトルロード命令を発行するのみで使用可能とした．

図 6 では，A_0 をベクトルロードのソースアドレスと指定すると，A_0 から $ksize(= 2)$ 要素ロードする．次にインデックスに $stride(= 3)$ を加算し A_3 から $ksize(= 2)$ 要素ロードを行う．設定したベクトル長 $length(= 4)$ に到達するとロードを完了する．行方向にロードする要素数はカウンタにより管理し，$ksize$ の値により任意に変更可能である．この手法により Algorithm1 のようなカーネルサイズごとの処理の分岐を不要とし，畳込み演算に必要な命令数を大幅に低減を図った．ベクトルロード時の一次元化を用いた場合の演算は Algorithm2 に示される．Algorithm2 の 6 行目で示した VMAC(Vector Multiply and Accumulate) 命令のみで畳み込みを行うことができるため，畳み込み演算の効率を大幅に向上させることが可能なった．

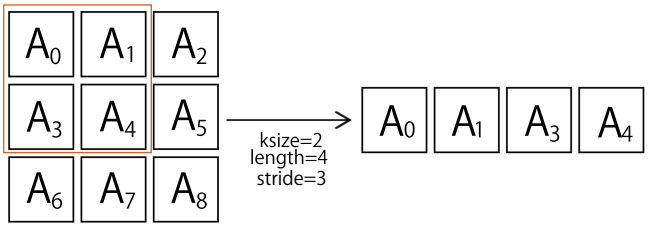

図 6　畳み込み演算の 1 次元化
Fig. 6　Shaping on vector load

この提案手法は畳み込み演算の最適化に頻繁に用いられる im2col と同様のデータ展開を行う．im2col では畳み込みを行う前にメモリ上に展開したデータをコピーしていくため，実行時間のオーバヘッドが大きいだけでなく，大量のメモリを使用してしまう欠点がある．例えば倍精度浮動小数点の 28×28 入力データをカーネルサイズ 3×3，ストライド 1 と仮定し，im2col でデータを展開すると 10kB 以上のメモリが余分に必要となる．その影響でキャッシュサイズの小さい組込みプロセッサではデータの追い出しが頻繁に起こる可能性がある．一方で，本提案手法では使用するメモリは一切増加しないという利点がある．

また，この提案手法は Gather-Scatter の変種である．しかしながら，一般的な Gather-Scatter ではインデックスベクトルによりアクセスするアドレスを選択を行う必要があり，余分なメモリアクセスが発生する．そのため，Sparse でサイズの大きい行列の演算において Gather-Scatter によるメモリアクセスは有効だが，3×3 のような小規模なカーネルに対してはオーバヘッドが支配的になることが想定される．対照的に，本手法では制御レジスタからの信号及び先頭アドレスのみでロード可能なため，畳み込み演算に対して有効だと考えられる．

Algorithm 2 一次元化を行った畳み込み演算

$length \leftarrow k^2$
$ksize \leftarrow k$
$stride \leftarrow w_{in}$
$v0 \leftarrow \{K[0], K[1], ..., K[k^2 - 1]\}$
$v1 \leftarrow k \times k$ elements around $A[adr]$
$s0 \leftarrow \sum_{i=0}^{k^2-1} v0[i] \times v1[i]$ //Vector Multiply and Accumulate
$out \leftarrow s0$

4.3　低精度演算

近年多く用いられる CNN では，各畳み込み層には複数の入出力チャネルを持ち，それに応じた複数のカーネル (VGG では 64 から 512 個) による畳み込み演算を処理する必要がある．そこで 16bit 浮動小数点 (表 1) 及び 4bit 整数 (表 2) を導入し，精度と引き換えに並列度の向上を図った．実装後の二次元ベクトル演算器のうち，VFP を示したものが図 7 である．64bit のレジスタを 16bit のデータ 4 要素で共有し，毎サイクル 16 要素の演算を実現する．64bit の浮動小数点レジスタを 16bit×4 で共有し，一つ目の提案したベクトルロードの手法と組み合わ

表 1　16bit 浮動小数点型フォーマット
Table 1　Format of 16bit floating point

ビット長 (bit)	指数部 (bit)	仮数部 (bit)	表現範囲 (絶対値)
16	5	10	$5.96046e^{-8}$ ～65504

表 2　4bit 整数型の表現範囲
Table 2　Range of 4bit integer

ビット長 (bit)	表現範囲	
	signed	unsigned
4	-8～7	0～15

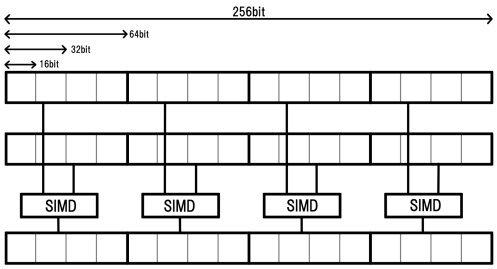

図 7　16bit 浮動小数点をサポートした二次元ベクトル演算
Fig. 7　2D vector operation with 16bit floating point

せることにより，同時に 4 つのカーネルを処理することを可能とした．

5. 評　　価

提案手法による畳み込み演算の性能向上および面積オーバヘッドの評価を行った．実装は RMT Processor に対して Verilog HDL により行った．また，性能評価は Cadence 社の NC-Verilog を用いたシミュレーションにより，畳み込み演算の実行サイクル数を計測した．面積評価には TSMC 130nm のスタンダードセルライブラリを使用し遅延制約は 7ns で，Synopsys 社の Design Compiler による論理合成を行った．

5.1　性 能 評 価

まず 1 チャネルに対しての，倍精度スカラ，従来の倍精度ベクトル演算，提案手法のベクトルロード用いた倍精度ベクトル演算，提案手法に加え 4 回ループアンローリングを行った倍精度ベクトル演算，半精度ベクトル演算の実行サイクル数を比較した．この結果は図 8 のようになった．提案手法のベクトルロードを用いた場合，従来のベクトル演算の 1.49 から 1.77 倍，スカラ演算の 4.52 から 6.93 倍の高速化を実現できた．さらにループアンローリングを用いると，従来のベクトル演算の 1.78 から 2.34 倍，スカラ演算の 5.975 から 8.28 倍の性能向上が得られた．ソフトウェアパイプライニングなど，他のソフトウェア最適化手法と組み合わせることにより，さらなる性能向上が期待できる．一方で半精度浮動小数点での演算では，データアドレスのミスアラインを回避するためのデータ整形のオーバヘッドが大きく，性能向上は困難だった．

次に 4 チャネルに対しての畳み込み演算の実行サイクル数を調べた．半精度ベクトル演算に関しては，ループアンローリングの有無で 2 パターンに関して評価を行った．測定結果は図

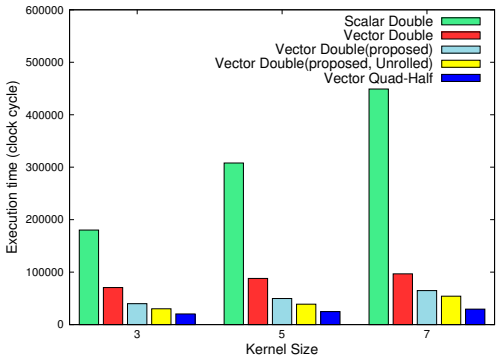

図 8　畳み込み演算の実行サイクル数
Fig. 8　Execution time of convolution

9 のようになった．半精度浮動小数点を用いることで，1 サイクルあたりに処理可能なチャネル数が倍精度ベクトル演算の 4 倍となるため，理想的には倍精度ベクトル演算の 4 倍の高速化が見込まれるはずである．実際の測定では，倍精度スカラ演算の 20.5 から 22.8 倍の性能向上を得られたが，倍精度ベクトル演算の 2.47 から 2.97 倍程度の性能向上にとどまった．その原因として，複数のカーネルのパックや入力データの整形のオーバヘッドが大きかったことが挙げられる．半精度ベクトル演算のトータル実行時間のうち 25.8% から 31.8% をデータの整形に要した．この点に関して，ハードウェア機構によるオーバヘッドの軽減を行うことも可能だが，ソフトウェア設計による対策を行う方が好ましいと考えられる．組込み AI 向けに使用する場合，使用用途が明確な可能性が高く静的な学習で事足りることが考えられる．従って認識に使用するパラメータは静的な学習により得たもので，事前に 16bit 浮動小数点のパラメータをパックしておくことで，カーネルのパックを動的に行う必要はなくなる．また，同じ層での入力データは同一のまま大量のカーネル (例えば VGG では 64 から 512 個) を用いた畳み込み処理が行われることを考慮すると，カーネル数が増えるほどデータ整形のオーバヘッドは相対的に低減される．またデータのパックが終わると，元のデータを格納していた領域は使用しないため再利用可能であり，浪費するメモリも比較的少量ですむ．これらの理由から，ソフトウェア設計次第で理想である 4 倍に近い性能を得ることができると考えられる．

5.2　面 積 評 価

今回の提案手法であるベクトルロードの機能拡張，二次元ベクトル演算器の SIMD モード拡張を実装することによる面積オーバヘッドを調べた．メモリコントローラはステートマシンの変更のみであったため，面積オーバヘッドは小さい．16bit 浮動小数点の演算器は拡張により，VFP Execution Unit の面積増加率は 39.6% と比較的高い割合となった．搭載するベクトルユニット数を増加させる場合には，演算機の使い回しをするなど SIMD ユニットの実装を再考する必要がある．

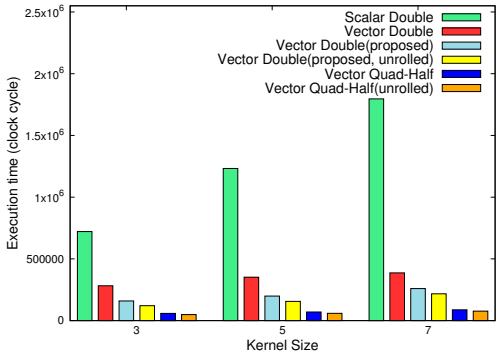

図 9　4 チャネルでの畳み込み演算

Fig. 9　Execution time of convolution of 4 channels

表 3　Vector Unit 拡張による面積オーバヘッドの評価

Table 3　Evaluation of area overhead

モジュール名	セルエリア μm^2			面積増加率
	original	proposed	diff	%
decoder	74,797	83,973	9,176	12.3
Memory Control	95,548	101,791	6,243	6.53
VINT Control Unit	88,767	88,716	-51	-0.575
VINT Execution Unit	1,057,965	1,059,386	1,421	0.134
VFP Control Unit	86,943	86,977	34	0.391
VFP Execution Unit	2,255,995	3,149,761	893,766	39.6
Total	3,660,015	4,470,604	910,589	24.9

6. ま と め

本研究ではベクトルロードの機能拡張により，カーネルサイズが小さいことの多い畳み込み演算において，ベクトル演算の効率を向上させる手法を提案した．その結果，ベクトルロード時のデータ整形により従来手法の最大 1.77 倍の高速化を行った．さらに，16bit 浮動小数点や 4bit 整数を導入し複数チャネルのカーネルを並列に処理することにより，さらなる Data Level Parallelism(DLP) の活用をした．半精度浮動小数点による演算では倍精度演算での 2.47 から 2.97 倍ほどの性能向上が得られた．ただし，ソフトウェア設計により 4 倍近い性能の向上を達成できると考えられる．

本論文での対象外ではあったが全結合層や ReLU などの計算においても，ベクトル演算を用いた高速化が期待できる．一方で，CNN 全体を動作させた場合，ベクトル化できない部分や，大量のパラメータを用いることによるキャッシュミスやポリューションが全体性能に与える影響は不透明である．そのため，CNN にデータを入力し，認識結果を得る一連の過程に対しての性能の評価と向上を今後の課題とする．

謝辞　本研究は国立研究開発法人 科学技術振興機構 (JST) の助成を受けたものである

文　献

[1] Arm architecture reference manual: Armv8, for armv8-a architecture profile, 2017. `https://static.docs.arm.com/ddi0487/da/DDI0487D_a_armv8_arm.pdf`.

[2] Nvidia turing gpu architecture, 2018. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization /technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, Dec 2014.

[4] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 27–39, June 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, Vol. abs/1512.03385, , 2015.

[6] A. Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

[7] L. Lai, N. Suda, and V. Chandra. CMSIS-NN: efficient neural network kernels for arm cortex-m cpus. *CoRR*, Vol. abs/1801.06601, , 2018.

[8] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2479–2486, June 2016.

[9] F. Li and B. Liu. Ternary weight networks. *CoRR*, Vol. abs/1605.04711, , 2016.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, Vol. abs/1603.05279, , 2016.

[11] S. T. H. Rizvi, G. Cabodi, and G. Francini. Gpu-only unified convmm layer for neural classifiers. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 0539–0543, April 2017.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, Vol. abs/1409.1556, , 2014.

[13] K. Suito, R. Ueda, K. Fujii, T. Kogo, H. Matsutani, and N. Yamasaki. The Dependable Responsive Multithreaded Processor for Distributed Real-Time Systems. *IEEE Micro*, Vol. 32, No. 6, pp. 52–61, December 2012.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[15] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, Vol. 35, No. 5, pp. 1299–1312, May 2016.

[16] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pp. 161–170, New York, NY, USA, 2015. ACM.

[17] 松井司, 大槻周平, 山崎信行. Rmt processor の vector unit のスループット向上手法. 組込み技術とネットワークに関するワークショップ ETNET2016, pp. 127–132, 2016.