

ROMへのアクセスレイテンシが大きいマイコンを対象とした畳み込みニューラルネットワークの最適化

下平 健太^{1,a)} 本田 晋也² 高田 広章¹ 枝廣 正人¹

概要：近年、組込みシステムにおいても深層学習における推論の利用が求められており、小規模組込みシステムではマイコン使用を前提としたアルゴリズムが必要となっている。一方で、性能向上の流れの中でチップの製造プロセスの微細化が進み、ROM(FLASH メモリ)を外付けにする構成のマイコンの利用が増えてきている。しかしながら、現在の DNN 推論プログラムは従来の FLASH メモリを内蔵しているマイコン向けに作られており、アクセスレイテンシが大きい外付け FLASH のみを持つマイコンにおいて課題がある。本研究では、DNN の一種である畳み込みニューラルネットワークにて用いられる畳み込み関数に対し、FLASH メモリを外付けするマイコンにおいて効率的なアルゴリズムを複数提案する。評価では、アルゴリズムの効率性を示すとともに、それらのアルゴリズムを使用条件に応じて使い分けることで、実行効率の向上させた。

1. はじめに

近年、多層ニューラルネットワーク (以下 DNN) に代表される深層学習は、画像認識 [1] や音声認識 [2] および自然言語処理 [3] などの幅広い分野において使われ注目を集めており、機器制御等の小規模な組込みシステムにおいても機器の故障検知などで深層学習の推論の利用が求められている。

深層学習を実装するうえで用いられるフレームワークや開発環境は、現状では汎用 PC 向けに用意されていることが多く、学習と推論の両方に GPU や、推論に専用 HW や FPGA を利用することが一般的である。しかし、小規模な組込みシステムでは消費電力やメモリ使用量などのリソース制約があるため、多くのリソースが必要なこれらの演算処理装置は使用できず、小規模組込みシステム向けの CPU である 1 チップ・マイクロコンピュータ (以下マイコン) を使用することになる。マイコンとは、メモリや、CPU、入出力回路などの周辺回路などのコンピュータに必要な機能を 1 チップに集積したものである。

マイコン上での深層学習を行う際の留意点として、マイコンのハードウェア及びソフトウェアスタックによって提

供される限られたリソースを最大限に活用するために、モデルや環境を積極的に最適化する必要がある [4]。

これまでの一般的なマイコンは、FLASH メモリや SRAM が内蔵されており、キャッシュはないか小容量 (数 KB) という特徴を持つ。また、FPU や SIMD 演算器を持つ場合もある。本論文ではこのような構成のマイコンを MCU と呼ぶ。

一方で、組込みシステムに求められる性能に対応するため、クロック周波数を向上させた、クロスオーバープロセッサ (COP) と呼ばれる構成のマイコンが登場し、利用が増えてきている。COP の特徴としては、500MHz を超える周波数での動作が可能で数百 MB 程度の FLASH メモリと SDRAM が外付けされ、数十 KB 程度のキャッシュや数百 KB 程度の SRAM を内蔵している。一方、FLASH メモリや SDRAM が外付けであるため、それらへのアクセスレイテンシが大きいという特徴がある。

現在、DNN の推論で用いられるプログラムやその他実装アルゴリズムは、高速アクセス可能な内蔵 FLASH メモリを持つ MCU を前提に設計されている。そのため、COP においては、適切な実装アルゴリズムでない場合がある。

本研究では、プロセッサコアに ARM 社の Cortex-M7[5] を搭載した COP 向けに、DNN の一種である畳み込みニューラルネットワークにおける畳み込み関数の効率的な実装アルゴリズムを複数提案する。

また、提案した複数の実装アルゴリズムの比較を行い、

¹ 名古屋大学
Nagoya University

² 南山大学
Nanzan University

^{a)} shimohira@ertl.jp

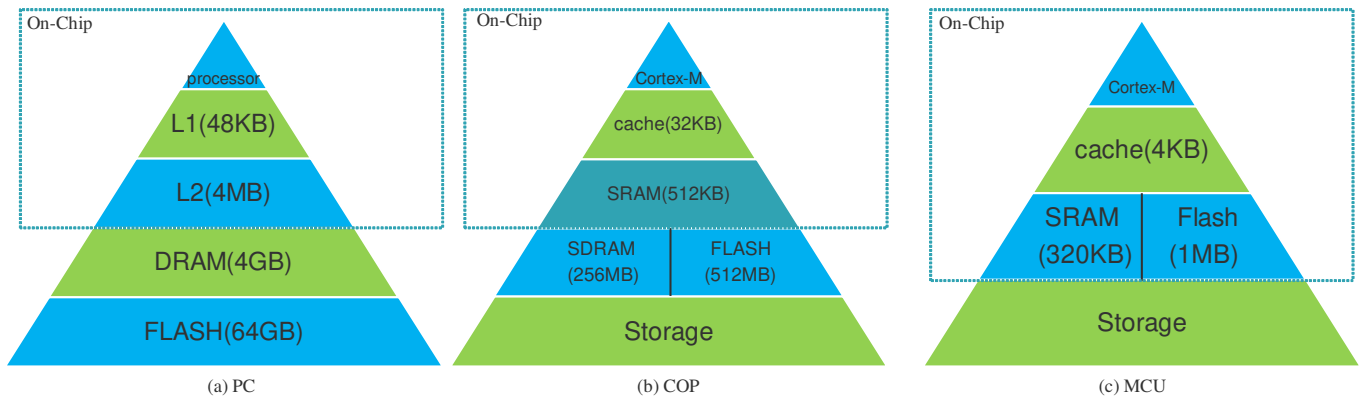


図 1 PC/COP/MCU のメモリ階層

表 1 MCU と COP のスペック例

名称	ST マイコン	NXP マイコン
マイコンの構成	MCU	COP
型番	STM32F746ZG	EVK-MIMXRT1060
プロセッサコア	Cortex-M7	Cortex-M7
最大クロック	216MHz	600MHz
キャッシュ	4KBytes	32KBytes
内蔵 FLASH メモリ	1MBytes	-
外部 FLASH メモリ	-	512MBytes
SRAM	320KBytes	512KBytes
SDRAM	なし	256MBytes

使用状況に応じた実装アルゴリズムの使い分けについても検討し考察を行う。

2. 小規模組込みシステム

本章では、小規模組込みシステムの特徴について説明し、特にその中で使われるマイコンの構成およびマイコン向けニューラルネットワーク・ライブラリである CMSIS-NN について説明する。

本論文において、組込みシステムは大規模組込みシステムと小規模組込みシステムの 2 つに分類する。

大規模組込みシステムとは、携帯電話や家電製品といった複数のコンピュータやオペレーティングシステムを用いた多機能のソフトウェアを持つシステムのことを指す。それに対して小規模組込みシステムとは、マイコンを用いて、故障検知や機器制御等に導入されている少機能のソフトウェアを持つシステムのことを指す [6]。

小規模組込みシステムにおいてはリソース制約が大きなネックとなっており、低消費電力、メモリ使用量の削減などが求められる。また、リアルタイム性も求められることが多く [7]、深層学習を実装する際にも例外ではなく、入力に対し素早く認識を行う必要がある [8]。

2.1 クロスオーバープロセッサの構成

本論文において、マイコンの構成を一般的な構成 (MCU) とクロスオーバープロセッサ (COP) の 2 種類に分類する。それぞれの構成を PC との比較を含め、図 1 に示す。また、それぞれの構成の例を表 1 に示す。

図 1(c) に MCU の一般的なメモリ階層を示す。FLASH メモリと SRAM がチップに内蔵されており、SDRAM がついている場合に外付けの構成となっている。

図 1(b) に示す COP は SRAM が内蔵で、FLASH メモリと SDRAM が外付けという構成になっている。COP は位置づけとしては、図 1 に示すように、従来の MCU と PC の中間にあたり、MCU より性能の高いマイコンとして組み込みシステムにおいて用いられる。

具体的な例として、既存の MCU の最大周波数は 200MHz 程度であったが、NXP 社の COP である「EVK-MIMXRT1060」は、CPU に Cortex-M7 を搭載し、600MHz で動作する。ただし、FLASH メモリが外付けであるため、FLASH メモリに置かれたデータを CPU に読み込む速度が MCU と比べて遅いという特徴がある。そのためキャッシュや、内蔵 SRAM を利用した実装をすることによって FLASH メモリから直接データを読み取る処理を少なくする必要がある。

2.2 Cortex-M

本論文では、マイクロプロセッサコアとして ARM 社の Cortex-M7 を対象とする。Cortex-M シリーズはマイコンや FPGA 向けに設計されたマイクロプロセッサコアである。様々な電子機器デバイスに対して、コストと消費電力を重視し、アプリケーションに最適化した組み込みプロセッサとなっている。

Cortex-M シリーズには、パイプラインの段数等が異なる複数の実装が存在する。ハイエンド向けの Cortex-M4 や Cortex-M7 には 8bit、もしくは 16bit の SIMD 演算による処理の高速化が可能である。

2.3 CMSIS-NN

CMSIS-NN は、Cortex-M プロセッサ上でのニューラルネットワークの性能を最適化し、メモリフットプリントを最小化するために開発されたソフトウェアライブラリである [9] [10].

パラメータは量子化された整数型の 8bit または 16bit に対応している。CMSIS-NN のライブラリは NNfunctions と NNSupportFunctions の 2 つのグループに分けられている。NNfunctions には畳み込み関数やプーリング関数といった実際の推論の処理を行う関数が含まれており、NNSupportFunctions には NNfunctions 内の関数で使われるデータ変換や活性化関数のテーブルなどの関数が含まれている。これらの関数をニューラルネットワークの実装部分で呼び出すことで、推論を行う。

CMSIS-NN の特徴としては、整数型の SIMD 命令により、畳み込み処理やプーリング処理の高速化を実現していることが挙げられる。その際に用いる SIMD 命令は 16bit 整数型命令を使用する。また、畳み込み関数においては、SIMD 命令を用いないベースライン実装も用意されている。

CMSIS-NN はオープンソースであり、各種マイコン向け DNN フレームワークにおいても使用されている。

3. 予備実験

本研究を進めるにあたって、現存の COP の上で DNN のプログラムを動かした際の性能を調査し、問題点を明らかにする。また、COP のメモリ性能についても評価を行う。

評価では表 1 で示した 2 つのマイコン、ST 社の一般的な MCU である STM32F746ZG (以下 ST マイコン) [11] と、NXP 社の COP である EVK-MIMXRT1060 (以下 NXP マイコン) [12] を用いる。

3.1 CMSIS-NN 畳み込み関数による評価

3.1.1 計測内容

2 種類のマイコン上で、CMSIS-NN の 2 種類の畳み込み実装アルゴリズムを、扱う重みデータ量を変更して実行し、それぞれの実行時間を計測する。なお、入力及び重みは 8bit であり、重みは初期化時に FLASH メモリに配置している。

2 種類の実装アルゴリズムの詳細を以下に示す。

- im2col(FLASH) :
Cortex-M ベースのマイコン向けに最適化された畳み込み実装アルゴリズムである。画像データを変換し畳み込み関数を行列演算により実現できるようにする im2col 関数を入力データと重みデータに適用し、その後の行列演算を SIMD を用いた積和命令によって高速に処理する。
- ベースライン実装 :
SIMD 等の高速な命令は用いず、多重ループ文を用い、

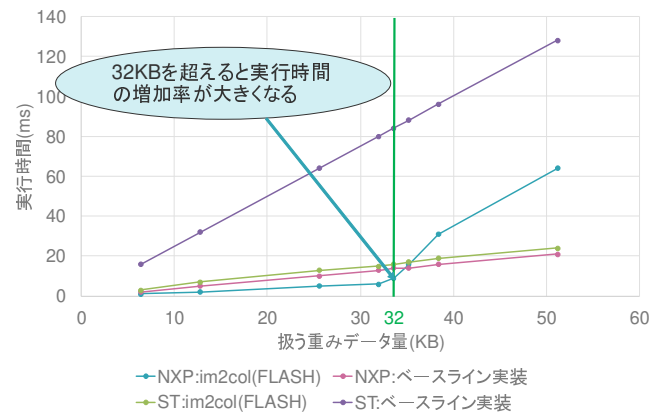


図 2 CMSIS-NN 畳み込み関数の計測結果

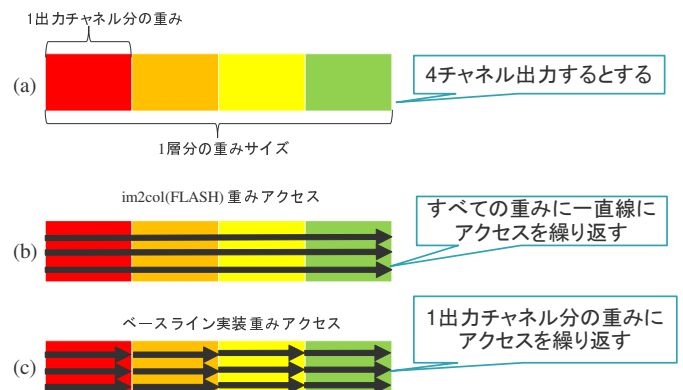


図 3 CMSIS-NN 実装アルゴリズム図

通常命令で積和命令を繰り返す。

3.1.2 計測結果

計測結果を図 2 に示す。ST マイコンについては、im2col(FLASH)、ベースライン実装の両方の結果において重みデータ量が増えると実行時間が線形に増加していることがわかる。また、im2col(FLASH) のほうがベースライン実装よりも高速であることもわかる。

次に、NXP マイコン上での結果では、ベースライン実装においては線形に実行時間が増加していることがわかる。しかし、im2col(FLASH) では、重みデータ量がキャッシュ容量である 32KB を超えたあたりから、実行時間の増加率が大きくなっていることがわかる。この結果は im2col(FLASH) とベースライン実装の重みアクセスの実装アルゴリズムの違いによるものである。

それぞれの実装アルゴリズムを図 3 を用いて比較し説明する。図 3 (a) は、1 層の畳み込み関数で用いる重み全体を示しており、4 チャンネル分の出力が結果として出るとする。

図 3 (b) は im2col(FLASH) の重みアクセスの実装アルゴリズムを示している、この実装アルゴリズムでは、すべての出力チャンネルの要素を順に計算していく。そのため、計算時にはすべての重みに対して一直線にアクセスを繰り返していく。全体の重みサイズがキャッシュ容量を超えている場合、キャッシュミスが発生し、FLASH メモリから

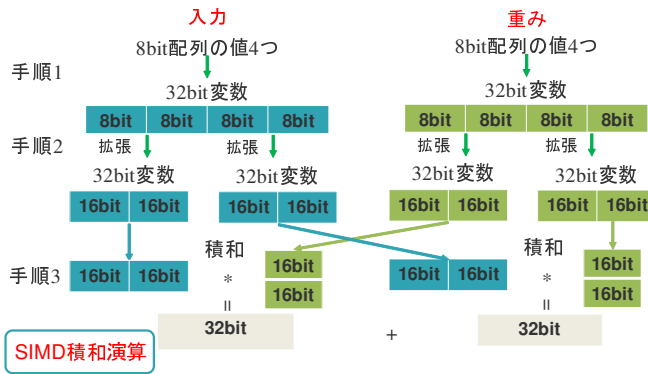


図4 ベースライン SIMD の概要

変更前	変更後
6000a780: ldr.w r6, [r2], #4	6000a780: ldr.w r4, [r0], #4
6000a784: mov.w r12, r6, ror #8	6000a784: sxtb16 lr, r4
6000a788: sxtb16 r12, r12	6000a784: mov.w r4, r4, ror #8
6000a78c: ldr.w r5, [r], #4	6000a78c: sxtb16 r4, r4
6000a790: mov.w r0, r5, ror #8	6000a790: ldr.w r3, [r7], #4
6000a794: sxtb16 r0, r0	6000a794: sxtb16 r12, r3
6000a798: sxtb16 r6, r6	mov.w r3, r3, ror #8
6000a79c: sxtb16 r5, r5	sxtb16 r3, r3
6000a7a0: smlad r1, r0, r1	smlad r2, r3, r4, r2
6000a7a4: smlad r1, r5, r6, r1	smlad r2, r12, lr, r2

図5 記述変更前後のアセンブリ記述

の読み込みが発生することになる。

一方、図3(c)のベースライン実装の重みアクセスの実装アルゴリズムは、1つの出力チャンネルで用いる重みデータに繰り返しアクセスし、1チャンネルを完成させ、その後次の出力チャンネルで用いる重みデータにアクセスするという処理を繰り返していく。そのため1出力チャンネルで用いる重みがキャッシュ容量以下であれば、キャッシュにヒットし続けるため、FLASHメモリへのアクセスが発生しない。

この違いから、STマイコンは、NXPマイコンと比べキャッシュ容量は小さいが、FLASHメモリが内蔵でありキャッシュミスペナルティが小さいため、重みのサイズがキャッシュサイズより大きい場合であっても、im2col(FLASH)の速度低下は発生しないと考えられる。

一方、NXPマイコンはCOPであり、im2col(FLASH)では、キャッシュ容量を超える重みを用いてしまうと低速であるFLASHメモリからの読み取り回数が増えてしまうため、実行時間が増加したと考えられる。

3.2 COPのメモリ読み込み性能評価

3.2.1 計測内容

前節より、NXPマイコンはCOPの特性上、FLASHメモリからの読み取り回数が多い実装の場合、レイテンシが大きいことから性能が低下してしまうと予想される。このことを確認するためNXPマイコンのメモリ読み込み性能を計測する。計測項目は以下の4項目である。

- 外部FLASHメモリからの読み取りレイテンシ
- キャッシュからの読み取りレイテンシ
- SDRAMからの読み取りレイテンシ

表2 NXPマイコン：500byteのデータ読み込み速度の計測結果

メモリ	計測結果 (cycles)
外部FLASHメモリ	2,523cycles
キャッシュ	390cycles
SDRAM	2,713cycles
SRAM	668cycles

• SRAMからの読み取りレイテンシ

時間計測は、Cortex-Mのデバッグ機能であるData Watchpoint and Trace Unit(DWT)を用いてクロック精度で計測する。

計測用のプログラムは、まずキャッシュをクリアした後に、計測対象のメモリからキャッシュサイズより大きい500byteデータを読み込み、その後、同じメモリ領域から再度データを読み込みその時間を計測する。このフローを1000回行い、平均値をメモリ性能計測結果とする。

3.2.2 計測結果

計測結果を表2に示す。結果から、もっとも性能が高いのはキャッシュであることがわかる。FLASHメモリと比べると、7倍近い性能差があることがわかる。よって、im2col(FLASH)の実行結果が重みデータ量が大きくなることでFLASHメモリからの読み取り回数が増え、低速な結果であったことが裏付けられる。また、SRAMの性能もFLASHメモリの3倍以上高い。

この実験から、COPであるNXPマイコンは、キャッシュやSRAMを効率的に使用する実装でない場合、FLASHメモリのレイテンシによって全体のスループットが低下することが考えられる。

4. CMSIS-NNのCOP向け改良

予備実験から、CMSIS-NNの畳み込み関数は、重みのサイズがキャッシュサイズ以上となるとCOPで性能が出ないことがわかった。そのため、本研究ではCOP上での高効率実行を目的とし、CMSIS-NNの畳み込み関数の改良を行い、高性能な実装アルゴリズム手法を提案する。

本研究では、三種類の実装アルゴリズムを提案する。

- ベースラインSIMD：
キャッシュ効率の良いベースライン実装にSIMD演算を適用した実装アルゴリズム
- im2col(SDRAM):
im2col(FLASH)をSDRAMを利用して最適化した実装アルゴリズム
- im2col(SRAM):
im2col(FLASH)をSRAMを利用して最適化した実装アルゴリズム

4.1 ベースラインSIMD

本節では、キャッシュ効率が良く性能の高い実装アルゴ

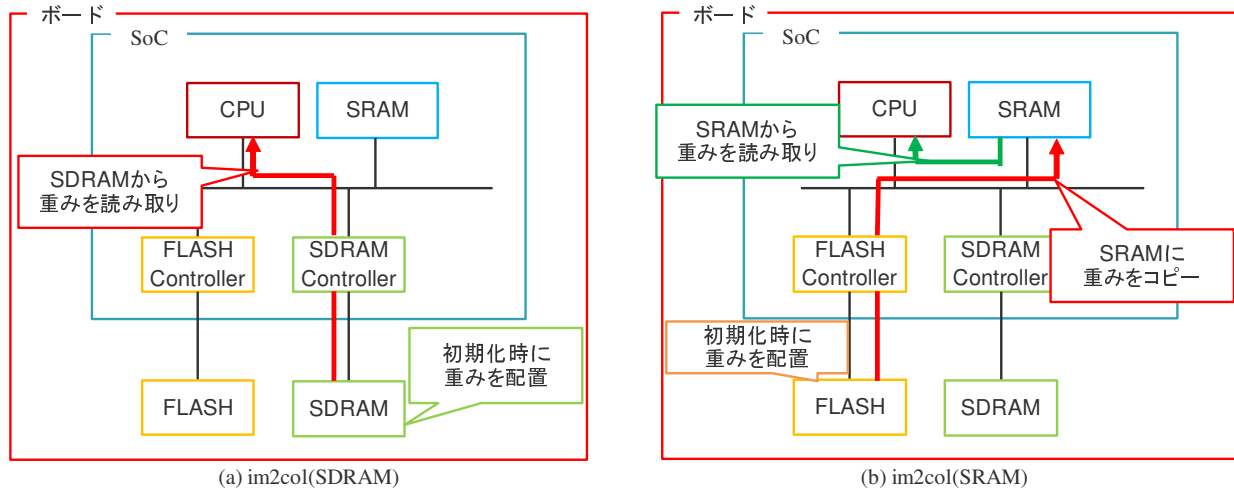


図 6 im2col(SDRAM/SDRAM) の概要

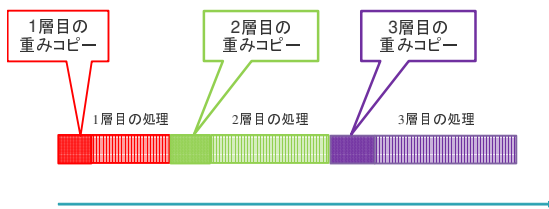


図 7 im2col(SRAM) の流れ

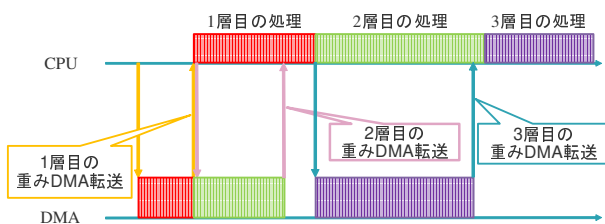


図 8 DMA による効率化手法

リズムである，ベースライン SIMD を提案する．

予備実験から，COP 上でのキャッシュ性能は高いことがわかっている．ベースライン SIMD はキャッシュ効率の良いベースライン実装の演算部分に変更を加え，積和演算時に SIMD 演算を適用するようにした手法である．

図 4 にアルゴリズムの概要を示す．SIMD 演算の処理は大きく三つの処理に分けられる．最初に，入力と重みは共に量子化された整数型 8bit の配列に格納されている．手順 1 として，配列の中から 4 つの値を取り出し，それを 32bit 配列の中にそれぞれ格納する．次に手順 2 として，格納された 8bit の値を 16bit に拡張し，16bit の値 4 つを 2 つの 32bit 配列の中にそれぞれ格納する．そして手順 3 として，入力と重みの 32 ビット配列をそれぞれ 1 つずつ呼び出し，16bit の整数型 SIMD 積和演算を実行し，結果の値を足し合わせていく．これらの手順を繰り返すことによって得られた最終的な出力を 8bit に変換しなおし，最終的な出力結果とする．

また，ベースライン SIMD のコンパイル結果のアセンブリ記述を確認して，C 言語記述を一部変更することによってパイプライン処理をより効率化したベースライン SIMD (効率化) も提案手法とする．記述変更前と変更後のアセンブリ記述を図 5 に示す．

アセンブリ記述を比較すると，変更後の処理ではロード命令である `ldr` 命令間の命令数が 1 つ増えていることがわかる．Cortex-M7 はインオーダー実行であるため，高速化するのではないかと考えられる．

4.2 im2col(SDRAM)

COP には，大容量の SDRAM が付属していることが多く，NXP マイコンにおいても 256MB の SDRAM がある．im2col(SDRAM) では，図 6(a) に示すように，初期化時に SDRAM に重みを配置しておき，im2col(FLASH) の呼び出し時に，SDRAM から重みにアクセスするようにする．SDRAM は大容量であるため，重みを常に配置しても問題ないと考えられる．

利点として，初期化時に SDRAM に重みを配置しているため，推論実行時に FLASH メモリからのコピーが不要であることが挙げられる．

4.3 im2col(SRAM)

COP には，SDRAM よりも小容量の SRAM が内蔵されている場合がある．NXP マイコンでは，512KB の SRAM が搭載されている．この SRAM を用いた効率化手法を提案する．

im2col(SRAM) では，図 6(b) に示すように 1 レイヤーの畳み込み関数の実行時に，FLASH メモリに置かれている重みを SRAM にコピーし，im2col(FLASH) の実行時に SRAM から重みを読み取る．これを，図 7 に示すように，畳み込み層の計算を行う毎にその層で用いる重みを SRAM

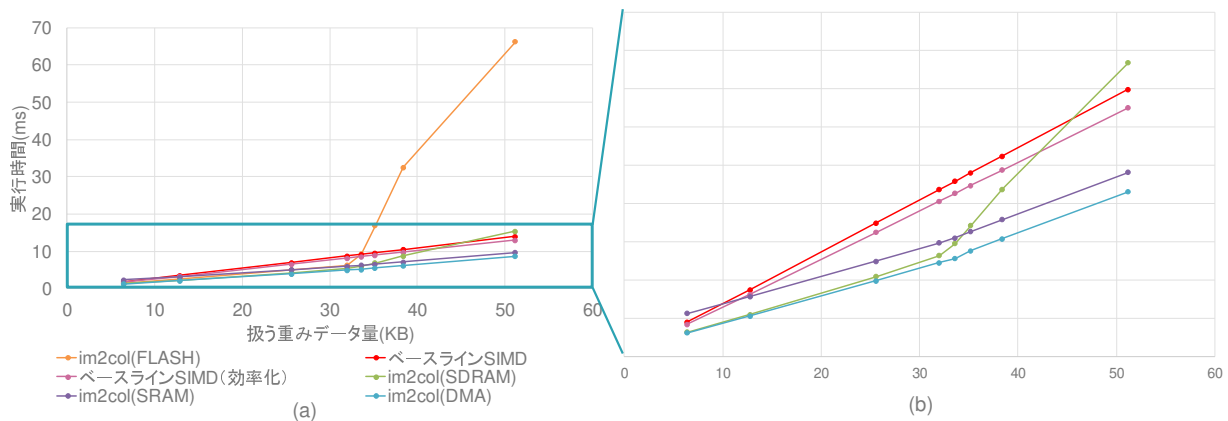


図 9 提案手法の実行結果

にコピーしてから、畳み込み処理を実施する。

この実装の欠点として、実行時に重みを FLASH メモリから SRAM にコピーする時間が必要となることが挙げられる。この欠点を補う実装として、Direct Memory Access(以下 DMA)を用いた効率化が考えられる。DMA を用いた効率化手法の実装アルゴリズムを図 8 に示す。

まず、1 層目で用いる重みを SRAM に DMA 転送する。そして、1 層目の畳み込み関数を実行している間に、DMA を用いて 2 層目の重みの SRAM への転送を実行する。この一連の処理を 3 層目以降もを繰り返すことで、2 層目以降の重みの転送時間をなくすことができる。しかし、SRAM のメモリ使用量は、実行中の層の重みと次の層で扱う重みをコピーする重みを同時に扱うため増えることになる。

5. 評価結果

前章までで述べた実装アルゴリズムを COP 上で動作させ、実行時間とメモリ使用量の比較を行う。また、CNN プログラムを各手法を組み合わせたハイブリッド方式で実行し、実行時間を計測する。

5.1 実行時間の比較

前章までで述べた以下の (1) から (5) までの手法を NXP マイコン上で実行する。

- (1) ベースライン SIMD
- (2) ベースライン SIMD(効率化)
- (3) im2col(SDRAM)
- (4) im2col(SRAM)
- (5) im2col(DMA)

なお、im2col(DMA) は、im2col(SRAM) から畳み込み層で用いる重みをコピーする時間を削除することにより実行時間を見積もった。

まず、im2col(FLASH) とそれぞれの拡張手法との比較結果を図 9 の (a) に示す。結果から、すべての拡張手法において、扱う重みデータ量が大きい際に im2col(FLASH) のような実行時間の大きな跳ね上がりはなく、性能がよく

なっていることがわかる。

次に、拡張手法をそれぞれ比較した結果を図 9(b) に示す。まず、ベースライン SIMD について考察する。実行結果から、キャッシュ容量である 32KB 付近を超えても実行時間が線形に増加していることがわかる。よってキャッシュ効率を向上させつつ、実行することができていることがわかる。また、ベースライン SIMD のパイプライン処理を効率化させたベースライン SIMD(効率化) は、ベースライン SIMD と同様に実行時間が線形に増加しつつも、性能が向上していることがわかる。

SDRAM を用いた効率化手法である im2col(SDRAM) について考察する。この結果から、キャッシュ容量である 32KB 以下の重みを扱う場合ではベースライン SIMD よりも実行時間が短くなっていることがわかる。しかし、32KB を超えたあたりから、実行時間の増え方が大きくなり、50KB 付近ではベースライン SIMD よりも実行時間が長くなっている。このことより SDRAM を用いた効率化手法は、キャッシュ依存の実装アルゴリズムであることがわかる。

SRAM を用いた効率化手法である im2col(SRAM) について考察する。結果から、ベースライン SIMD と同じく、キャッシュ容量である 32KB を超えても線形に実行時間は増加しており、かつベースライン SIMD よりも高速であることがわかる。

また、im2col(SRAM) と DMA 転送の効果を見積もった im2col(DMA) との比較をすると、DMA による高速化比率は約 11%という結果となり、そこまで大きな高速化は得られないと分かった。さらに、DMA を行う場合、CPU が別のプログラムの動作時に SRAM を用いる場合などに衝突が起こってしまう可能性があるためその都度評価が必要になるという問題もあり、手間のかかる DMA 実装を行う必要性は低いと考えられる。そのため、im2col(DMA) は採用しないこととした。

すべての拡張手法を比較した結果から、扱う重みデータ量がキャッシュ容量である 32KB 以下の場合には im2col

表 3 CNN の各畳み込み層で用いる重みデータ量

	データ量
1 層目の重み	2.4KB
2 層目の重み	25.6KB
3 層目の重み	51.2KB

(SDRAM) が、以上であるならば im2col(SRAM) が最速であると分かる。

5.2 メモリ使用率の比較

各手法における SRAM と SDRAM の使用率の比較を行う。評価の際には、CIFAR10[13] に対する 3 層の畳み込み層を持つ CNN を用いて比較を行う。評価した CNN における各畳み込み層で用いる重みデータ量と各手法における結果を表 3 と表 4 に示す。

まず、ベースライン SIMD については、FLASH メモリからの読み取りによって実行するため、SRAM、SDRAM ともに使用率は 0% である。

im2col(SDRAM) については、すべての層の重みを初期化時に SDRAM 置くが、SDRAM の容量が大きいため、使用率は 0.03% に留まり他のプログラムが SDRAM を用いたとしても問題はないと考えられる。

im2col(SRAM) については、畳み込み層 1 層を呼び出す直前にその層の重みをコピーするため、1 層分の重みを SRAM 上で使用することになる。そのため最大のメモリ使用率はもっとも大きな重みを扱う 3 層目の重みを SRAM にコピーした場合であり、使用率は 10% となっている。

im2col(DMA) については、計算している畳み込み層で用いる重みと、次の畳み込み層で用いる重みを同時に SRAM 上で扱うことになるため SRAM の使用率は増加する。そのためメモリの最大使用率は、前後 2 層の重みの合計値が最大になるときである。今回の CNN の場合、2 層目と 3 層目の重みの合計値が最大であり、使用率は 15% となっている。

前節での結果から、DMA による高速化比率が約 10% に留まり、かつ SRAM の使用率も大きいことから、COP において手間のかかる DMA 実装は適していないのではないかと考えられる。

5.3 ハイブリッド実行

前節までの結果により、COP で CNN を動かす際には畳み込み層で用いる重みの大きさに応じて処理方法を選択する方式を取る必要があると考えられる。

(1) 扱う重みがキャッシュ容量以下の場合

im2col(SDRAM) を選択するのが良い。ほかの方式よりも高速であり、初期化時に大容量の SDRAM に重みを配置するだけでよいので実装も容易である。

(2) 扱う重みがキャッシュ容量以上の場合

表 4 各手法の最大メモリ使用率

	SRAM(512KB)	SDRAM(256MB)
ベースライン SIMD	0%	0%
im2col(SDRAM)	0%	ネットワーク全体の重み 0.03%
im2col(SRAM)	最も大きな層の重み 10%	0%
im2col(DMA)	前後 2 層の重みの最大合計値 15%	0%

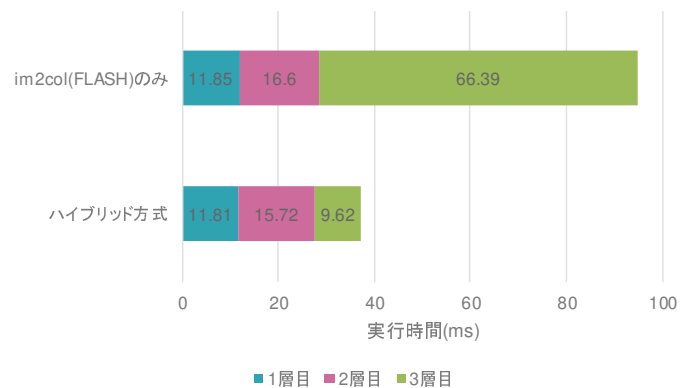


図 10 cifar10 の計測結果

im2col(SRAM) を選択するのが良い。しかし、ほかの処理における SRAM の使用の衝突などによって十分な SRAM の空き領域が確保できないと考えられる場合、ベースライン SIMD (効率化) を用いたほうが良い場合もある。

(3) 扱う重みが SRAM 容量を超えている場合

ベースライン SIMD (効率化) を選択するのが良い。この選択手順を用いて、cifar10 の CNN のプログラムを NXP マイコン上で実行する。

1 層目は、2.4KB の重みを用いる。これは NXP マイコンのキャッシュ容量である 32KB 以下であるため、im2col(SDRAM) を用いる。

2 層目は、25.6KB の重みを用いる。よって同じく im2col(SDRAM) を用いる。

3 層目は、51.2KB の重みを用いる。これはキャッシュ容量より大きいことから im2col(SRAM) を用いる。

比較対象として、すべての層を im2col(FLASH) で動作させる。比較結果を図 10 に示す。結果としてすべての層において高速化したとともに、全体として、2.5 倍の高速化が得られた。

6. おわりに

本研究では、COP 向けに CMSIS-NN の畳み込み関数の拡張を行い、性能の高い実装アルゴリズム手法を複数提案

し、実行した。結果として、もともとの実装アルゴリズムに対してすべての提案手法が性能向上した。また、各手法のメモリ使用量や動かしたいネットワークの大きさに応じて手法を使い分けることによってさらなる性能向上が見込めるといった結果になった。

今後は、DNN 向けコンパイラである Glow の調査、およびマイコン上での DNN の性能向上を目指したい。

謝辞 本研究を進めるにあたりご協力いただいたトヨタ自動車株式会社の皆様に深く御礼申し上げます。

参考文献

- [1] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. Visual wake words dataset. arXiv preprint arXiv:1906.05721, 2019.
- [2] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. arxiv preprint arXiv:1804.03209, 2018.
- [3] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, Vol. 13, No. 3, pp. 55–75, 2018.
- [4] Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, Paul N. Whatmough. MICRONETS: NEURAL NETWORK ARCHITECTURES FOR DEPLOYING TINYML APPLICATIONS ON COMMODITY MICROCONTROLLERS. arXiv preprint arXiv:2010.11267, 2010.
- [5] ARM 社 CPU Cortex-M7.
<https://www.arm.com/ja/products/silicon-ip-cpu/cortex-m/cortex-m7>(参照 2021/2/16).
- [6] 中島毅. 受注開発型ソフトウェア開発における誤りの除去と管理に関する研究. PhD thesis, 早稲田大学, 2008.
- [7] 高田広章. 組込みシステム開発技術の現状と展望. 情報処理学会論文誌, Vol. 42, No. 4, pp. 930–938, 2001.
- [8] 松本斗貴, 中本幸一, 趙茜. 組込みシステムにおける深層学習フレームワークによる学習結果を用いた認識機能の自動生成の試み. 第 80 回全国大会講演論文集, Vol. 2018, No. 1, pp. 103–104, 2018.
- [9] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient neural network kernels for arm cortex-m cpus. arXiv preprint arXiv:1801.06601, 2018.
- [10] github Armsoft-ware CMSIS-NN.
https://github.com/ARM-software/CMSIS_5/tree/develop/CMSIS/NN
- [11] STM32F746ZG - STMicroelectronics.
<https://www.st.com/ja/microcontrollers-microprocessors/stm32f746zg.html> (参照 2021 2 月 16 日).
- [12] i.MX RT1060 Evaluation Kit — NXP.
<https://www.nxp.com/design/development-boards/i.mx-evaluation-and-development-boards/mimxrt1060-evk-i.mx-rt1060-evaluation-kit:MIMXRT1060-EVK> (参照 2021 2 月 16 日).
- [13] CIFAR10 and CIFAR100 datasets.
<https://www.cs.toronto.edu/~kriz/cifar.html> (参照 2021 2 月 16 日)