Vol.2021-SLDM-196 No.12
2021/12/1

並列演算を効率化する組込みシステム向けデータ整形機構

間宮 暉之[1,a)]　山﨑 信行[1,b)]

概要：近年, AI 等の膨大な演算量のアプリケーションでは, 演算に必要なデータが非連続なメモリアドレスに配置されている場合が多い. これらのアプリケーションでは, 通常のメモリアクセスで効果的に必要なデータのみにアクセスすることが困難であり, 演算当たりのメモリアクセス回数が増える. 演算性能を向上させるために単純に演算器の数のみを増やすというアプローチをとると, 演算密度の低い無駄の多い演算の回数が増加する. 消費電力の観点から演算効率がより重要視される組込みシステムにおいては, この問題は大きな課題となる. この問題に対処するために本論文では, 演算に必要なデータを転送しつつ演算器が演算しやすいように並び替えを行うデータ整形機構 (DRU: Data Rearrangement Unit) を提案する. 演算器が演算しやすいように DRU がデータを転送しながら整形することで, メモリアクセス回数を減らしつつ演算密度を向上させることができ, 従来手法と比較して非常に高効率な並列演算を可能とする. 本論文では, RMTP SoC[1][2] 上に DRU を設計・実装し, 畳み込み演算のスループットを約 13%の面積増加で最大で 94 倍まで向上させることに成功した.

キーワード：ニューラルネットワーク, データ並列性, データ整形, 組込みシステム

Data Rearrange Unit for Efficient Parallel Data Computation in Embedded System

Akiyuki Mamiya[1,a)]　Nobuyuki Yamasaki[1,b)]

Abstract: Recently demands for computation intensive applications such as convolutional neural networks (CNNs) have been increasing. In these applications, valid data for computation are allocated in non-continuous addresses. Therefore, common burst memory access pattern results in a low spatial locality of valid data for computation per access. As a result, just increasing the number of data parallel execution units does not greatly improve in throughput, as computation resource is wasted by computing invalid data. This is especially a problem in embedded systems in which constraints in power consumption provoke a requirement for high computation efficiency. In this paper, we introduce a Data Rearrange Unit (DRU), a hardware unit rearranging computation data to increase spatial locality of valid data. The DRU drastically reduces the main memory access rate and increases computation efficiency by decreasing memory access to reduce power consumption. We demonstrate the effectiveness of our DRU by implementation on the RMTP SoC[1][2] improving convolution throughput on a data parallel execution unit by a maximum of 94times, while only increasing the total cell area by about 13%.

Keywords: neural network, data-parallel, data rearrange, embedded-systems

1. はじめに

近年, パターン認識や AI アプリケーションではニューラルネットワークが利用されている. ニューラルネットワークのパラメータ数は, 認識プロセスの複雑化に伴いサイズが拡大傾向にある. その結果, 複数の隠れ層で特徴を抽出する 畳み込みニューラルネットワーク (CNN) が主流となっている. CNN の演算は, アプリケーションにおけるニューロン数の爆発的な増加による膨大な計算量に対応す

[1] 慶應義塾大学大学院
[a)] mamiya@ny.ics.keio.ac.jp
[b)] yamasaki@ny.ics.keio.ac.jp

るために, データ並列性を抽出し, SIMD 器, ベクトル演算器, GPU 等を用いて並列演算を行うことが主流である.

CNN は全結合層と畳み込み層で構成されている. 計算負荷の最も大きな割合を占めるのは畳み込み層 [3] であるため, 畳み込み演算の並列演算の効率が重要である. 畳み込み層では, 複数の MAC (Multiply-Accumulate) 命令を並列に実行することでデータ並列性を抽出できる. しかし, 畳み込み演算で必要となる有効データは, メモリ上で連続しておらず散っている. そのため, 1 回の演算で並列演算器ができるだけ多くの有効データを演算できるようにする必要がある. また, 非連続なメモリアドレスへのアクセスは演算ごとのメモリアクセス回数を肥大化してしまう. これらの無駄な演算やメモリアクセスは, 特に消費電力を重要視する組込みシステムでは課題となる. ほとんどの組込みシステムはバッテリー駆動であるため, 過剰な電力使用はシステム全体の稼働率低下に繋がる.

これらのメモリアクセスを軽減する手法には, ソフトウェアとハードウェアのアプローチが存在する. ソフトウェアによるアプローチは, 一時的に演算用のデータをメインメモリ内に割り当てるため, メモリアクセスの電力コストがかかり, 膨大なメインメモリ領域を必要とする [4]. また, メインメモリへの大量アクセスによりキャッシュの大部分が汚染される. ハードウェア手法は, ほとんどが CNN アクセラレータに専用の手法 [5][6][7] であるため, 汎用的な演算器へ対応させることは困難である. 面積の制限によりメインメモリや演算器の数に制限のある組込みシステムにおいてこれらの手法は, 演算のオーバーヘッドの解消に伴い, 新たな面積オーバーヘッドを生んでしまう可能性が高い.

本論文ではまず, ニューラルネットワークにおける有効データの非連続なメモリ配置について説明する. 次に演算とメモリアクセスのコストを削減するためのデータ整形機構 (DRU: Data Rearrangement Unit) を提案する. そして, 複数のニューラルネットワークデータセットを用いて評価を行う.

本論文の構成は, 以下の通りである. 2 章では, CNN の背景と有効データのメモリ配置について説明する. 3 章では, 演算に必要なデータを転送しつつ演算器で演算しやすいように整形する DRU を提案する. 4 章では, 提案手法の評価を行い, 5 章では, 論文の結論と今後の研究について述べる.

2. 背景

2.1 畳み込みニューラルネットワーク

CNN は入力データの特徴抽出のために, 畳み込み層と全結合層で構成されている. 畳み込み層は全入力データと畳み込み演算を行うカーネルを用いることで, 特徴量の抽出を行う. **図 1** に示すように CNN 内の各ニューロンは幅, 高さ, チャネルの 3 次元の要素で構成されている. 各パラメータを**表 1** に示す.

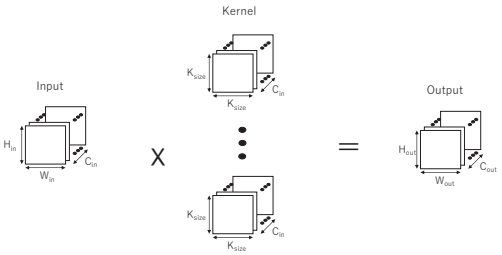

図 1 畳み込みニューラルネットワーク

表 1 CNN パラメータ

パラメータ	説明
H_{in}/W_{in}	入力データの高さ/幅
H_{out}/W_{out}	出力データの高さ/幅
C_{in}/C_{out}	入力/出力ニューロンのチャネル数
k_{size}	カーネルサイズ
S	ストライド

2.2 非連続アドレス領域へのメモリアクセス

前述した畳み込み層では複数の MAC 演算を行うために, k_{size} を元にして入力データの有効なデータにアクセスを行う必要がある. メインメモリがキャッシュライン単位でアクセスされるシステムにおいては, キャッシュライン内の有効なデータが占める割合が高いほどメモリアクセスの効率がよい. CNN の k_{size} は, 小さい値をもつことが多いため, メモリアクセスの大部分は有効なデータが占めない. また, 一度のメモリアクセスでカーネル全ての有効データにアクセスすることが不可能なため, **図 2** で示すように複数回のメモリアクセスが必要となる. これらの要素によりメモリアクセス回数は増えてしまう.

アクセス回数を削減する手法として, im2col[4] はソフトウェアによるアプローチとして利用されている. この手法の重大な欠点は, 一時的にメインメモリ内の大きな割合を整形されたデータが占めてしまうという点にある. そのため, メインメモリのサイズが限定される組込みシステムにおいては, この手法は使用が困難である. また, データキャッシュの汚染も行われてしまう.

ハードウェアによるメモリアクセス改善のアプローチとして, 様々な CNN 専用演算器の開発が行われている. SPE[8] では, CNN 内の同値のカーネルニューロンの pruning が行われることで メモリアクセスの回数を減らすことに成功している. その他のハードウェアアプローチとしては, CNN 専用のアクセラレータの開発が行われており, シストリックアレイ [5], spatial architecture[6], 複数チップ [7] などの多種に渡る. これらのアプローチは CNN 専用のチップを必要とし, 組込みシステムで使用するにはコストが高く, 柔軟性が低い.

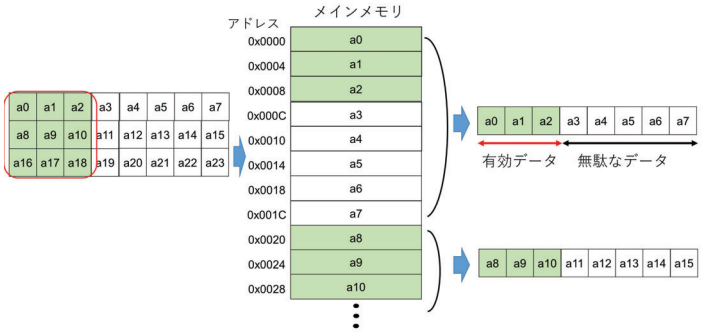

図 2 キャッシュライン内の有効データ

3. 提案手法

3.1 データ整形機構 (DRU)

畳み込み演算ではその特徴から, 入力データを含むキャッシュラインは，複数回アクセスされる．例として, **図 3** では, k_{size} が 3 の時, キャッシュラインの一部分のみが有効であるため同じキャッシュラインは 8 度もアクセスされる．式 1. ではチャネル数 C_{in} で畳み込み演算にかかるメモリアクセス回数 M を示す．本論文では，この問題を解決するためにメモリアクセス回数を減らすことのできる DRU を提案する．

$$M = k_{size} \times (W_{in} \times H_{in}) \times C_{in} \quad (1)$$

DRU は, ニューラルネットワークアプリケーション等の有効データの低い空間的局所性を改善するため，有効データが連続アドレスへ再配置されるように整形する．提案アーキテクチャの DRU は**図 4** で示されるように有効データの整形を行うためにメインメモリと同じバスに接続されるハードウェアモジュールである．

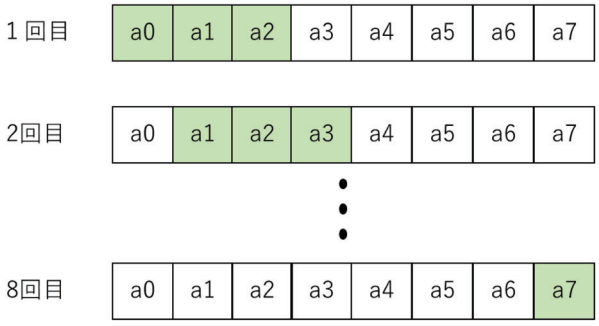

図 3 有効データへのキャッシュラインアクセス

DRU は DMAC (Direct Memory Access Controller), DC (Data Composer), DD (Data Decomposer) の 3 つのサブユニットから構成されている．DMAC は，メモリアクセス回数の改善のために DRU 上に実装される．DRU では初めに，メインメモリ内の畳み込み演算の入力データに対し，DMAC を用いて, 図 4 の Pre-modified Data DPRF (Dual Port Register File) に演算に必要となる連続データを (不要データも含めて) バーストメモリアクセスで必要最低限のクロックサイクルで一度に書き込む．Pre-modified Data DPRF はワーキングメモリとして使用され, 保持できる上限まで入力データを読み込む．DPRF が空になった場合, 入力データ全体がワーキングメモリ内に取り込まれるまで自動でバーストメモリアクセスを行う．

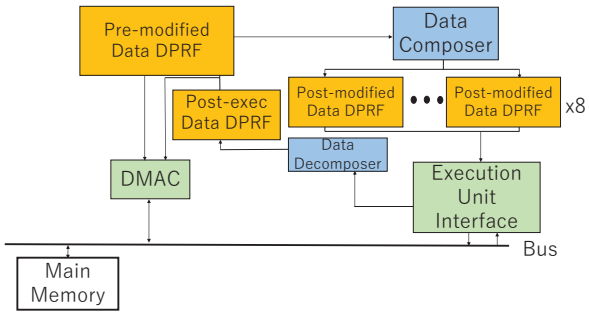

図 4 データ整形機構のアーキテクチャ図

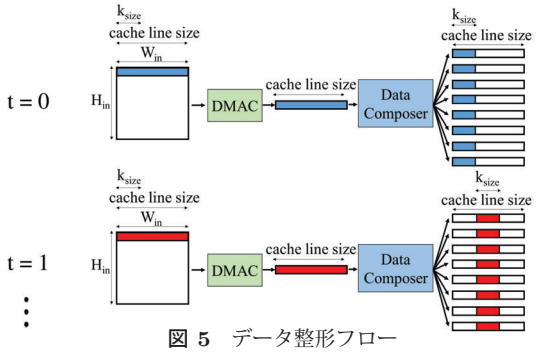

図 5 データ整形フロー

次に, サブブロックの DC では, Pre-modified Data DPRF が保持する行列データに対してデータ整形を行う．DC は, 制御レジスタを持ちプロセッサから整形のパラメータを書き込むことで整形を行う．整形のパラメータとして, 入力行列サイズの W_{in} と H_{in}, カーネルサイズの k_{size} を用いる．

データ整形は**図 5** に示すフローで行われる．DMAC を用いて読み込んだキャッシュラインに対して，DC では Valid Data Selector (VDS) が k_{size} に応じて毎クロック最大で 8 つの有効データブロックを抽出する．VDS は前のクロックサイクルで抽出した有効データブロックを格納したアドレスに続くようにオフセットを計算し，有効データが連続のアドレスに配置されるようにその有効データを詰めていく．キャッシュラインの読み込みは，入力行列の縦方向に k_{size} 回行われる．この整形プロセスにより，カーネルニューロンのサイズに対応し，一度の MAC 演算で並列演算を行うことのできる密度の高いデータブロックが生成される．

図 6 に示すように抽出のプロセスはパイプライン化されるように設計している．最大で 8 つの整形データブロックが毎クロック生成され, Post-modified Data DPRF に格納される．この整形プロセスは, 入力行列データ全てに対して行われるまで続く．VDS は W_{in}, H_{in} を参照し行列の最後の要素を抽出したことを判断する．Post-modified Data DPRF に整形データを格納後, DRU は DMAC を介してメインメモリに整形後データを書き戻すことができる．書き

戻しを行うと, メモリアクセス回数 M は式.2 で示すように $1/(1+k_{size})$ の回数分だけ削減される.

$$M = \frac{1+k_{size}}{k_{size}}(W_{in} \times H_{in}) \times C_{in} \quad (2)$$

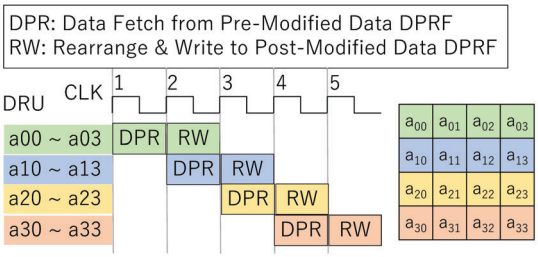

図 6 整形のパイプライン化

k_{size} が 1 よりも大きい場合, 整形データのメインメモリへの書き戻しはメモリアクセス回数を削減できる. しかし, 整形データ全てをメインメモリに書き戻す場合, メインメモリ内の大部分を一時的なストリーミングデータが占めてしまう. そのため本論文では, この問題に対処するために execution unit interface を DRU に設計・実装し, **図 7** に示すように DRU とプロセッサ内の演算器に専用データパスを設けることで Post-modified Data DPRF に格納されている整形後データを演算器に直接転送できるように設計する. データパスを通る整形後データは, 演算器内で事前に確保されたレジスタに格納される. 演算器は, 整形データとの MAC 演算を転送直後に行うことができるため, メモリアクセスを行う必要がない. よって, DPRF のサイズの制限などによるパイプラインのストールがない限り, 毎クロック演算を行うことが可能である.

演算後の結果は, 演算後のデータフローを監視する write-back unit を設けることで切り出すことができる. その後, 専用データパスを介して DRU に演算結果を転送する. これらの, 演算器側へのマイナーな変更でメモリアクセス回数は劇的に改善され, 式.3 で示す回数のみメモリアクセスが必要となる.

$$M = \frac{W_{in} \times H_{in}}{k_{size}} \quad (3)$$

DRU に転送された演算結果は, DD でメインメモリに書き戻す形に整形される. DMAC のバーストアクセスを最大限に活用するために, 演算結果はキャッシュライン内の有効データの密度が最大となるように連続するアドレスに詰めて配置するように設計する. 演算器から書き戻された演算済データは, オフセットを割り当てられ, Post-exec Data DPRF 内に書き戻される. DD に整形済みのデータが存在すれば, それらのデータは DRU の DMAC を用いてバーストアクセスでメインメモリに書き戻される.

専用データパスを用いた DRU は, パイプライン化されるため演算と整形を同クロックサイクルで行うことが可能である. 演算データをメインメモリから演算器に転送する必要がないため, データ整形の効率はソフトウェアのアプローチと比較して各段に上昇している. この演算器との専用データパスを有する DRU は, 演算器側の少量の変更のみで演算効率を劇的に改善することが可能となる.

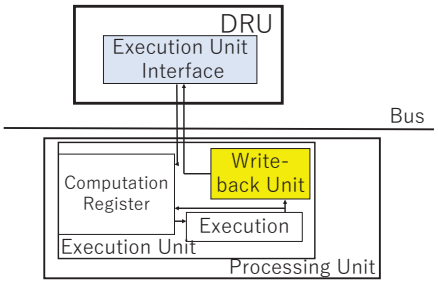

図 7 Execution Unit Interface

4. 評価

4.1 対象プロセッサ

本論文において, DRU は**図 8** に示す Space Responsive Multithreaded Processor (SRMTP) SoC[2] の 256bit 幅のメモリバス上に設計・実装する. SRMTP は宇宙機での使用をメインターゲットとした, 分散リアルタイム処理用組込み SoC である. SRMTP はリアルタイム処理用プロセッサとして 8-way 優先度付き SMT プロセッサの RMTPU を集積している. 宇宙機などの分散リアルタイムシステムの制御では, タスクを優先度順に時間制約内に実行するリアルタイム実行を行う必要があり, RMTPU は 8 つの優先度付きスレッドを優先度順に実行することで, リアルタイム実行を可能としている. また, リアルタイム実行の妨げとなるコンテキストスイッチにかかる時間をコンテキストキャッシュというコンテキスト情報 (PC, GPR, FPR) を保持する専用キャッシュを用いることで大幅に削減している. また, RMTP の別バージョンである D-RMTP は [1], ヒューマノイドロボットの制御等に用いられている.

RMTPU の各ユニットのパラメータを**表 2** に示す. RMTPU はデータレベル並列性を抽出する演算器としてベクトルユニットを実装している. 評価では, DRU の execution unit interface の対象演算ユニットはこのベクトル演算器が用いられている. ベクトル演算を行う際には, 最大 512 エントリを保持可能なベクトルレジスタを各スレッドで行う演算のデータ幅に応じて予約する. また, RMTPU のベクトルユニットは, 複数のベクトル演算命令をあたかも一つのベクトル演算命令であるかのように扱うことのできる複合演算命令の機能を有している. 複合演算命令は, 複数のベクトル演算命令を複合演算命令バッファに格納することで使用する. バッファされた命令は, ベクトル演算器用の専用命令で発行される. 本論文の評価では, MAC 命令を複数回実行するため, 複合演算命令の機能を使用することで, ベクトル演算時の命令フェッチを行う回数を削減し

ている．

本評価では，前述した専用命令を用いずに複合演算バッファ内の命令を発行可能にするコントロール信号を追加する．こらのコントロール信号は，DRU から整形後データの転送を検知するとアクティブになる．このマイナーな変更により，DRU とベクトルユニット間での整形と演算をパイプライン化することができる．具体的にはまず，DRU 内で有効データの整形と Post-modified Data DPRF への格納が並列に行われる．整形が行われたデータは，ベクトル演算器へ転送され，同クロックサイクルで並列に実行される．この際，複合演算命令の機能を用いることで各演算命令のフェッチレイテンシが 1 クロックサイクルとなり，演算の実行フローを効率化している．

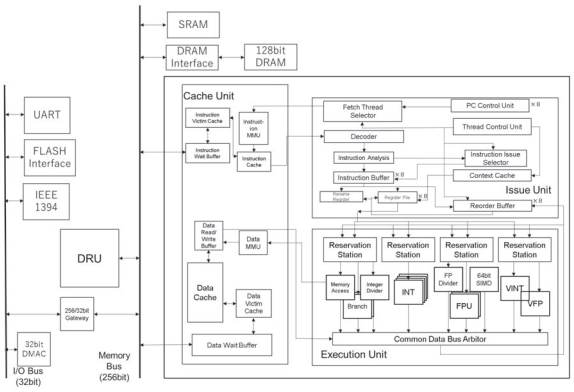

図 8 SRMTP ブロック図

表 2 RMTPU パラメータ

アクティブスレッド数	8
キャッシュスレッド数	32
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32bit × 32 entry × 8set
整数リネームレジスタ数	32bit × 64 entry
浮動少数点レジスタ数	64bit × 8 entry × 8set
浮動小数点リネームレジスタ数	64bit × 64 entry
整数演算器	4 + 1 (割り算器)
浮動小数点演算器	2 + 1 (割り算器)
64bit 整数演算器	1
整数ベクトル演算器	1 (8IU × 2lines)
浮動小数点ベクトル演算器	1 (4FPU × 2lines)
分岐ユニット数	2
メモリアクセスユニット数	1

4.2 演算効率と面積評価

性能評価は，Cadence XCELIUM Parallel Simulator を用いたシミュレーションで行う．ベンチマークとしては，CNN で主に使用されるイメージセットの CIFAR-10[9], ImageNet[10], SVHN[11] を使用する．各データセットの画像は SRMTP SoC の DRAM サイズに合わせて ImageNet は 256×256, SVHN は 32×32 に画像サイズを統一する．各ベンチマークでは使用画像数を 10 に統一し，異なる k_{size} で畳み込み演算にかかるクロックサイクル数の評価を行う．DPRF のサイズは，Pre-modified Data DPRF と Post-exec Data DPRF で 4KB に設定する．Post-modifed Data DPRF は，512B, 1KB, 2KB それぞれのサイズで評価をとる．**図 9**, **図 10**, **図 11** に CIFAR-10, ImageNet, SVHN それぞれで DRU とベクトルユニットを使用する場合とベクトルユニットのみの場合の評価結果を示す．ImageNet のアプローチにおいて k_{size} が 4，バッファサイズが 2KB の際，スループットは 94 倍まで向上している (図 10 参照)．

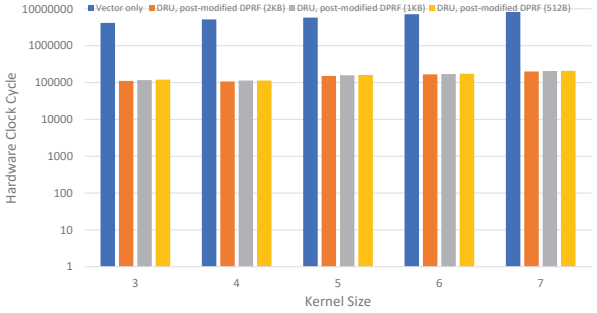

図 9 CIFAR-10

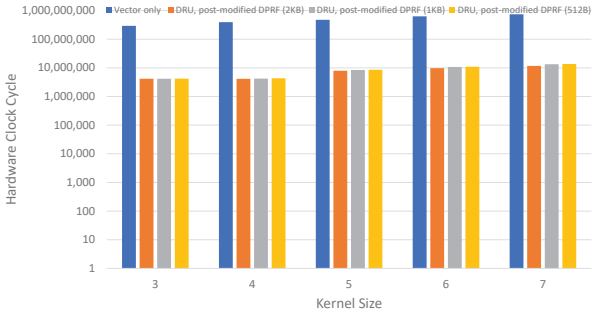

図 10 ImageNet

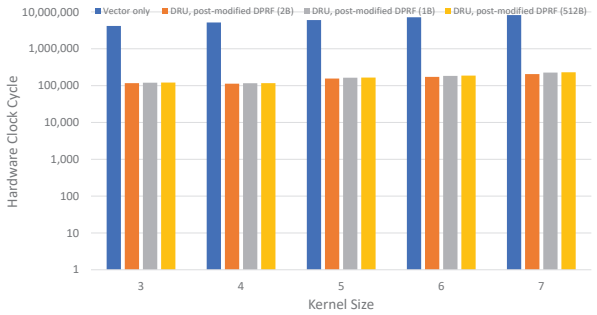

図 11 SVHN

面積評価として，論理合成後の各 Post-modified Data DPRF サイズを持つ DRU のセルエリアを SRMTP SoC ののセルエリアと比較し，**表 3** に示す．512B の Post-modified Data DPRF は，SRMTP 全体のセルエリアの約 12.8%のみを占めるため面積オーバーヘッドはそれほど大きくない．

表 3 DRU セルエリア

モジュール名	セルエリア (um^2)	DRU の割合
SRMTP	54,112K	-
SRMTP + DRU (512B Post-modified Data DPRF)	62,072K	12.8%
SRMTP + DRU (1KB Post-modified Data DPRF)	62,528K	13.4%
SRMTP + DRU (2KB Post-modified Data DPRF)	63,398K	14.6%

4.3 考察

DRU のパフォーマンスは全ての k_{size} においてベクトルユニットのみを使用する場合と比較して，演算スループットが大幅に向上している．一方で，ベクトルユニットのみの評価では，k_{size} に比例してメモリアクセス回数が増加するため，実行クロックサイクル数は増えることが確認できる．DRU を用いた場合，小さめの k_{size} である 3 から 5 の値では，CIFAR-10，SVHN の小さい画像サイズで平均で約 38 倍のスループット向上，ImageNet の大きい画像サイズで平均で約 74 倍のスループット向上となる．大きめの k_{size} の 6 と 7 では，小さい画像サイズでスループット向上は平均で約 40 倍に向上する．しかし，ImageNet では，スループットは平均で約 60 倍となり向上率が低下する．これは画像サイズが大きい場合，DMAC による Pre-modified Data DPRF への読み込みが頻繁に行われるため，パイプラインのストールが行われる回数が増えることに起因すると考えられる．また，k_{size} は Post-modified Data DPRF の占有率に影響を与える．Execution unit interface の読み出し帯域幅は，Post-modified Data DPRF の書き込み帯域幅の 1/8 であるため，k_{size} が大きいほど，頻繁に整形後データが DPRF を占有してしまう．

Post-modified DRPF の 1KB，512B へのサイズの縮小は，2KB と比較して小さい k_{size} では平均で約 1.08 倍，大きい k_{size} では平均で約 1.16 倍の実行サイクル数の増加となる．ベクトルユニットのみの評価と比較して，DPRF サイズの変更に伴う数千クロックサイクルの実行時間の増加は，平均で約 0.001%のみの実行クロックサイクルの増加となる．そのためほとんど影響がないと考えることができ，SRMTP への実装では，DRPF のサイズは 512B へ削減することが最適であると考えられる．512B の DPRF では，SRMTP 全体のセルエリアの約 12.8%のみを占めるため DRU は比較的シンプルな組込みシステムにおいても実用可能であると考えられる．

5. 結論と今後の課題

CNN の大部分を占める畳み込み演算では，演算に必要なデータは非連続なメモリアドレスに配置されるため，並列演算器では密度の低い行列演算が行われる．無駄な演算による消費電力の浪費は, バッテリー駆動の組込みシステムでは課題となる．

本論文では必要なデータを整形することで, 行列演算の密度を向上させるデータ整形機構 (DRU) を提案する．演算に必要なデータは連続するメモリアドレスに配置されるため，メモリアクセス回数が削減される．さらに DRU は, 整形と演算を execution unit interface を介することでパイプライン化を行った．

評価結果より，SRMTP に実装された DRU はベクトルユニットのみを用いた場合と比較して，演算のスループットを最大 94 倍向上することができた．

DRU は SRMTP の約 13%のセルエリアを占め面積オーバーヘッドは組込み用としては許容範囲内と考えられ, 様々な組込みシステムで用いることが可能であると考えられる．

今後の課題として，DRU をスカラユニットやその他のデータ並列演算器 (SIMD 器や GPU 等) で応用を可能とする必要がある．さらに，畳み込み演算だけでなく疎行列演算など, その他の低密度な演算に対応可能にしていく．

参考文献

[1] 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.

[2] Shota Nakabeppu, Yosuke Ide, Masahiko Takahashi, Yuta Tsukahara, Hiromi Suzuki, Haruki Shishido, and Nobuyuki Yamasaki. Space responsive multithreaded processor (srmtp) for spacecraft control. In *2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pp. 1–3, 2020.

[3] Zichao Yang, Marcin Moczulski, Misha Denil, Nando Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. 12 2014.

[4] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In Guy Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Université de Rennes 1, Suvisoft.

[5] Lin Li, Jianhao Hu, Qiu Huang, and Wanting Zhou. Bit-serial systolic accelerator design for convolution operations in convolutional neural networks. *IEICE Electronics Express*, Vol. 17, pp. 20200308–20200308, 10 2020.

[6] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, Vol. 52, No. 1, pp. 127–138, 2017.

[7] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, 2014.

[8] Maurice Yang, Mahmoud Faraj, Assem Hussein, and Vincent Gaudet. Efficient hardware realization of convolutional neural networks using intra-kernel regular pruning. 03 2018.

[9] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, Vol. 115, No. 3, pp. 211–252, 2015.

[11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 01 2011.