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
CPSY2025-69,DC2025-141(2026-03)

OSCARベクトルマルチコアにおける推論処理の高速化に向けたローカルメモリ管理手法

権藤　創太[†]　上林　嶺[†]　朱　允楷[†]　水本　幸希[†]　野谷　優仁[†]
北村　俊明[†]　笠原　博徳[†]　木村　啓二[†]

† 早稲田大学　〒169–8555 東京都新宿区大久保 3–4–1
E-mail: †sota0147@asagi.waseda.jp, rei107@suou.waseda.jp, zhuyunkai@kasahara.cs.waseda.ac.jp,
kouki.mizumoto@fuji.waseda.jp, notani@moegi.waseda.jp, t-kita@kta.att.ne.jp, kasahara@waseda.jp, keiji@waseda.jp

あらまし　組み込み環境においても深層学習推論処理が広く用いられるようになると共に，これら処理のさらなる高速化かつ低消費電力化が求められている．このためのアプローチとしてコンパイラと協調動作してベクトル化とローカルメモリ最適化を行うベクトルマルチコアは有用である．本稿では，コンパイラ協調ベクトルマルチコアチップ上での推論処理を高速化するメモリ管理手法を提案する．本手法では従来のローカルメモリ管理を拡張し，ベクトルアクセラレータ用のメモリ管理を実現する．性能評価にはResNet-18を用い，FPGA上に実装したベクトルマルチコアチップで検証を行った．その結果，1コアでの逐次実行に対しベクトル化と4コア並列化により，畳み込み処理では最大126.4倍，全結合層では13.6倍の速度向上が確認された．
キーワード　ベクトルアクセラレータ，マルチコアプロセッサ，深層機械学習，並列処理

Local Memory Management Scheme for Inference Acceleration on the OSCAR Vector Multicore

Sota GONDO[†], Rei KAMBAYASHI[†], Yunkai ZHU[†], Koki MIZUMOTO[†], Masahito NOTANI[†],
Toshiaki KITAMURA[†], Hironori KASAHARA[†], and Keiji KIMURA[†]

† Waseda University
E-mail: †sota0147@asagi.waseda.jp, rei107@suou.waseda.jp, zhuyunkai@kasahara.cs.waseda.ac.jp,
kouki.mizumoto@fuji.waseda.jp, notani@moegi.waseda.jp, t-kita@kta.att.ne.jp, kasahara@waseda.jp, keiji@waseda.jp

Abstract　As deep learning inference processes have been widely used even in embedded areas, their higher performance with lower power dissipation has also been strongly demanded. Using parallelizing compiler cooperative vector multicores with local memory is a promising approach to realize it. This paper proposes a local memory management scheme to accelerate inference on the OSCAR vector multicore architecture, co-designed with the OSCAR compiler to reduce power consumption. The proposed scheme extends prior local memory management approaches and enables efficient memory management for the OSCAR vector accelerator. Performance evaluation was conducted using ResNet-18 on the OSCAR vector multicore chip implemented on an FPGA. Experimental results show that, compared with sequential execution on a single core, vectorization and multicore parallelization on four cores achieve speedups of up to 126.4× for convolution computations and 13.6× for the fully connected layer.
Key words　vector accelerator, multicore processor, deep learning, parallel computing

1. はじめに

自動運転車やAIロボット制御などの組み込みシステムでは，畳み込みニューラルネットワーク（CNN）に代表される深層学習モデルの推論処理が広く利用されている．これらの推論処理は，計算コストの大きい畳み込み処理や行列積を含み，その高速化が課題となる．

推論処理の高速化には，多数の高速な演算ユニットや広帯域

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2026 by IEICE

なメモリの導入が有効であるが，ハードウェアの単純な増強は消費電力量やそれに伴う発熱量の増大を招き，巨大なバッテリーや冷却装置などが必要となる．しかし組み込みシステムにおいては，電力消費量や実装面積などの制約によりこれら大規模なハードウェア資源の利用が困難である場合が多い．そのため，高性能と高電力効率を両立した計算機システムに対する要求が一層高まっている．

高性能かつ高電力効率な処理を達成するためには，プログラムの並列性を抽出し，電圧や周波数を適切に制御したマルチコアプロセッサ上に処理を割り当て実行する手法が有効である [1]．例えば，低電圧・低動作周波数なマルチコアプロセッサ上で並列処理を行うことにより，高動作周波数な単一プロセッサ上での逐次処理と比較し，同一性能を低消費電力で達成可能である [2]．そのため，プログラムの自動並列化や電力制御などを行う OSCAR 自動並列化コンパイラ [3] と，高速化のため各コアにローカルメモリとベクトルアクセラレータを搭載した OSCAR ベクトルマルチコアアーキテクチャが提案されている．さらに，コア近接のローカルメモリを最大限活用することも低消費電力化には重要である [4].

これまでには本アーキテクチャ上での高速化に向け，OSCAR 自動並列化コンパイラの粗粒度タスク並列処理をベースに，粗粒度タスク間で共有されるデータをローカルメモリ上で授受しデータ転送を最適化する整合分割手法とメモリ管理手法が提案されている手法などが提案されている [5], [6]．本手法では，粗粒度タスク間のデータ転送をタスク処理とオーバーラップすることで隠蔽するプリロード・ポストストア手法を併用する．

一方，ローカルメモリを持つアーキテクチャでは，ループ処理の進行に沿って，ダブルバッファを用いて将来実行するイタレーションのデータをプリフェッチする手法が広く用いられている．

CNN の畳み込み処理では粗粒度タスク間のみならず単体の粗粒度タスク内部でのみ消費されるデータもあり，このようなデータの転送にはダブルバッファリングによるデータ転送隠蔽が効果的である．そこで本稿では，OSCAR コンパイラによる粗粒度タスク間データプリロード・ポストストアを伴うローカルメモリ管理手法に，粗粒度タスク内のダブルバッファリングの導入を提案する．

また様々なモデルに対し高速化を達成するため，全結合層などに対してもベクトルアクセラレータ活用に向けた整合分割が重要となる．しかし従来の整合分割手法では，同じ次元より整合判定を行うという制約のため，ベクトル化に適した整合次元を一部最内側としたままの適用が不可であった．そこで本稿にて，多重ループ群において異なるループ次元の組合せで整合判定を行うよう手法の拡張を提案する．

本稿の構成を以下に示す：第 2. 節，第 3. 節では本稿で使用する OSCAR ベクトルマルチコアアーキテクチャと OSCAR 自動並列化コンパイラについて，第 4. 節では今回提案するローカルメモリ管理について，第 5. 節では性能評価について述べ，最後に第 6. 節でまとめる．

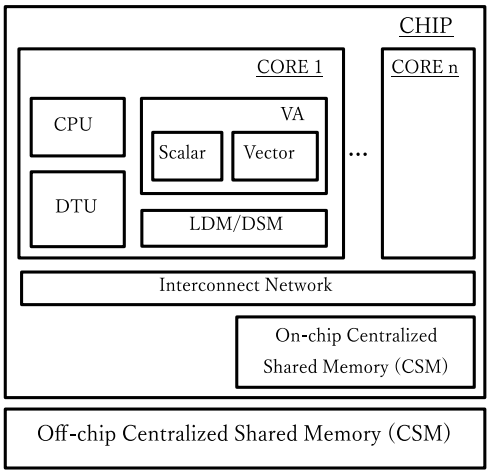

図 1　OSCAR ベクトルマルチコアアーキテクチャ

2. OSCAR ベクトルマルチコアアーキテクチャ

2.1 OSCAR ベクトルマルチコアアーキテクチャの構成

OSCAR ベクトルマルチコアアーキテクチャ (図 1) は OSCAR マルチコアアーキテクチャ [7] を基に構成されている．1 コアあたりに CPU，ベクトルアクセラレータ (VA)，ローカルデータメモリ (LDM)，データ転送ユニット (DTU) を持ち，各コアは集中共有メモリ (CSM) に接続されている．各コアの LDM は小容量なメモリであり，主に自コアの VA にデータを共有するために利用する．LDM は自コアの CPU，VA，DTU だけでなく，他コアからもアクセス可能であるため，分散共有メモリ (DSM) としての役割も担う．またチップ内外にそれぞれ LDM/DSM と比較し大容量な CSM を持つ．本稿においては，Off-chip CSM を主記憶として利用する．DTU はメモリ間のデータ転送に利用する．

2.2 ベクトルアクセラレータ (VA)

OSCAR ベクトルマルチコアアーキテクチャに搭載される VA はベクトルプロセッサ型のアクセラレータであり [8]，CPU と独立して動作可能である．

各コアの VA にはベクトル演算器とスカラ演算器が搭載される．演算で使用するレジスタはベクトルレジスタ，マスクレジスタ，整数レジスタ，浮動小数点レジスタを備え，VA 単体で条件分岐などの制御文も実行可能である．

ベクトル演算で用いる各ベクトルレジスタは 512 バイトを上限とした可変長であり，専用命令によりベクトル長を指定可能である．ベクトル命令は FP64, FP32, FP16 に対応した浮動小数点演算と整数演算が可能であり，例えば FP64, FP32, FP16 を用いた浮動小数点演算を行う場合の最大ベクトル長はそれぞれ 32 要素，64 要素，128 要素である．

また各コアのベクトルアクセラレータは，自コアの LDM にのみアクセスが可能である．そのため計算データのロード/ストアは自コアの LDM に対して行う．

2.3 データ転送ユニット (DTU)

OSCAR ベクトルマルチコアアーキテクチャに搭載される DTU は DMA コントローラの一種であり，メモリ間のデータ転

送専門のハードウェアである．本稿においては，主に自コアのLDM と主記憶間のデータ転送に利用し，VA で計算するデータの高速な供給を行う．また，DTU はプログラム実行機構 [9] を備えており，ホスト側での転送パラメータを設定が不要である．そのため CPU, VA と独立して動作可能である．

データ転送幅は 16bit, 32bit, 64bit に対応しており，レジスタを用いて自由に設定可能である．また，配列に対しては一度に三次元までの部分配列としてデータ転送が可能である．四次元以上の配列に対する転送はソフトウェアで配列を分割し，DTU プログラムで四元以上のアドレスを更新することで転送を行う．

3. OSCAR 自動並列化コンパイラ

3.1 OSCAR 自動並列化コンパイラの構成

OSCAR 自動並列化コンパイラは，C 言語で記述された逐次プログラムを入力とし，並列化 C プログラムを出力するコンパイラである．本コンパイラでは，入力されたプログラムを基本ブロック・ループブロック・関数呼び出しブロックに分割しマクロタスク (MT) を生成した後，三種類の並列化 (MT 同士の並列化を行う粗粒度タスク並列化，ループイタレーションレベルで並列化を行う中流度並列化，基本ブロック内のステートメント間の並列化を行う近細粒度並列化) を組み合わせたマルチグレイン並列化 [10] を行う．さらに，粗粒度並列処理を基にメモリ管理最適化や電力制御などの各種最適化を適用し，最適化コードを生成する．

3.2 ローカルメモリ管理

OSCAR 自動並列化コンパイラは，整合分割 [3] [5] を用いたローカルメモリ管理手法を備える．

整合分割とは，同一の配列へデータ依存のあるループ群であるターゲットループグループ (TLG) に対し，依存関係を考慮しループの分割を行う手法である．本手法により，ループ間で共有される計算データに対するデータ転送が削減され，また巨大な計算データがローカルメモリに収まるよう分割されることでベクトル化が可能となる．

3.3 OSCAR ベクトルマルチコアに向けたコンパイルフロー

様々な学習済モデルを OSCAR 自動並列化コンパイラで最適化するため，深層学習コンパイラ TVM を用いたコンパイルフロー (図 2) が提案されており，本稿でも利用する．本コンパイルフローで用いる TVM は学習済モデルと OSCAR 自動並列化コンパイラを接続するほか，CNN において長くなる傾向にある出力チャネル方向をベクトル化に適した最内側ループとするレイアウト変換を行う拡張を適用した [11].

4. 提 案 手 法

4.1 粗粒度タスク間ローカルメモリ管理手法への粗粒度タスク内データ転送オーバーラップ手法の導入

従来より主記憶とローカルメモリ間のデータ転送のオーバーラップによるデータ転送オーバーヘッド隠蔽技術は広く用いられており，OSCAR ベクトルマルチコアアーキテクチャでも，これを可能とする CPU やベクトルアクセラレータ (VA) と独立して動作可能なデータ転送ユニット (DTU) が搭載されている．

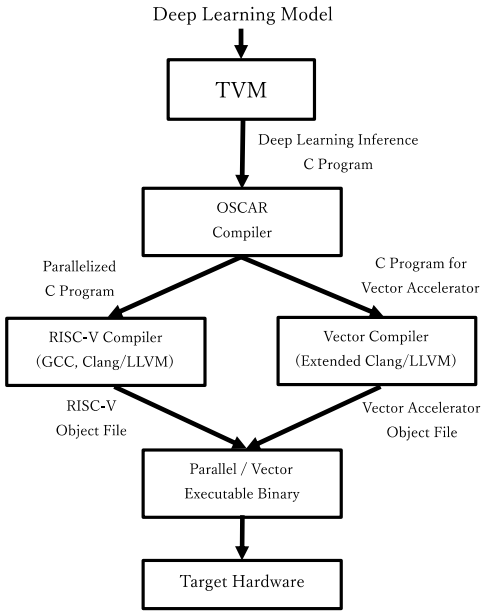

図 2　OSCAR ベクトルマルチコア向けコンパイルフロー

OSCAR コンパイラによるローカルメモリ管理では，整合分割後の粗粒度タスク（MT）間に生じるデータ転送を MT の処理とオーバーラップするプリロード・ポストストア手法が適用される．しかしながら，例えば CNN で行われる畳み込み処理では，MT 間で共有されるデータとは別に、MT 内部のみで生存するデータがあり，これがローカルメモリサイズよりも大きい場合は対象 MT の処理の進行に合わせてデータ転送を行うことになる．このようなデータ転送のダブルバッファリングを用いた計算とデータ転送のオーバラップ処理の実現は広く行われている．本稿では，このダブルバッファリングによる MT 内データ転送隠蔽を，OSCAR コンパイラのローカルメモリ管理の枠組みで実現する手法を提案する．

まず，この MT 内データ転送オーバーラップの導入の対象を，実装の簡単化のため整合分割にて生成される各ターゲットループグループ (TLG) 中に存在する畳み込み処理とする．これらの畳み込み処理に対し，指示文と畳み込み処理のパターンから構成されるヒント情報を TVM により挿入し，OSCAR コンパイラがパターンに対応したオーバーラップ転送用コードを生成する．指示文は畳み込み処理の入出力特徴量テンソル (in, out) やカーネルテンソル (w) を引数として指定する．畳み込み処理パターンの構成要素は以下の通りである：

- 畳み込み処理のループネスト構造
- 整合分割による各ループネストの分割状況
- ローカルメモリ上に確保するバッファの個数

本オーバーラップ転送では，整合分割によるローカルメモリ管理で用いられるローカルメモリ上のテンプレート配列 [5] をダブルバッファリングのバッファとして利用する．ローカルメモリ上のメモリ割り当ての様子を図 3 を用いて説明する。

コンパイラはまず，TLG 中に存在するローカルメモリ管理対象である全ての主記憶上の配列に対するテンプレート配列を生成する．図では，主記憶上の配列 “in”, “w”, “out” が，そこから

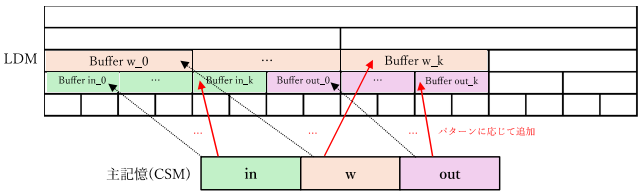

図 3　オーバーラップ転送用にテンプレート配列を追加する様子

点線矢印で指される LDM 上のテンプレート配列 “Buffer in_0, ...”, “Buffer w_0, ...”, および “Buffer out_0, ...” に対応づけられる．その後、畳み込み処理パターンに応じたオーバーラップ転送に必要な数のテンプレート配列が追加可能であるかチェックする．チェックにより生成済のテンプレート配列と追加分のテンプレート配列の合計容量がローカルメモリに収まる場合，追加分のテンプレート配列の生成が確定する．畳み込み処理パターンのバッファ個数 k に従い，k-1 個のテンプレート配列が追加生成される．図では，“in”, “w”, “out” から実践矢印で指されている “Buffer in_k”，“Buffer w_k”，および “Buffer out_k” がこれに対応する．

最後に，主記憶上の配列とローカルメモリ上のテンプレート配列間のデータを行う，DTU をコードを生成する．この時，現在のループネスト順と整合分割による各ループの分割状況より，ループ不変なデータ転送をループ外へ追い出すデータ転送配置最適化を行う．

4.2　混合次元整合分割

整合分割 [3] や多次元整合分割 [5] では，TLG 中のループ群に対して最外側ループ同士より整合判定を行う．そのため，整合可能な次元がループネストの内側に存在する場合には，従来の手法では整合分割を適用できない．

このような場合，依存関係が保ったままループインターチェンジなどを行い，整合可能な次元を最外側に変更することで整合分割を適用することが可能である．

一方で，CNN に代表される深層学習モデルの推論処理では，整合可能な次元を最内側に保持することが望ましい層が存在する．例えば全結合層などでは，整合可能な次元が長くなる特徴があり，ベクトル化に適している場合がある．そこで本稿では，整合可能な次元を最内側に保持したまま整合判定を行い，整合分割を適用する手法を提案する．

本手法では，各 n 重ループ ($n \in \mathbb{N}$) における各次元の組で整合判定を行う．n 重ループ群における整合次元の組はアルゴリズム 1 に従い決定する．本アルゴリズムでは，各 n 重ループより 1 つずつ次元を取り出し，次元の組にて整合判定を行う．その後，整合可能なループ数 (整合数) が最も多い組か，同数の場合は TLG の中で最大コストのループ (標準ループ) の整合次元がより外側となる組が選出される．なお計算量の抑制に向け，配列アクセス範囲情報を用いて整合不可能な組を除外している．

最後に，整合次元の組にて，ストリップマイニングを適用することで整合分割を完了する．

例えば図 4 では Loop1 から Loop3 において，従来では最外側である次元の組 (1, 1, 1) で整合判定が行われるため整合分割

Algorithm 1 n 重ループ群における整合次元の組の選出

Input: 標準ループ $\mathrm{MT}_{\mathrm{std}}$ を含む m 個の n 重ループ
Output: 整合次元の組 T

for $i \leftarrow 1$ **to** m **do**
  n 重ループ i における全次元の集合 R_i を生成
  整合判定対象次元の集合 $S_i \leftarrow \emptyset$
  for all 次元 $r \in R_i$ **do**
    if 配列アクセスが範囲アクセスまたは一点アクセス **then**
      $S_i \leftarrow S_i \cup \{r\}$
    end if
  end for
end for
for all 次元の組 $(s_1, s_2, \ldots, s_m) \in S_1 \times S_2 \times \cdots \times S_m$ **do**
  組に対して整合判定を実行
  if 整合不可能 **then**
    continue
  end if
  if 整合数が最大，または同数かつ $\mathrm{MT}_{\mathrm{std}}$ の次元がより外側 **then**
    $T \leftarrow (s_1, s_2, \ldots, s_m)$
  end if
end for

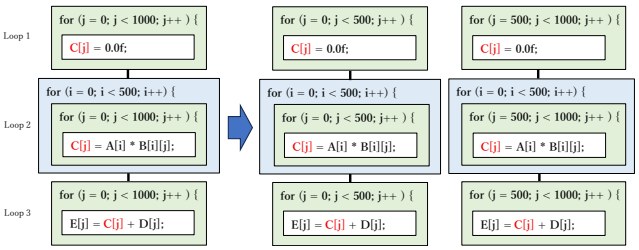

図 4　ループネストの内側を含めた整合分割イメージ

不可であった．これに対し，本稿で提案する整合判定の拡張により整合可能な j ループである次元の組 (1, 2, 1) が選出され，整合分割の適用が可能となる．図では選出された次元の組 (1, 2, 1) にて 2 分割を行った例を示している．

本手法では，ループネストの内側次元も整合次元として選択可能である．しかし内側次元が選択された場合，整合次元より外側に位置する非整合次元のループに対しては分割が行われない．そのため，外側のループの回転数が大きい場合，ローカルメモリ容量の制約により最内側ループを含む整合次元のループの回転数が小さくなる．

一方，ベクトルアクセラレータを最大限に活用するためには，最内側ループの回転数が最大ベクトル長以上であることが望ましい．そこで本稿では，最内側ループの回転数を最大ベクトル長まで確保するため，整合次元より外側のループに対してもストリップマイニングを用いた分割を行う．これにより，ベクトルレジスタ長を有効に活用する．

5.　性 能 評 価

5.1　評 価 方 法

提案手法の評価を，スキップ接続を導入し勾配消失問題を軽減した CNN である ResNet [12] より，ResNet-18 における畳み

込み層と全結合層を用いて行った．ベクトル化には積和計算を含む多重ネストループを入力とし，指示文引数・整合分割による多重ループの分割状況より構成されるヒント情報より対応するベクトル命令列を自動生成し出力するライブラリを OSCAR 自動並列化コンパイラ内に実装し利用した．指示文引数の構成は以下の通りである：

- 積和計算に用いる配列式
- 任意個のループ変数とその回転数の組

畳み込み層と全結合層の積和演算以外の処理に対してはベクトル化を行っていない．

計算に用いるデータ型には FP32 を用い，初期データは主記憶上に配置した．

5.2 評価環境

Xilinx 社の VCU118 FPGA [13] 上に実装した OSCAR ベクトルマルチコアアーキテクチャを使用した．CPU は RISC-V で，LDM の容量は 64KB である．メモリコントローラのチャネル数は 1 である．また VA にはスカラ演算器とベクトル演算器が搭載されており，逐次実行には VA のスカラ演算器を利用した．

ホスト CPU コードのコンパイルには riscv64-unknown-linux-gnu-gcc (GCC) 12.2.0，ベクトルコードのコンパイルには本ベクトルアクセラレータ命令セット用に拡張した Clang/LLVM [14] 16.0.0 を用い，-O2 最適化を利用した．

5.3 畳み込み層の高速化

5.3.1 1 コア実行における最適化

図 5 に畳み込み処理中の積和計算部分における，逐次実行に対するベクトル化実行による速度向上率を示す．図中の畳み込み処理のパラメータは，入出力特徴量の高さ・幅をそれぞれ IH・IW, OH・OW，入出力チャネル数をそれぞれ Cin, Cout，カーネルの高さ・幅を KH, KW とし，(IH×IW, Cin),(KH×KW),(OH×OW, Cout) で表している．

カーネルサイズ方向のループアンローリングにより，特に ResNet-18 モデル全体のボトルネックとなる 7×7 および 3×3 カーネルの畳み込みにおいて性能向上が顕著となる傾向が確認され，39.4–47.7 倍の速度向上が得られた．一方で軽量な 1×1 カーネルの畳み込みでは 29.1–30.8 倍の速度向上が得られた．

また図 6 に，畳み込み処理全体における，逐次実行に対する速度向上率を示す．ベクトル化のみを適用した場合，7×7 および 3×3 カーネルでは 15.2–27.1 倍，1×1 カーネルでは 6.4–7.1 倍の速度向上がそれぞれ得られた．さらにベクトル化に加え，畳み込み処理中のループ不変なデータ転送をループ外へ追い出すデータ転送配置最適化を適用した場合，7×7 および 3×3 カーネルでは 24.1–31.7 倍，1×1 カーネルでは 6.9–8.0 倍の速度向上が得られた．

続いて図 7 に，オーバーラップ転送適用時における，畳み込み処理全体の実行サイクル数に対するベクトル計算の実行サイクル数の割合を示す．オーバーラップ転送を行うことで，7×7 および 3×3 カーネルでは 12.3–34.9%，1×1 カーネルでは 3.8–4.5% 増加し，データ転送待ち時間が削減されている．

また，畳み込み処理全体では図 6 より，計算量の大きな 7×7 および 3×3 カーネルにおいて，オーバーラップ転送前にデータ転送の割合が比較的高い層にて速度向上が大きくなる傾向が確認された．

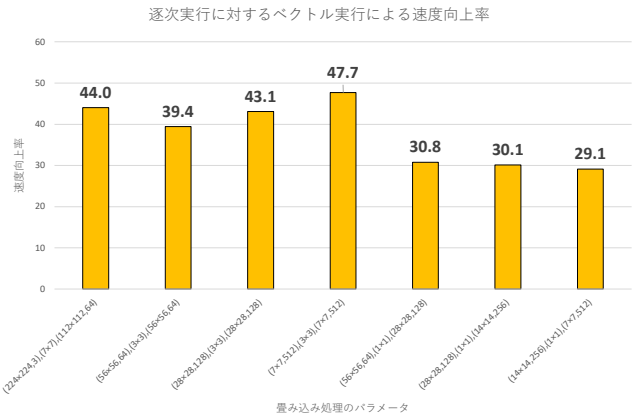

図 5　積和計算部分のベクトル化実行による速度向上 (対逐次実行)

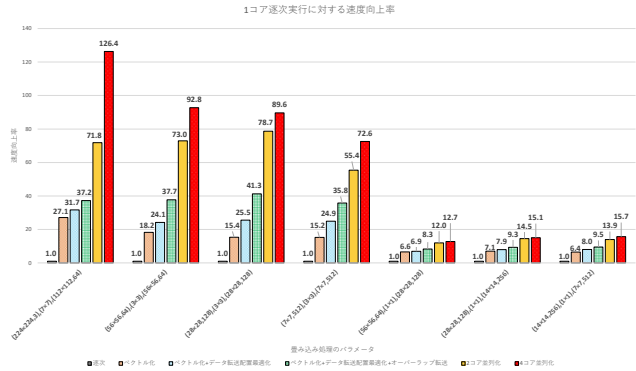

図 6　畳み込み処理全体の逐次に対する速度向上率

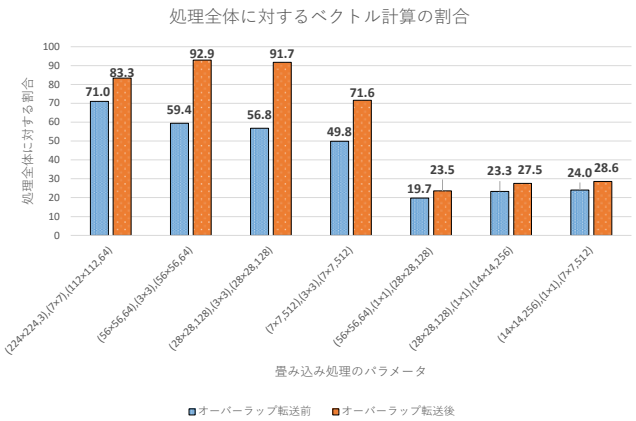

図 7　畳み込み処理全体対するベクトル計算の割合

5.3.2 マルチコア並列化

1 コアにてベクトル化・データ転送配置最適化・オーバーラップ転送を適用したものを，2 コアおよび 4 コア用に並列化した場合，1 コアに対し 7×7 カーネルでは 1.9 倍および 3.4 倍，3×3 カーネルでは 1.6–1.9 倍および 2.0–2.5 倍，1×1 カーネルでは 1.5–1.6 倍および 1.5–1.7 倍の速度向上が得られた (図 6)．

オーバーラップ転送により処理全体に対しベクトル計算が支配的となった層は，メモリコントローラのチャネル数による制約下においても 2 コアまで良好な速度向上が得られる傾向が確

表 1　外側ループの分割に伴うベクトル化による速度向上の変化

整合次元より外側ループの分割	ベクトル化による速度向上率 (対逐次)
無	5.9
有	30.6

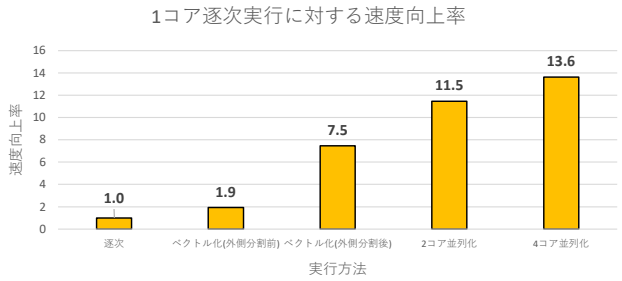

図 8　全結合層全体の逐次に対する速度向上率

認された．一方で処理全体に対しデータ転送が支配的となった層や，4 コアまで並列化した場合は同制約の影響により頭打ちとなった．

1 コアでの逐次実行に対しては，図 6 より，4 コアにてベクトル化・データ転送配置最適化・オーバーラップ転送を実行した場合，7×7 カーネルでは 126.4 倍，3×3 カーネルでは 72.6–92.8 倍，1×1 カーネルでは 12.7–15.7 倍の速度向上が得られた．

5.4　全結合層の高速化

表 1 に全結合層中の積和計算部分における，整合次元より外側のループを分割しベクトル長を確保する際の，逐次実行対するベクトル化実行による速度向上率を示す．ベクトル長を最大まで確保することでベクトル演算パイプラインの利用効率が向上し，逐次実行に対し 30.6 倍の速度向上が得られた．

また図 8 に，全結合層全体における，逐次実行に対する速度向上率を示す．積和計算部分のベクトル化により，全結合層全体において最大 7.5 倍の速度向上が得られた．

さらに図より，1 コアにてベクトル化を適用したものを，2 コアおよび 4 コア用に並列化した場合，1 コアに対し 2 コアで 1.5 倍，4 コアで 1.8 倍の速度向上が得られた．また 1 コアでの逐次実行に対し，4 コアにてベクトル化実行した場合，13.6 倍の速度向上を確認できた．

6.　ま　と　め

本稿では，OSCAR ベクトルマルチコアアーキテクチャを対象に，深層学習推論の主要なボトルネックとなる畳み込み層や全結合層に対し，自動でローカルメモリ管理・オーバーラップ転送・ベクトル化・コア並列化を行う手法を提案し，評価を行った．その結果，1 コアにおける逐次実行に対し 4 コア並列化を行うことで，7×7 カーネルでは 126.4 倍，3×3 カーネルでは 72.6–92.8 倍，1×1 カーネルでは 12.7–15.7 倍の速度向上が得られた．さらに，従来の整合分割を拡張することで全結合層などに対してもベクトル化が可能となり，1 コアにおける逐次実行に対し 4 コア並列化を行うことで 13.6 倍の速度向上が確認された．

謝辞　本研究の成果の一部は JST【ムーンショット型研究開発事業】【JPMJMS2031】の支援を受けたものです．

文　　献

[1] J. Shirako, M. Yoshida, N. Oshiyama, Y. Wada, H. Nakano, H. Shikano, K. Kimura, and H. Kasahara, “Performance Evaluation of Compiler-Controlled Power Saving Scheme,” Proceedings of The 20th ACM International Conference on Supercomputing Workshop on Advanced Low Power Systems (ALPS 2006), p.480–493, July 2006.

[2] T. Hirano, H. Yamamoto, S. Iizuka, K. Muto, T. Goto, T. Wake, H. Mikami, M. Takamura, K. Kimura, and H. Kasahara, “Evaluation of Automatic Power Reduction with OSCAR Compiler on Intel Haswell and ARM Cortex-A9 Multicores,” Proceedings of The 27th International Workshop on Languages and Compilers for Parallel Computing (LCPC), p.239–252, Sept. 2014.

[3] A. Yoshida, K. Koshizuka, and H. Kasahara, “Data-Localization for Fortran Macro-Datafrow Computation Using Partial Static Task Assignment,” Proceedings of The 10th International Conference on Supercomputing, pp.61–68, Jan. 1996.

[4] NVIDIA, “The Future of GPU Computing,” available from (https://www.nvidia.com/content/gtc/documents/sc09_dally.pdf)(accessed on 2026-02-6).

[5] K. Yamamoto, T. Shirakawa, Y. Oki, A. Yoshida, K. Kimura, and H. Kasahara, “Automatic Local Memory Management for Multicores Having Global Address Space,” Proceedings of the 29th International Workshop on Languages and Compilers for Parallel Computing(LCPC), pp.282–296, Sept. 2016.

[6] 大高凌聖，小池穂乃花，磯野立成，川角冬馬，北村俊明，見神広紀，納富　昭，木村貞弘，木村啓二，笠原博徳，“各コアがローカルメモリを持つ組み込みベクトルマルチコアでの畳み込み層演算の評価，” 研究報告システム・アーキテクチャ（ARC），vol.2023，no.32，pp.1–7，March 2023.

[7] K. Kimura, Y. Wada, H. Nakano, T. Kodaka, J. Shirako, K. Ishizaka, and H. Kasahara, “Multigrain parallel processing on compiler cooperative chip multiprocessor,” 9th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT’05), pp.11–20, Feb. 2005.

[8] M. Awaga and H. Takahashi, “The μ VP 64-Bit Vector Coprocessor: A New Implementation of High-Performance Numerical Computation,” IEEE Micro，vol.13，no.5，p.24–36，Sept. 1993.

[9] 末次智貴，野谷優仁，水本幸希，大西文彬，権藤創太，朱　允楷，川角冬馬，北村俊明，笠原博徳，木村啓二，“プログラム実行機構を持つ OSCAR ベクトルマルチコア用データ転送ユニット，” 研究報告システム・アーキテクチャ（ARC），vol.2025，no.33，pp.1–8，March 2025.

[10] H. Kasahara, H. Honda, A. Mogi, A. Ogura, K. Fujiwara, and S. Narita, “A Multi-Grain Parallelizing Compilation Scheme for OSCAR (Optimally Scheduled Advanced Multiprocessor),” Proceedings of the 4th International Workshop on Languages and Compilers for Parallel Computing(LCPC), pp.283–297, Aug. 1991.

[11] F. Onishi, R. Otaka, K. Fujita, T. Suetsugu, T. Kawasumi, T. Kitamura, H. Kasahara, and K. Kimura, “Automatic Deep Learning Parallelization for Vector Multicore Chips with the OSCAR Parallelizing and the TVM Open-Source Deep Learning Compiler,” Proceedings of The 36th International Workshop on Languages and Compilers for Parallel Computing (LCPC), p.96–110, Oct. 2023.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Proceedings of the IEEE conference on computer vision and pattern recognition, pp.770–778, Dec. 2015.

[13] Xilinx, “VCU118 Evaluation Board User Guide (UG1224),” available from (https://docs.amd.com/v/u/en-US/ug1224-vcu118-eval-bd)(accessed on 2026-01-27).

[14] 丸岡　晃，無州祐也，狩野哲史，持山貴司，北村俊明，神谷幸男，高村守幸，木村啓二，笠原博徳，“LLVM を用いたベクトルアクセラレータ用コードのコンパイル手法，” 並列／分散／協調処理に関する『松本』サマー・ワークショップ (SWoPP 松本 2016)，vol.2016，no.4，pp.1–6，2016.