Vol.2018-ARC-230 No.13
Vol.2018-SLDM-183 No.13
Vol.2018-EMB-47 No.13
2018/3/7

OSCARベクトルマルチコアプロセッサのための自動並列ベクトル化コンパイラフレームワーク

宮本 一輝[1]　牧田 哲也[1]　高橋 健[1]　柏俣 智哉[1]　河田 巧[1]　狩野 哲史[1]　北村 俊明[2]　木村 啓二[1]
笠原 博徳[1]

概要：科学技術計算や画像処理，機械学習の分野をはじめとして，アプリケーションの高速化が求められている．これらのアプリケーションの高速化を実現するために各種アクセラレータが利用されている．しかし，アクセラレータを有効に利用するためには，データ配置やホストとアクセラレータ間のデータ転送，同期などを考慮し，アクセラレータ用プログラムを作成する必要があり，プログラマが手動でこれらを行うには開発コストが大きくなる．これに対し，自動並列化やメモリ最適化技術を備える OSCAR 自動並列化コンパイラに自動ベクトル化技術を取り入れ，ベクトルアクセラレータを利用することにより，アプリケーションの高速化，低消費電力化を実現するとともに，開発コストの削減が可能だと考える．本稿では自動車，医療などの組み込みシステムからハイパフォーマンスコンピューティングまで利用できる低消費電力高性能 OSCAR ベクトルマルチコアプロセッサのソフトウェア開発期間を最小化し，誰にでも使いやすい環境を提供するための自動並列ベクトル化コンパイラのフレームワークを提案する．提案フレームワークに基づいて OSCAR コンパイラに拡張実装を行った自動ベクトル化の詳細についても述べる．本フレームワークを利用し，自動生成を行ったベクトル化プログラムに対してベクトルマルチコアシミュレータ上で性能評価を行ったところ，2 つの CPU コアと 2 つのアクセラレータコアで実行した場合，1 つの CPU コアのみによる実行と比べて，行列積では 28.21 倍，2DConvolution では 8.08 倍の性能向上が得られた．FPGA で実装されたベクトルマルチコアエミュレータ上で行列積の性能評価を行ったところ，1 つの CPU コアと 1 つのアクセラレータコアで実行した場合，1 つの CPU コアのみによる実行と比べて，24.91 倍の速度向上が得られた．

1. はじめに

科学技術計算や画像処理，機械学習等の分野をはじめとして，アプリケーションの高速化及び低消費電力化が求められている．これらのアプリケーションの多くは高いデータ並列性を含んでおり，GPU や SIMD 命令拡張などのアクセラレータの利用により高性能化が行われている．しかしアクセラレータを有効に利用するためには，データ配置やホストとアクセラレータ間のデータ転送，同期などを考慮し，アクセラレータ用プログラムを作成する必要があるため，手動プログラミングによる開発コストが大きいことが問題となる．そこでアクセラレータを有効活用しつつソフトウェアの生産性を向上させるために，コンパイラによるアクセラレータ利用プログラムの自動生成化の実現が望まれる．

このような課題を解決するためにアクセラレータを利用するアプリケーションの開発を容易化するコンパイラや開発環境の提案及び開発がされてきた．例えば，NVIDIA の CUDA では各種 GPGPU 向けのライブラリが公開されているが [1], [2]，GPGPU 向けのプログラムを自作する場合，開発者は CUDA 言語を学習する必要があるとともに，データ転送や同期を明示的に記述する必要があり，通常の CPU 用プログラミングを構築するよりもコストがかかる．また，GPU はホストと GPU 間のデータ転送能力がボトルネックとなり，また計算ノードあたりの消費電力が高いという問題がある．

一方，筆者等はこれまで自動並列化やメモリ最適化技術を開発し，これらの最適化機能を OSCAR 自動並列化コンパイラに実現してきた．この OSCAR 自動並列化コンパイラに対して自動ベクトル化技術を取り入れ，ベクトルアクセラレータを利用することにより，アプリケーションの高速化，低消費電力化を実現するとともに，アプリケーショ

[1] 早稲田大学 理工学術院 基幹理工学部 情報理工学科
Dept. of Computer Science and Engineering,
Waseda University.
[2] 早稲田大学 アドバンストマルチコアプロセッサ研究所
Waseda University
Advanced Multicore Processor Research Institute.

ン開発コストの削減を目指す．ベクトルアクセラレータに関するこれまでの取り組みとして，LLVM[3] を拡張しベクトル命令用 Intrinsic 関数を挿入した C ソースコードからベクトルアクセラレータ用のオブジェクトコードを生成するバックエンドコンパイラを開発した [4]．

本稿では，OSCAR コンパイラが対象としてきた OSCAR マルチコアアーキテクチャ [5] にベクトルアクセラレータを付与した，OSCAR ベクトルマルチコアアーキテクチャ用の自動並列ベクトル化コンパイラのフレームワークを提案する．提案手法では，従来の OSCAR コンパイラによる自動並列化やメモリ最適化に加えて，ホストとアクセラレータ間のデータ転送やアクセラレータ制御コードの挿入，及びベクトルアクセラレータ用プログラムの自動ベクトル化を行う．これら最適化の結果はマルチコア CPU 用並列化 C コードとベクトルアクセラレータ用 C コードとして各々生成される．その後，CPU 用バックエンドコンパイラとアクセラレータ用バックエンドコンパイラからそれぞれオブジェクトファイルが生成され，最終的に OSCAR ベクトルマルチコアアーキテクチャ用の実行バイナリとしてリンクされる．本稿では，提案フレームワークによる以上のコンパイルフローと自動ベクトル化機能について詳しく述べる．さらに，実装した自動ベクトル化機能を使用し，ベクトル化した主要カーネルに対して OSCAR ベクトルマルチコアシミュレータおよび FPGA 上で性能評価を行った結果についても報告する．

以下第 2 節では評価対象とする OSCAR ベクトルマルチコアアーキテクチャ及びベクトルアクセラレータアーキテクチャについて，第 3 節では OSCAR ベクトルマルチコアアーキテクチャに対する自動並列ベクトル化コンパイラのフレームワークについて，第 4 節では性能評価について，そして第 5 節ではまとめについて述べる．

2. OSCAR ベクトルマルチコアアーキテクチャ

本節では，提案するコンパイルフレームワークが対象とする OSCAR ベクトルマルチコアアーキテクチャ，及びアクセラレータとなるベクトルアクセラレータについて述べる．

2.1 プロセッサアーキテクチャ

OSCAR ベクトルマルチコアアーキテクチャは OSCAR マルチコアアーキテクチャ [5] をベースとし，各プロセッサエレメント (PE) 内にベクトルアクセラレータ (VA) を付与したアーキテクチャである．各 PE は相互接合網で接続され，プロセッサ外部には各 PE 間の共有データが格納される集中共有メモリ (CSM) が接続される．PE は CPU とローカルデータメモリ (LDM)，分散共有メモリ (DSM)，データ転送ユニット (DTU)，そして VA で構成される．OSCAR ベクトルマルチコアアーキテクチャ図を図 1 に示す．

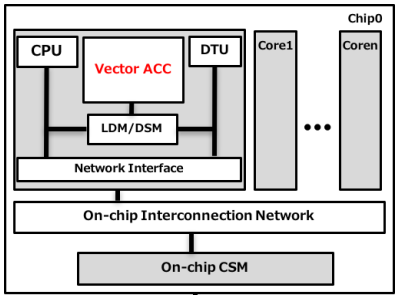

図 1 OSCAR ベクトルマルチコアアーキテクチャ図

LDM は基本的には自 PE 内のみがアクセスできる高速なメモリであり，各 PE のプライベートなデータが格納される．DSM は自 PE と他 PE の両方から同時アクセス可能なメモリであり，タスク間のデータ転送や同期フラグなどの PE 間で授受されるべき共有データが格納される．DTU は CPU，VA と独立にデータ転送を行うことができる DMA コントローラであり，タスク処理とデータ転送がオーバーラップ可能となっている．CSM は LDM や DSM に比べてアクセス時間が長いメモリだが，容量が大きくプログラム及びデータの全てが格納されている．プログラム実行時には DTU あるいは CPU のデータ転送命令によって，タスク処理前に CSM から各 PE の LDM や DSM に転送することにより，高速なメモリアクセスを実現することができる．VA はベクトル演算を搭載したアクセラレータであり，各 PE に搭載され，PE 内の CPU によって起動される．VA は LDM 及び DSM に対してのみアクセスすることができ，CSM に直接アクセスすることはできない．

2.2 ベクトルアクセラレータアーキテクチャ

マルチコア中の各コアが持つベクトルアクセラレータ (VA) は，富士通の VPP シリーズ [6] で開発されてきたベクトルプロセッサをベースにしたアクセラレータであり，ベクトル演算によってデータ並列性の利用できるプログラムの高速・低消費電力処理を目的としている．

本アクセラレータは CPU 非依存に設計されており，任意のプロセッサコアを CPU として使用することが可能となっている．VA にはベクトル演算器及びスカラ演算器が搭載されている．データレジスタはスカラ整数レジスタ (SR)，スカラ浮動小数点レジスタ (FR)，ベクトルレジスタ (VR)，マスクレジスタ (MR) で構成されている．VR の 1 エントリ当たりのサイズは 256Byte であり，8bit データの場合は 256 エレメント，64bit データの場合は 32 エレメントのデータが搭載可能となっている．ベクトルアクセラレータ図を図 2 に示す．

各ベクトル命令は MR を指定することでマスク演算を行うことが可能であり，条件分岐がある場合でも簡単にベク

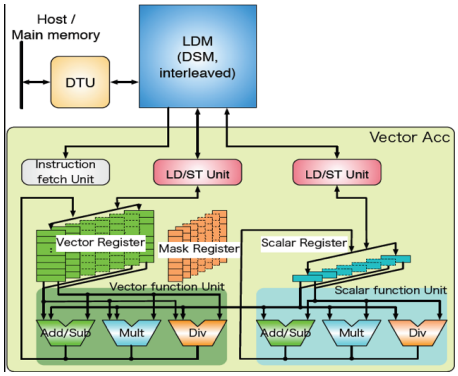

図 2 ベクトルアクセラレータ図

トル化することが可能となっている．また MR のエントリ 0 を指定すると，マスクを使用しないベクトル命令を実行することが出来る．ベクトル長は可変となっており，プログラム中でベクトル長設定命令を実行することによって指定することが可能となっている．ベクトル命令はチェイニングによってベクトル演算器間のパイプライン実行が可能となっている．

本ベクトルアクセラレータは組み込み用途も想定しており，スーパーコンピュータ用の長いベクトル長でも組み込み用の短いベクトル長でも高いスループットを実現することができる．

3. 提案する自動並列ベクトル化コンパイラフレームワーク

本節では OSCAR ベクトルマルチコアアーキテクチャに対する自動並列ベクトル化コンパイラフレームワークについて述べる．このフレームワークでは従来の OSCAR 自動並列化コンパイラ [7] で行ってきた自動並列化やメモリ最適化に加えて，自動ベクトル化，データ配置，ホストとアクセラレータ間のデータ転送，同期制御などを含むアクセラレータ用コード生成および LLVM[3] による VA 用オブジェクトコード生成を行う．また，本節で述べる自動ベクトル化手法に基づいて，OSCAR 自動並列化コンパイラに対して自動ベクトル化機能の拡張実装を行った．

OSCAR ベクトルマルチコアアーキテクチャに対する自動並列ベクトル化コンパイラフレームワークは図 3 のように，OSCAR 自動並列化コンパイラと，ホスト CPU 用ネイティブコンパイラ，及び VA 用ネイティブコンパイラとして使用する Clang/LLVM から構成される．OSCAR 自動並列化コンパイラでは 3.1 節に述べるように，逐次 C ソースコードを入力として自動並列化やメモリ最適化，アクセラレータ制御コードの挿入に加えて自動ベクトル化を行い，ホスト CPU 用並列化 C ソースコードと VA 用ベクトル化 C ソースコードを出力する．ホスト CPU 用コード並列化 C ソースコードは GCC や Clang などのホスト CPU 用ネイティブコンパイラによりコンパイルされ，ホスト CPU 用オブジェクトコードが生成される．VA 用ベクトル化 C ソースコードは 3.2 節に述べるように，VA 用コード生成を行うよう拡張された Clang/LLVM によってコンパイルされ，VA 用オブジェクトコードが生成される．最後にホスト CPU 用のオブジェクトコードと VA 用のオブジェクトコードをリンクすることによって，最終的な並列実行可能バイナリを生成する．

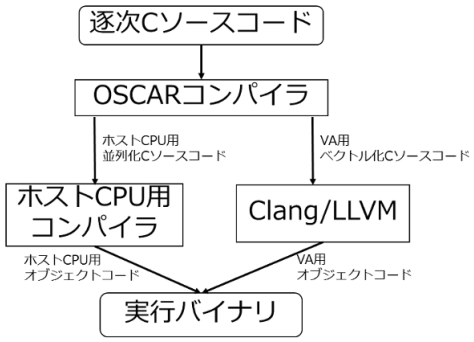

図 3 提案するフレームワーク図 [4]

3.1 OSCAR 自動並列化コンパイラ

OSCAR 自動並列化コンパイラでは逐次用 C ソースコードを入力とし，並列化やメモリ最適化のための解析やリストラクチャリングが行われる．これに加えて，ベクトルアクセラレータ利用への対応として，ベクトル化のための解析やリストラクチャリング，VA 実行部コードの分離，ホストとアクセラレータ間のデータ転送や同期制御コードの挿入などを行う．

3.1.1 自動ベクトル化

自動ベクトル化ではベクトル化のための解析およびコードリストラクチャリングが行われる．以下に，今回実装した自動ベクトル化の手順を述べる．

まず自動ベクトル化解析では最内側ループを対象としてベクトル化可能解析を行う．ベクトル化適用可能条件としてループ間データ依存性を考慮する必要があるため，OSCAR コンパイラの従来機能であるループ並列性解析によって並列化可能なループであるか確認を行う．その後，それぞれの配列添字およびループ誘導変数を解析することにより，配列がメモリに対して連続にアクセスされるか判定する．最内側ループに対して非連続にアクセスする配列が存在する場合，その配列をオペランドとして持つ命令はベクトル化不可能と判定される．また，ループ不変の変数はスカラ変数として扱われる．これらの解析により対象ループ内の命令がデータ連続アクセス，データ依存性なしと判定された場合に，演算子とオペランドのパターンマッチングにより対象アーキテクチャのベクトル命令の取得を行う．ベクトル命令が取得できた時点でベクトル化可能であると判定され，コードリストラクチャリングへと移る．

コードリストラクチャリングでは主に命令のベクトル化，オペランドのベクトル変数への変換，ループストリップマイニング，ループディストリビューションを行う [8]．ベクトル化対象ループ内にベクトル化可能命令とベクトル化不可能命令が混在する場合，ループディストリビューションを行うことで，ベクトル化が可能な命令のみにベクトル化を適用する．また，それぞれのデータ型によってベクトルレジスタ長の最大長が決まっているため，ストリップマイニングを行うことにより演算の粒度を調整し，ベクトル命令およびベクトル変数への変換を可能にする．ループ変形を行った後，ベクトル化可能な命令は Intrinsic 関数への変換を行い，ベクトル化可能なオペランドはベクトル型の変数へと変換される．Intrinsic 関数およびベクトル型は VA 用ヘッダファイルに定義されている．

ベクトル加算を例とした時の入力逐次 C ソースコードイメージを図 4 に，出力ベクトル化 C ソースコードイメージを図 5 にそれぞれ示す．

```
float ta, tb, tc;

for (i = 0; i <= size; i++) {
      ta = a[i];
      tb = b[i];
      tc = ta + tb;

      c[i] = tc;
  }
```

図 4 入力逐次 C ソースコード

```
__pvf vec_ta, vec_tb, vec_tc;
int vr_length = 64;

new_size = size - (size+1) % vr_length;
_pt_vlvl(vr_length);
for (i = 0 ; i <= new_max ; i += vr_length){
    vec_ta = _pt_vld_f(a+i);
    vec_tb = _pt_vld_f(b+i);
    vec_tc = _pt_vadd_f(vec_ta,vec_tb);
    _pt_vst_f(c+i,vec_tc);
    }
    vec_size = size-new_size;
    _pt_vlvl(vec_length);
    vec_ta = _pt_vld_f(a+i);
    vec_tb = _pt_vld_f(b+i);
    vec_tc = _pt_vadd_f(vec_ta,vec_tb);
    _pt_vst_f(c+i,vec_tc);
```

図 5 出力ベクトル化 C ソースコード

図 5 のようにそれぞれの命令が Intrinsic 関数に変換されており，例えばベクトルロード命令は_pt_vld()，ベクトルストア命令は_pv_vst() として表現されている．_pv_vlvl() はベクトル長設定であり，任意のベクトル長を設定することができる．図 5 では float 型の最大ベクトル長である 64 として設定している．また，この例はループ上限値が変数であるため，解析時点ではループストリップマイニング後に端数が出るかわからない．そのため，明示的にベクトル化ループ後に端数処理が行われている．端数処理は上記のベクトル長設定命令を利用することにより実現している．

3.2 Clang/LLVM

VA のネイティブコンパイラとして，LLVM バックエンドに VA のターゲットを拡張した Clang/LLVM を使用する [4]．Clang/LLVM では OSCAR コンパイラによって自動ベクトル化されたベクトル化 C ソースコードを入力として，VA のオブジェクトコードを生成する．

Clang/LLVM における VA 用ベクトル化 C ソースコードのコンパイル方法の概要を説明する．ベクトル化 C ソースコードを入力として，フロントエンドの Clang[9] によって LLVM の中間表現となる LLVM-IR に変換される．LLVM-IR においては，ベクトル化 C ソースコードにおけるベクトル型の変数は VectorType として表現される．各ベクトル演算に関しては，基本演算かつマスク無しのベクトル演算の場合はベクトル型をオペランドにした命令として，また複雑な演算やマスク有りの演算の場合は Builtin 関数に対応した LLVM-IR Intrinsic 関数の呼び出しとして，それぞれ表現される．

4. 性能評価

本節では提案するコンパイルフレームワークで自動生成した計算カーネルの性能を OSCAR ベクトルマルチコアシミュレータおよび FPGA に実装されている OSCAR ベクトルマルチコアエミュレータ上で評価した結果について述べる．

4.1 OSCAR ベクトルマルチコアシミュレータでの評価

4.1.1 評価環境

本評価で使用した OSCAR ベクトルマルチコアシミュレータの構成を表 1 に示す．

CPU には SPARC V9 命令セットのプロセッサを使用している．

VA はスカラ命令に関しては加減算ユニットと乗算ユニットがそれぞれ 1 本ずつ，ロードストアユニットが 1 本，ベクトル命令に関しては加減算ユニットと乗算ユニットがそれぞれ 1 本ずつ，ロード・ストアユニットが 2 本存在し，単一発行の構成となっている．各種ベクトル演算器は 256 ビットの演算幅を持っており，例えば 32 ビット単精度浮動小数点のベクトル演算では一度に 8 要素の演算が可能である．また，ロード・ストアユニットはベクトルロード・ストア命令の場合，LDM に対して一度に 8 バイト ×4 要素のリクエストを発行可能である．ベクトル演算ユニット及びロードストアユニットはチェイニングによるベクトル命令間のパイプライン実行が可能となっている．

各メモリのレイテンシは組み込み用途を意識し，LDM

Vol.2018-ARC-230 No.13
Vol.2018-SLDM-183 No.13
Vol.2018-EMB-47 No.13
2018/3/7

とが 2 クロックサイクル，DSM が 1 クロックサイクル，CSM が 60 クロックサイクルとなっている．LDM は複数バンクから構成されており，本評価におけるバンク数は 8 とした．各メモリはバースト転送が可能であり，例えば LDM からベクトルロードを行う場合，2 クロックサイクルのレイテンシの後は 1 サイクルごとに 1 バンクあたり 8 バイトずつロード可能である．

上記評価環境のもとで評価対象の計算カーネルを GCC でコンパイルし 1 つの CPU コアのみで実行した場合と，3 節で述べたフレームワークでコンパイルし，自動ベクトル化を適用した計算カーネルを 1CPU コアと 1VA コアにおいて実行した場合と 2CPU コアと 2VA コアにおいて実行した場合の性能を比較する．本評価では，CPU コア用バックエンドコンパイラとして gcc 4.7.2 を，アクセラレータ用バックエンドコンパイラとして Clang/LLVM 4.7.2 をベースに 3.1.1 節で述べた拡張を行った物をそれぞれ用いた．最適化オプションには共に-O2 を用いた．

表 1 OSCAR ベクトルマルチコアシミュレータの構成

CPU	Instruction Set	SPARC V9
	L1 Cache Size	32KB
	L2 Cache Size	512KB
VA	Scalar Int/FP ADD/SUB Unit	1
	Scalar Int/FP MUL Unit	1
	Scalar LOAD/STORE Unit	1
	Vector Int/FP ADD/SUB Unit	1
	Vector Int/FP MUL Unit	1
	Vector LOAD/STORE Unit	2
Memory	LDM Latency	2 clock cycle
	DSM Latency	1 clock cycle
	CSM Latency	60 clock cycle

4.1.2 評価プログラム

評価プログラムとして，科学技術計算や画像処理で頻出する計算カーネルである行列積と 2DConvolution を使用する．各評価プログラムのパラメータを表 2 に示す．

行列積では入力及び出力配列のサイズは 64x64，データ型は単精度浮動小数点型としている．2DConvolution では入力及び出力配列のサイズは 64x64，カーネルサイズは 3x3，データ型は単精度浮動小数点型としている．初期状態ではデータは CSM 領域に配置されている状態から評価を行い，VA を評価する場合は DTU もしくは CPU を用いて LDM にデータ転送を行い，性能評価を行う．

表 2 評価プログラムのパラメータ

Matmul	Data Size	64x64
	Data Type	32bit Floating-point
2DConv	Data Size	64x64
	Kernel Size	3x3
	Data Type	32bit Floating-point

4.1.3 評価結果

OSCAR ベクトルマルチコアシミュレータ上で行列積を動作させた場合の性能評価結果を図 6 に，2DConvolution を動作させた場合の性能評価結果を図 7 にそれぞれ示す．図中，グラフの左縦軸は実行クロックサイクル数を示しており，また右縦軸は 1CPU のみの実行での実行クロックサイクル数を 1 とした場合の速度向上率である．

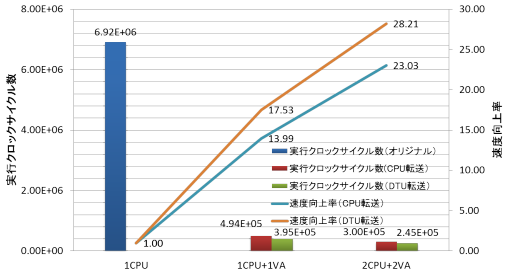

図 6 シミュレータ上での行列積の評価結果

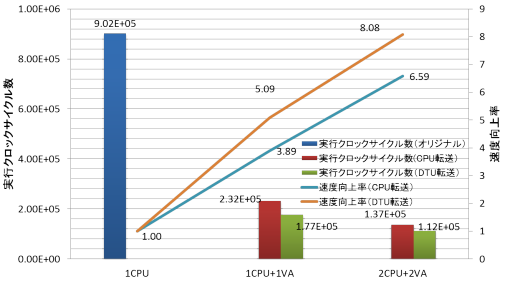

図 7 シミュレータ上での 2DConvolution の評価結果

今回の評価では行列積および 2DConvolution ともに自動ベクトル化後の演算の正当性を確認できており，適切なベクトル化が行われたことが示せた．

図 6 より，1CPU 単体の実行に比べ，1CPU+1VA+CPU 転送では 13.99 倍，1CPU+1VA+DTU 転送では 17.53 倍，2CPU+2VA+CPU 転送では 23.03 倍，2CPU+2VA+DTU 転送では 28.21 倍の速度向上率が得られた．また，1CPU+1VA+CPU 転送の実行に比べて，1CPU+1VA+DTU 転送は 1.25 倍，2CPU+2VA+CPU 転送の実行に比べて，2CPU+2VA+DTU 転送は 1.22 倍の速度向上率が得られた．図 7 より，1CPU 単体の実行に比べ，1CPU+1VA+CPU 転送では 3.89 倍，1CPU+1VA+DTU 転送では 5.09 倍，2CPU+2VA+CPU 転送では 6.59 倍，2CPU+2VA+DTU 転送では 8.08 倍の速度向上率が得られた．また，1CPU+1VA+CPU 転送の実行に比べて，1CPU+1VA+DTU 転送は 1.31 倍，2CPU+2VA+CPU 転送の実行に比べて，2CPU+2VA+DTU 転送は 1.23 倍の速度向上率が得られた．

これらの結果により，VA におけるベクトル実行によってプログラムの性能向上が可能であるとともに，DTU を

用いたデータ転送の有用性が確認できた．さらに逐次 C ソースコードから OSCAR 自動並列ベクトルコンパイラフレームワークにより，VA 用ベクトル化 C ソースコードおよびオブジェクトコードの生成が可能であることが確認できた．

4.2 FPGA 上の OSCAR ベクトルマルチコアエミュレータでの評価

4.2.1 評価環境

本評価では OSCAR ベクトルマルチコアアーキテクチャを FPGA 上に実装されているエミュレータを使用した．本エミュレータは Arria10 SoC Development kit[10] 上に 1CPU コア+1VA コアの構成で構築した．CPU コアには NIOS II を使用し，命令キャッシュ，データキャッシュ共に 64KB である．動作周波数は 100MHz である．

4.2.2 評価プログラム

評価プログラムとして行列積を使用した．パラメータは表 2 と同じである．

行列積では入力及び出力配列のサイズは 64x64，データ型は単精度浮動小数点型としている．あらかじめデータは LDM 領域に配置されている状態から評価を行った．

4.2.3 評価結果

OSCAR ベクトルマルチコアエミュレータ上で行列積を動作させた場合の性能評価結果を図 8 に示す．図中，グラフの左縦軸は実行時間を示しており，また右縦軸は 1CPU のみの実行での実行時間を 1 とした場合の速度向上率である．

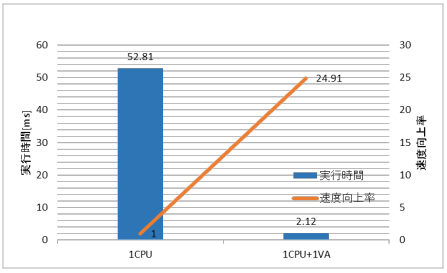

図 8 エミュレータ上での行列積の評価結果

図 8 より，1CPU 単体の実行に比べ，1CPU+1VA では 24.91 倍の速度向上率が得られた．

5. まとめ

本稿では組み込みシステムからハイパフォーマンスコンピューティングまで利用できる低消費電力高性能 OSCAR ベクトルマルチコアプロセッサの自動並列ベクトル化コンパイラのフレームワークを提案した．本フレームワークでは，OSCAR 自動並列化コンパイラにおいて自動並列化やメモリ最適化に加えて自動ベクトル化，アクセラレータの制御やホストとアクセラレータ間のデータ転送の自動挿入を行う．さらに LLVM を用いて，OSCAR コンパイラによって生成されたベクトル化 C ソースコードからベクトルアクセラレータのオブジェクトコードを生成する．

本手法のうち OSCAR 自動並列化コンパイラに対して，自動ベクトル化およびアクセラレータの制御コードの自動挿入部を拡張実装し，プログラムをコンパイルし OSCAR ベクトルマルチコアシミュレータ上で評価を行った結果，CPU 単体実行と 2CPU+2VA+DTU 転送の総クロックサイクル数を比較して，行列積では 28.21 倍，2DConvolution では 8.08 倍の性能向上が得られた．FPGA に構築した OSCAR ベクトルマルチコアエミュレータ上で行列積の評価を行った結果，CPU 単体実行と 1CPU+1VA の実行時間を比較して，24.91 倍の速度向上が得られた．以上より，本手法によって逐次 C ソースコードからベクトルアクセラレータ用ベクトル化 C ソースコードおよびオブジェクトコードが生成可能であることが確認された．

謝辞

本研究の一部は科研費基盤研究 (C)15K00085 の助成により行われた．

参考文献

[1] NVIDIACorporation: *CUDA Zone* (2018).
[2] Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., Segal, M., Papakipos, M. and Buck, I.: GPGPU: General-purpose computation on graphics hardware, *SC '06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing Article* (2006).
[3] llvm.org: *The LLVM Compiler Infrastructure* (2018).
[4] 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，情報処理学会研究報告，Vol. 2016-ARC-221, No. 4, pp. 1–6 (2016).
[5] Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., Ishizaka, K. and Kasahara, H.: Multigrain Parallel Processing on Compiler Cooperative Chip Multiprocessor, *Proc. of 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9)* (2005).
[6] Miura, K., Takamura, M., Sakamoto, Y. and Okada, S.: Overview of the Fujitsu VPP500 supercomputer, *Compcon Spring '93, Digest of Papers.* (1993).
[7] Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K. and Narita, S.: A multi-grain parallelizing compilation scheme for OSCAR (optimally scheduled advanced multiprocessor), *Fourth International Workshop Santa Clara* (1991).
[8] Aho, A. V., Lam, M. S., Sethi, R. and Ullman, J. D.: *Compilers: Principles, Techniques, and Tools (2nd Edition)*, Addison Wesley (2006).
[9] llvm.org: *clang: a C language family frontend for LLVM* (2018).
[10] IntelCorporation: *Arria 10 SoC Development Kit User Guide* (2018).