Vol.2021-ARC-244 No.20
Vol.2021-SLDM-194 No.20
Vol.2021-EMB-56 No.20
2021/3/25

OSCARコンパイラによるMATLAB/Simulinkアプリケーションの自動並列化

古山 凌[1,a)]　津村 雄太[1]　川角 冬馬[1]　仲田 優哉[1]　梅田 弾[1]　木村 啓二[1]　笠原 博徳[1]

概要：組込み向けシステム開発には,MATLAB/Simulink のようなツールが幅広く利用されており，設計から実装・検証を一貫して行うことができる．このような開発では，Embedded Coder や MATLAB Coder のような組み込みシステム上に実装する自動 C コード生成がサポートされているが，マルチコアプロセッサによる並列処理を十分に活用できていない．従来までに, Simulink モデル内のブロック間並列性を抽出する手法が提案されているが, ブロック内部の並列性や MATLAB 言語で記述されたカスタムブ ロック内の並列性を抽出していない. これに対して, MATLAB/Simulink から自動生成された C コードに対して, モデル内のブロック間並列性やブロック内部のループ並列性を抽出する OSCAR 自動並列化コンパイラを用いたマルチグレイン並列化手法が提案されている．しかしながら，コードレベルでの並列化ではコンパイラの解析機能に依存し，自動生成されたコード特有の冗長的な構造により，コンパイラによる解析が難しい場合が存在した．そこで，従来の OSCAR 自動並列化コンパイラの解析機能に加えて，シンボリック伝搬とループインターチェンジ及び動的メモリ確保解析機能を拡張し，より多くのアプリケーションで自動並列化可能にした．本稿では OSCAR 自動並列化コンパイラにより MATLAB/Simulink アプリケーションを並列化した結果，Intel Xeon X5670 での 4 コア実行時，画像勾配計算では 1.88 倍，二次元台座エッジの抽出では 1.79 倍，時系列データの移動平均計算では 5.04 倍の速度向上が得られた．また PSNR では 3.62 倍の速度向上が得られた．さらに画像回転では 2.63 倍，緑色検出では 3.48 倍の速度向上が得られた．同様に，上記のアプリケーションに対して Cortex-A72 を 4 コア持つ Raspberry Pi4 上での速度向上を確認した．これらの結果から OSCAR 自動並列化コンパイラによる MATLAB/Simulink アプリケーションの並列性抽出が有効であることを確認した．

1. はじめに

モデルベース開発ではツール上にて設計から実装や検証，コード生成までを一貫して行うことが可能であり，組み込みシステムの開発で有用となる．自動車や航空機，医用機器等のようなリアルタイムでの動作を前提とし，安全性が特に重要視される分野では従来の仕様書からシステム全体をコーディングする開発手法から MATLAB/Simulink [1] によるモデルベース開発手法に移行しつつある [2].

このようなモデルベース開発ではモデルからコードの自動生成をサポートしており，システムの安全性や開発効率性が大きく向上が可能となっている．しかしながら，現時点での自動コード生成ツールはマルチコアプロセッサ向けに出力コードを最適化するまでに至っていない．そのため，ハードウェア性能を十分に活用するためには自動生成されたコードの並列化が必要である．

[1] 早稲田大学
Waseda University.
[a)] ryo_k@kasahara.cs.waseda.ac.jp

このような問題を解決する為に MATLAB/Simulink アプリケーションを並列化するツールとして MathWorks 社が開発している Parallel Computing ToolBox[3] が挙げられる．これには並列化対象のループに対して開発者が指示文を挿入するもの [4] や，GPU 上での計算を配列宣言時に指示するもの [5] などが用意されている．しかし，これらは開発されたシステムの並列処理性能が開発者の能力に依存してしまうことや，大規模なアプリケーションに対しても綿密なチューニングが必要となってしまうことからシステム開発の効率が大幅に低下してしまうといった問題がある．一方，Parallel Computing ToolBox の中でも一部の MATLAB 関数の自動並列化 [6] をサポートしているが，その並列性抽出は，本来のアルゴリズムがもつ理論的な並列性能に比較して十分ではない場合がある．また，これは関数内のみの並列性抽出にとどまってしまう為，より大きな粒度での並列化が必要な場合に対応できないことが問題である．

既存の MATLAB/Simulink の並列化に関する研究とし

Vol.2021-ARC-244 No.20
Vol.2021-SLDM-194 No.20
Vol.2021-EMB-56 No.20
2021/3/25

てモデルブロック間の並列性を抽出するものが提案されている．しかし，このような手法ではモデルブロック内部の並列性を抽出する必要がある場合に対応できない．特に MATLAB/Simulink では機能単位で構成されるモデルブロックに加えて MATLAB 言語で記述されたカスタムブロックを用いた開発が可能となっているが，モデルレベルの並列化手法ではこのようなカスタムブロック内の並列性を抽出することが困難であり，アプリケーションがもつ本来の並列性を十分に引き出すことができない．

これに対して, MATLAB/Simulink から自動生成された C コードに対して, モデル内のブロック間並列性やブロック内部のループ並列性を 抽出する OSCAR 自動並列化コンパイラを用いたマルチグレイン並列化手法が提案されている [7]. コードレベルでの並列化ではコード全域の解析を行うことから，モデルレベルの並列化に比較して様々な粒度の並列性を抽出することができる．しかしその並列性解析能力はコンパイラの解析機能に依存するため，自動生成されたコード特有の冗長的な構造により，コンパイラによる解析が難しい場合が存在した．

本稿では，従来の OSCAR 自動並列化コンパイラの解析機能に加えて，シンボリック伝搬 [8] とループインターチェンジ機能 [9] 及び動的メモリ確保解析機能を拡張し，より多くのアプリケーションで自動並列化可能にした結果について報告する．さらに，解析機能を拡張した OSCAR 自動並列化コンパイラにより MATLAB/Simulink アプリケーションを並列化し，Cortex-A72 を 4 コアもつ Raspberry Pi4 と Intel Xeon X5670 により評価した結果についても報告する。

以下本稿では，2 節で関連研究を概観する．3 節で MATLAB/Simulink から自動生成された C コードの並列化手法，4 節で OSCAR 自動並列化コンパイラの概要を，それぞれ説明する。5 節で性能評価について報告し，6 節でまとめる．

2. 関連研究

MATLAB/Simulink アプリケーションの並列処理要求に対して，自動で並列化を行う手法がいくつか提案されている．Simulink モデルでは，そのモデル上に現れるブロック結線をブロック間の依存関係と見なすことができ，これを利用したブロック間の並列性解析が可能である．これを踏まえて，モデルベース開発アプリケーションの並列化に関連した研究ではモデル上に現れるモデルブロック間の並列性を利用した研究が提案されている．例えば Zhong 等の Simulink モデルの並列化手法 [10] では入力された Simulink モデルのモデルブロックのそれぞれを一つのノードとして見立て，有向非巡回グラフとして表現することで並列性を抽出し，モデルレベルの並列化を実現している．また久村等の Simulink モデルの並列化手法 [11][12] では Simulink モデルのブロック結線構造を依存関係と見立てた CSP 理論に基づいたモデルレベルの並列化を行っている．また Cha 等の並列化手法 [13] では Simulink モデルのブロック間のデータ依存情報を利用することでサブシステム及び関数レベルの並列性抽出を行っている．

一方，OSCAR 自動並列化コンパイラを用いた MATLAB/Simulink アプリケーションの並列化手法 [7] では，Simulink モデルの構造に縛られず，自動生成された C コードの全域の解析を行うことで大きな粒度から，きめ細やかな粒度の並列化を実現している．ただし，コードレベルでの並列化性能はコンパイラの解析能力に依存してしまう．MATLAB/Simulink が自動生成するコードは特有の冗長な構造をもつことや，複雑な動的メモリ確保を行うと言った特徴があり，コンパイラによる解析が困難な場合が存在した．本稿では，従来の OSCAR 自動並列化コンパイラの解析機能に加えて，シンボリック伝搬とループインターチェンジ機能，及び動的メモリ確保解析を拡張し，より広範なアプリケーションでの自動並列化を可能にした．

3. MATLAB/Simulink から自動生成された C コードの並列化手法

本節では MATLAB/Simulink から自動生成されたコードの特徴，及びそれらの特徴を踏まえたコンパイラによる並列性抽出の手法であるシンボリック伝搬，ループインターチェンジ，及び動的メモリ確保解析について述べる．

3.1 自動生成された C コードの特徴と並列性抽出手法

MATLAB Coder から生成された C コードの多重ループの例を**図 1** に示す．

- 配列添字がループインデックスで表現されていない

```
1  for (i = 0; i < 3; i++) {
2      temp = kernel[i];
3      for (j = 0; j < 1325; j++) {
4          if (!(temp == 0.0)) {
5              temp_x = (j * 984 - k) + 2;
6              temp_y = j * 982;
7              for (k = 0; k < 982; k++) {
8                output[temp_y] += kernel[temp]
                     * pict[temp_x];
9                temp_x++;
10               temp_y++;
11             }
12           }
13       }
14 }
```

図 1 自動生成された多重ループの例
Fig. 1 Example of Nested loops generated by MATLAB Coder

Vol.2021-ARC-244 No.20
Vol.2021-SLDM-194 No.20
Vol.2021-EMB-56 No.20
2021/3/25

- 冗長な一時変数への代入文を含む
- 最外側ループの回転数が小さい場合がある

といった特徴があり，これらはコンパイラによるループ並列性抽出を困難にする．例えば図 1 の 8 行目の output 配列の添字である temp_y はループインデックスではない．また，2 行目，5 行目及び 6 行目の一時変数への代入文は全て冗長な文である．さらに 1 行目のループが示すイタレーション数は 3 回転のみである．

また，MATLAB アプリケーションは型情報の解析が困難な動的メモリ確保を行う特徴がある．コンパイラによる並列性解析を行うためには，プログラム中のポインタの指し先箇所の解析が可能でなければならない．しかし，MATLAB アプリケーションでは，動的メモリ確保された構造体メンバがキャストされ，型情報の解析が困難であるといった特徴がある．

以上に述べた多重ループの特徴や，動的メモリ確保の特徴はコンパイラによる並列性を解析を困難にする．次に解析を困難にする理由の詳細，及びそれらを解決するための機能であるシンボリック伝搬とループインターチェンジ，及び動的メモリ確保解析について述べる．

3.1.1 シンボリック伝搬

コンパイラによるループイタレーションレベルの並列性解析時にはループボディにて定義される配列添字がループインデックスによって表現されていることが必要条件である．しかし，前述したように自動生成された C コードは多重ループ内に複数の冗長文を含み配列添字が一時変数で表現されていることが特徴であり，ループ並列性解析が困難である．これを解決するために図 1 に対してシンボリック伝搬を適用した後のコードを**図 2** に示す．シンボリック伝搬を適用することで図 1 に含まれていた冗長文の削除，及び一時変数の伝搬を行い，配列添字をループインデックスを用いた線形式で表現できたことが分かる．この結果，コンパイラによる並列性解析が可能になる．

```
1  for (i = 0; i < 3; i++) {
2      for (j = 0; j < 1325; j++) {
3          if (!(kernel[i] == 0.0)) {
4              for (k = 0; k < 982; k++) {
5                output[j * 982 + k] +=
6                  kernel[i] *
7                  pict[(j * 984 - i) + 2 + k];
8              }
9            }
10       }
11   }
```

図 2 自動生成された多重ループに対するシンボリック伝搬の適用例
Fig. 2 Example of application of symbolic range propagation to automatically generated nested loops

3.1.2 ループインターチェンジ

ループイタレーションレベルの並列性を抽出する際には並列化可能ループのうち，最外側ループをイタレーション単位でマルチコアに割り当てる．最外側ループの回転数が十分に大きくない場合，ループを均等に分割してマルチコアに割り当てることができない．よって多重ループが持つ並列性を最大限に活用するためには，回転数の大きいループを外側にインターチェンジする必要がある．しかし MATLAB/Simulink から自動生成される図 1 のような最内側ループボディ以外にも代入文を含む多重ループは依存関係を壊してしまうためにループインターチェンジが適用できない．シンボリック伝搬を適用し冗長文を削除した図 2 では完全ネストループに変形することができたため，ループインターチェンジの適用が可能となっている．この結果ループインターチェンジ適用前と比較して，より大きなループ並列性の抽出が可能となる．

3.1.3 動的メモリ確保解析

MATLAB 言語において配列は，その要素数を明示的に記述せずに宣言することが可能である．このため，多くの MATLAB アプリケーションはコード中で動的メモリ確保を行う．同様に，それらから自動生成された C コードも動的メモリ確保を行うコードが含まれている．しかし，コンパイラによる並列性解析を行うためには，ポインタの指し先箇所の解析が可能でなければならない．

次に MATLAB が生成する構造体の例，及び動的メモリ確保の例をそれぞれ**図 3** と**図 4** に示す．

自動生成された動的メモリ確保を行うコードでは図 3 の 1 行目から 4 行目のようなベースとなる構造体が宣言される．加えてそれぞれのアプリケーションの構成に応じて 5 行目から 8 行目，また 9 行目から 12 行目のような複数の構造体が宣言される．これら複数の構造体に対して，ベースとなる構造体へのキャストを行うことで，図 4 の 1 行目から 7 行目に示す動的メモリ確保関数を共通化して用いる

```
1  struct arrayBase {
2      void *data;
3      int size;
4  };
5  struct arrayStruct1 {
6     struct struct1 *data;
7     int size;
8  };
9  struct arrayStruct2 {
10    struct struct2 *data;
11    int size;
12 };
```

図 3 自動生成された構造体の例
Fig. 3 Example of an automatically generated structure

Vol.2021-ARC-244 No.20
Vol.2021-SLDM-194 No.20
Vol.2021-EMB-56 No.20
2021/3/25

```
1  void allocArrayData
2  (struct arrayBase *arrayst, int num
3      , int elemsize)
4  {
5      arrayst->data = malloc(num*elemsize);
6      arrayst->size = num;
7  }
8
9  struct arrayStruct1 st1;
10 allocArrayData((struct arrayBase*)(&st1)
11     , num, elemsize);
```

図 4 自動生成された動的メモリ確保の例
Fig. 4 Example of an automatically generated structure

ことができる．

従来の OSCAR 自動並列化コンパイラでは上記のような共通化された動的メモリ確保関数において，メモリ確保を行ったポインタを保持する構造体が異なる場合，その解析が出来ない．

本稿では，上記のような動的メモリ確保に対して，共通化された動的メモリ確保関数において，メモリ確保を行ったポインタを保持する構造体が異なる場合においても構造体中のポインタを保持するメンバが同一であれば，そのポインタメンバが指すヒープ領域を適切に解析できるように拡張した．これにより，動的メモリ確保を行う多くの MATLAB/Simulink アプリケーションの解析が可能になった．

4. OSCAR 自動並列化コンパイラ

本稿ではシンボリック伝搬，ループインターチェンジ，及び動的メモリ確保解析機能を OSCAR 自動並列化コンパイラに拡張し，MATLAB/Simulink アプリケーションに対して適用し評価を行う．本節では OSCAR 自動並列化コンパイラ及び OSCAR 自動並列化コンパイラがもつマルチグレイン並列化手法の概要について述べる．

4.1 OSCAR 自動並列化コンパイラの概要

OSCAR 自動並列化コンパイラとは入力された C コード，または Fortran コードの並列性を自動で解析，及び抽出し，メモリ最適化を行った並列化済みプログラムを出力する Source-to-source コンパイラである [14].

次に OSCAR 自動並列化コンパイラがもつマルチグレイン並列化手法について述べる．マルチグレイン並列化手法とは以下の 3 種類の並列化手法を階層的に適用することで様々な粒度の並列性を抽出する手法である．

- 粗粒度並列化: 基本ブロック，ループブロック，関数呼び出しブロック間の並列化
- 中粒度並列化: ループイタレーションレベルの並列化
- 近細粒度並列化: 基本ブロック内のステートメントレベルの並列化

OSCAR 自動並列化コンパイラはマルチグレイン並列化手法により，プログラム全域の並列性を抽出することが可能である．これにより MATLAB/Simulink アプリケーションに対しても，MATLAB で記述されたアプリケーションや Simulink のモデルブロック構成に影響を受けずに，コード全域の並列性を抽出することが可能である．

表 1 評価アプリケーション一覧
Table 1 Evaluated Applications

アプリケーション名	入力サイズ
画素間勾配計算	1325 × 982 × 3
二次元台座エッジの抽出	1920 × 1080
時系列データの移動平均	1826 × 1000
PSNR	391 × 1000
画像回転	165 × 192
緑色検出	384 × 512 × 3

5. 性能評価

本節では評価対象であるアプリケーション群について述べる．評価には Arm Cortex A72 コアを持つ Raspberry Pi4（4 コア）の SMP マシンと，Intel Xeon X5670（6 コア）プロセッサを用いた．MATLAB/Simulink アプリケーションから得られた C プログラムを OSCAR コンパイラで並列化し，これらの評価マシン上で評価した結果を報告する．

5.1 評価対象アプリケーション

本評価では MATLAB/Simulink で開発された 6 つのアプリケーションの並列処理性能評価を行う．全てのアプリケーションにおいて，組込みプロセッサ上での動作を想定し，入力データを float 型にしてコード生成を行った．表 1 にアプリケーション一覧を示す．

5.1.1 画素間勾配計算

画素間勾配計算 [15] は畳み込み演算によりピクセル間の勾配計算を行う．x 方向の勾配，及び y 方向の勾配をそれぞれ計算する際に畳み込み演算を行う．畳み込み演算を行う多重ループに対してシンボリック伝搬を適用することで冗長文の削除，及び配列添字の変換を行い，コンパイラによるループレベル並列性抽出が可能となった．また，ループインターチェンジによりさらなるループ並列性抽出をおこなった．これにより畳み込み演算や配列の初期化を含めて実行時間の約 99%が並列化可能ループとなった．評価に用いた入力画像サイズは 1325×982×3 である．

5.1.2 二次元台座エッジの抽出

二次元台座エッジの抽出 [16] では畳み込み演算による

エッジ検出を行う．本評価では 5.1.1 節の画素間勾配計算アプリケーションと同様にシンボリック伝搬とループインターチェンジによる並列性抽出を行ったことにより約 99%が並列化可能ループとなった．評価に用いた入力画像サイズは 1920×1080 である．

5.1.3 時系列データの移動平均計算

時系列データの移動平均計算では，入力データに対して畳み込み演算を用いて 7 日間移動平均を計算する．本評価では 5.1.1 節の画素間勾配計算アプリケーションと同様にシンボリック伝搬とループインターチェンジによる並列性抽出を行ったことにより約 99%が並列化可能ループとなった．評価に用いた入力データサイズは 1826 × 1000 である．

5.1.4 ピーク信号対雑音比の計算 (PSNR)

ピーク信号対雑音比 [17] の計算について述べる．PSNR は画像のエンコーダ性能の評価に用いられ，エンコード後の画質劣化の評価指数として利用される．この関数では 2 つのエンコーダの入力画像と出力画像の間の画素値の平均的な変化を求める．以下の式 1 に PSNR を求める式を示す．

$$PSNR = 10\log_{10}(R^2/MSE) \quad (1)$$

R は画素値の最大値，MSE はエンコード前の画像とエンコード後の画像間の累積二乗誤差を示す．

本アプリケーションは 99%程度が並列処理可能である．評価に用いた入力画像サイズ，及び出力画像サイズは 391×660×3 である．

5.1.5 画像回転

画像回転 [18] では，双一次内挿法を用いて画像を回転させる．具体的にはグリッド空間へ変換した入力画像と回転用のカーネルの積を計算する．本アプリケーションは動的メモリ確保解析を行うことで並列性の抽出が可能となり，実行時間の 81.2%を占める並列化可能ループに対してループイタレーションレベルの並列処理を行う．評価に用いた入力画像は 165×192 である．

5.1.6 緑色検出

緑色検出 [19] では入力 1 として画像の RGB 値のうち R と B を 0 にしたもの，入力 2 として元の画像を用いる．入力 1 から入力 2 を減算した後，Otsu 法による閾値処理を用いてバイナリイメージ化を行うことで画像内の緑色箇所を検出する．本アプリケーションは実行時間の 72.8%を占める並列化可能ループに対してループイタレーションレベルの並列処理を行う．評価に用いた入力画像サイズは 384×512×3 である．

5.2 評価環境

本評価では汎用的な組込みマルチコアプロセッサとして用いられる 64-bit Cortex-A72 プロセッサを搭載する Raspberry Pi4 と Intel Xeon CPU X5670 にて計測を行う．

5.2.1 Cortex-A72

Cortex-A72 の構成を表 5.2.1 に示す．クアッドコアを搭載し，最大動作周波数は 1.5GHz である．また L1 キャッシュとして命令キャッシュを 48KB，データキャッシュを 32KB 各コアが持ち，L2 キャッシュは 4MB を搭載している．

5.2.2 Xeon X5670

Intel Xeon X5670 の構成を表 5.2.2 に示す．6 コアを搭載し, 最大動作周波数は 2.93GHz である．また L1 命令キャッシュ，L1 データキャッシュ，L2 キャッシュ，L3 キャッシュは順に 32Kbyte/core, 32Kbyte/core, 256Kbyte/core, 12Mbyte である．なお，評価では Raspberry Pi4 との対比のために 4 コアまで用いた．

5.3 Cortex-A72 での評価

図 5 に 5.1 節に述べたアプリケーション群を OSCAR 自動並列化コンパイラを用いて並列化を行い，Cortex-A72 上で行った評価結果を示す．縦軸は各アプリケーションの自動生成コードの逐次実行時間を 1 としたときの速度向上率を示している．

Cortex-A72 の 4 コア実行時，画素間勾配計算では 1.94 倍，二次元台座エッジの抽出では 1.36 倍，時系列データの移動平均計算では 2.97 倍の速度向上が得られた．この 3 種のアプリケーションは冗長な代入文を含む多重ループが実行時間のほとんどを占めるが，OSCAR 自動並列化コンパイラに新たに拡張したシンボリック伝搬とループインターチェンジ機能の組み合わせにより並列性解析，及び効果的なループ並列性抽出が可能になった．

時系列データの移動平均計算の 2 コア実行時では，コア数よりも特に大きな速度向上率が得られているが，これはシンボリック伝搬による冗長文の削除，及びループインターチェンジによるストライドアクセス回数の削減により

表 2 Cortex-A72 の構成
Table 2 Parameter of Cortex-A72.

コア数	4
最大動作周波数	1.5GHz
L1 命令キャッシュ	48KByte/コア
L1 データキャッシュ	32KByte/コア
L2 Cache	4MByte

表 3 Intel Xeon X5670 の構成
Table 3 Parameter of Intel Xeon X5670.

コア数	6
最大動作周波数	2.93GHz
L1 命令キャッシュ	32KByte/コア
L1 データキャッシュ	32KByte/コア
L2 Cache	12MByte

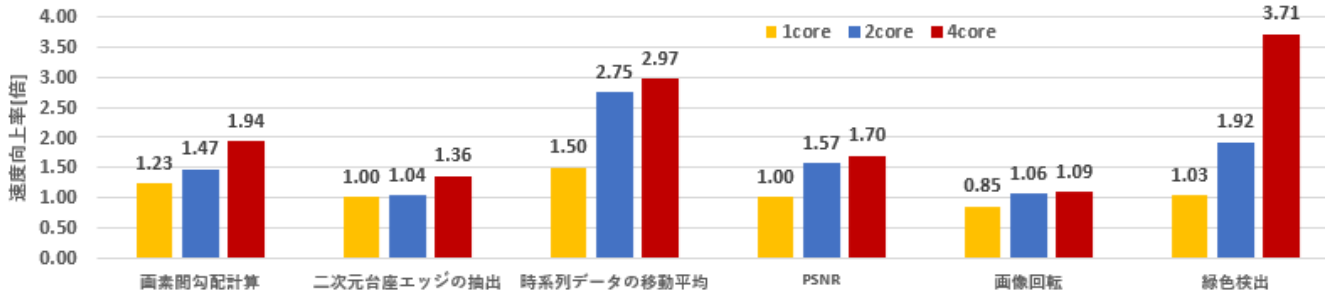

図 5 Cortex-A72 での評価結果

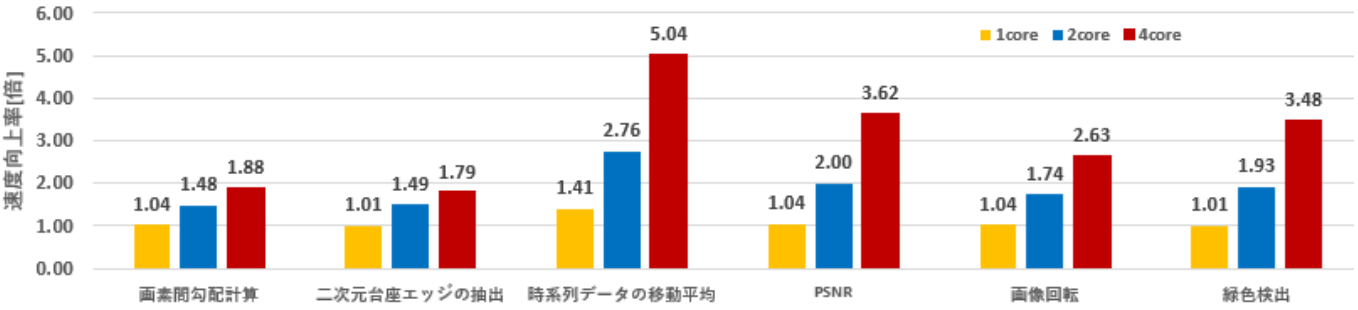

図 6 Xeon X5670 での評価結果

キャッシュヒット率が向上したためである．

PSNR と緑色検出では，基本的なループ並列性解析によってそれぞれ 1.70 倍，3.71 倍の速度向上が得られた．

画像回転では動的メモリ確保解析を行うことで並列性の抽出が可能になったが 1.09 倍の速度向上にとどまった．動的メモリ確保時にはライブラリ内部の排他制御が発生する．この際，キャッシュ無効化のブロードキャストや，パイプラインのフラッシュを伴うメモリオーダリングの命令が実行される．特に Cortex-A72 では Xeon X5670 に比較して，組込みプロセッサによる限られたメモリバンド幅やキャッシュ間通信性能による速度低下の影響を受けやすい．本性能は，この速度低下の影響と並列化による速度向上の影響の両方が働いたためと考えられる．

5.4 Xeon X5670 での評価

図 6 に 5.1 節に述べたアプリケーション群を OSCAR 自動並列化コンパイラを用いて並列化を行い，Xeon X5670 上で行った評価結果を示す．縦軸は各アプリケーションの自動生成コードの逐次実行時間を 1 としたときの速度向上率を示している．

Xeon X5670 の 4 コア実行時，画素間勾配計算では 1.88 倍，二次元台座エッジの抽出では 1.79 倍，時系列データの移動平均計算では 5.04 倍の速度向上が得られた．この 3 種のアプリケーションは冗長な代入文を含む多重ループが実行時間のほとんどを占める．

画素間勾配計算と二次元台座エッジの抽出ではシンボリック伝搬，時系列データの移動平均計算ではシンボリック伝搬とループインターチェンジを用いた並列性解析を行うことで効果的なループ並列性抽出が可能になった．時系列データの移動平均計算の 2，4 コア実行時では，コア数よりも特に大きな速度向上率が得られているが，これはシンボリック伝搬による冗長文の削除，及びループインターチェンジによるストライドアクセス回数の削減によりキャッシュヒット率が向上したためである．

さらに基本的なループ並列性解析によって PSNR では 3.62 倍，緑色検出では 3.48 倍の速度向上が得られた．

また，画像回転は動的メモリ確保解析を行うことで並列性の抽出が可能になり 2.63 倍の速度向上が得られた．画像回転は Cortex-A72 上では 1.09 倍の速度向上にとどまったが，Xeon X5670 上にて 2.63 倍の速度向上が得られたことから，本手法の並列性抽出の有効性が確認できた．

以上の評価結果から，従来の OSCAR コンパイラが持つ並列性解析能力によるマルチグレイン並列処理，及び本稿にて拡張したシンボリック伝搬とループインターチェンジ，及び動的メモリ確保解析機能の有効性が確認できた．

6. まとめ

本稿では OSCAR 自動並列化コンパイラの従来機能に加えて，新規に拡張したシンボリック伝搬とループインターチェンジ機能，及び動的メモリ確保解析を用いることで MATLAB/Simulink アプリケーションの自動並列化を行った．この結果，Raspberry Pi4（Cortex-A72 を 4 コア搭載）実行時，各アプリケーションの自動生成コードの逐次実行時間と比較して，画像勾配計算では 1.94 倍，二次元

台座エッジの抽出では 1.36 倍，時系列データの移動平均計算では 2.97 倍の速度向上が得られた．また PSNR では 1.70 倍の速度向上が得られた．さらに画像回転では 1.09 倍，緑色検出では 3.71 倍の速度向上がそれぞれ得られた．また，Intel Xeon X5670 での 4 コア実行時，画像勾配計算では 1.88 倍，二次元台座エッジの抽出では 1.79 倍，時系列データの移動平均計算では 5.04 倍の速度向上が得られた．また PSNR では 3.62 倍の速度向上が得られた．さらに画像回転では 2.63 倍，緑色検出では 3.48 倍の速度向上がそれぞれ得られた．

以上から，提案手法は MATLAB/Simulink アプリケーションを，コード生成機能により C コードを自動生成し，OSCAR コンパイラで並列化するため，ユーザは簡単に高速並列処理を行うことができ，今回新規実装した，複雑な動的メモリ確保，シンボリック伝搬とループインターチェンジを組み合わせた並列性解析機能を利用することによって，従来効果的な並列化が困難であった画素間勾配計算，二次元台座エッジの抽出，時系列データの移動平均計算，画像回転のようなアプリケーションでも効果的な自動並列化を行うことができるようになった．

参考文献

[1] MathWorks: Simulink. https://jp.mathworks.com/products/simulink.html.
[2] IPA:組込みシステムの先端的モデルベース開発実態調査報告. https://www.ipa.go.jp/sec/softwareengineering/reports/20120323.html.
[3] Parallel Computing Toolbox Document. https://jp.mathworks.com/help/parallel-computing/.
[4] MathWorks:parfor documentation. https://jp.mathworks.com/help/matlab/ref/parfor.html?s_tid=srchtitle.
[5] MathWorks:gpuArray documentation. https://jp.mathworks.com/help/parallel-computing/gpuarray.html?s_tid=srchtitle.
[6] 自動並列をサポートした MATLAB 関数の実行. "https://jp.mathworks.com/help/parallel-computing/run-matlab-functions-with-automatic-parallel-support.html".
[7] 梅田弾, 鈴木貴広, 見神広紀, 木村啓二, 笠原博徳. 組み込み向けモデルベース開発アプリケーションのプロファイル情報を用いたマルチコア用マルチグレイン並列処理. 情報処理学会論文誌, 第 57 巻, pp. 718–729, 2016.
[8] Hansang.B, Rudolf.E. Interprocedural symbolic range propagation for optimizing compilers. In *LCPC*, 第 4339 巻, 2005.
[9] Allen.J, Kennedy.K. Automatic loop interchange. In *ACM*, p. 233–246, June 1984.
[10] Z. Zhong and M. Edahiro. Model-based parallelization for simulink models on multicore cpus and gpus. In *2019 International SoC Design Conference (ISOCC)*, pp. 103–104, 2019.
[11] 久村孝寛, 枝廣正人, 中村祐一, 石浦菜岐佐, 竹内良典, 今井正治. Simulink モデルにもとづいた並列 c コード生成. コード生成と通信技術，組込み技術とネットワークに関するワークショップ ETNET, 2011.
[12] T. Kumura, Y. Nakamura, N. Ishiura, Y. Takeuchi, and M. Imai. Model based parallelization from the simulink models and their sequential c code. 2012.
[13] Minji Cha, S. Kim, and K. H. Kim. An automatic parallelization scheme for simulink-based real-time multicore systems. 2012.
[14] K.Aida M.Okamoto A.Yoshida H.Kasahara, H.Honda and W.Ogata. Oscar fortran multigrain compiler. In *Parallel Language and Compiler Research in Japan*. Springer,Boston,MA, 1995.
[15] Structurally varying bitonic filter. https://jp.mathworks.com/matlabcentral/fileexchange/68541-structurally-varying-bitonic-filter.
[16] 2 次元の畳み込み-matlab conv2-mathworks 日本. https://jp.mathworks.com/help/matlab/ref/conv2.html?searchHighlight=conv2&s_tid=srchtitle.
[17] Mathworks. ドキュメンテーション psnr. https://jp.mathworks.com/help/vision/ref/psnr.
[18] Mathworks. ドキュメンテーション imrotate. https://jp.mathworks.com/help/vision/ref/imrotate.
[19] Mathworks. file exchange green color detection simulink model/. https://jp.mathworks.com/matlabcentral/fileexchange/54036-green-color-detection-simulink-model.