情報処理学会論文誌 Vol.57 No.2 718–729 (Feb. 2016)

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

梅田 弾[1,a)] 鈴木 貴広[1] 見神 広紀[1] 木村 啓二[1] 笠原 博徳[1]

受付日 2015年5月24日, 採録日 2015年11月5日

概要：現在の組み込みシステム開発では MATLAB/Simulinik に代表されるモデルベース開発ツールがよく使用されるようになっている．また，開発されたモデルの複雑化とともに，このようなツールで開発されるアプリケーションのマルチコア上での高性能化，低消費電力化の要求が高まってきている．この要求に対して，モデル中のブロック間並列性を利用した並列化の提案はされているが，ブロック間だけでなく，ブロック内の並列性を利用したアプリケーション全体の並列性を有効利用できる方式は提案されていない．そこで，本論文では逐次 C プログラムから並列化 C プログラムを生成可能な OSCAR 自動並列化コンパイラを用いて，MATLAB/Simulink から Embedded Coder を使って自動生成された C プログラムに対して，モデル上に現れるブロック間並列性および，ブロック内のベクトル演算やユーザカスタマイズのコードからループ並列性を抽出し，マルチグレイン並列化を行う．また，マルチグレイン並列化の際に，Simulink 上で得られたプロファイル情報を使ったタスクスケジューリングを行うことによりスケジューリングの精度向上を行う．提案手法により Xeon X5670 上の 6 コアを使い，逐次実行時間と比較して道路追従アプリケーションでは 4.21 倍，血管抽出アプリケーションでは 5.80 倍，異常検出アプリケーションでは 4.10 倍の速度向上率が得られた．また，道路追従アプリケーションに関しては逐次の最悪実行時の実行時間と比較して，4.81 倍の速度向上率が得られた．

キーワード：マルチコア，並列処理，自動並列化コンパイラ，モデルベース開発，MATLAB/Simulink

Multigrain Parallelization Using Profile Information of Embedded Applications Generated by Model-based Development Tools on Multicore Processors

Dan Umeda[1,a)] Takahiro Suzuki[1] Hiroki Mikami[1] Keiji Kimura[1] Hironori Kasahara[1]

Received: May 24, 2015, Accepted: November 5, 2015

Abstract: Model-based development tools such as the MATLAB/Simulink have become popular for development of embedded systems recently. These applications require high performance and low power processing on multicores. Therefore, several researchers have proposed parallel processing of these applications utilizing parallelism among blocks in these models. However, no one proposes a method to extract all parallelism from not only among blocks but also in a block in these models. This paper proposes multigrain parallelization of C program generated by Embedded Coder from MATLAB/Simulink utilizing both coarse grain task parallelism among blocks and loop parallelism in a block including a vector operation or user's customized code using the OSCAR automatic parallelizing compiler. The compiler generates a parallelized C program from a sequential C program. The proposed method utilizes profiling information on Simulink to improve scheduling results into a multicore. It attains 4.21 times speedup for road tracking application, 5.80 times speedup for vessel detecting application and 4.10 times speedup for abnormality detecting application using six cores of Xeon X5670 compared with case of an ordinary sequential execution. Also, it attains 4.81 times speed up for road tracking application in worse case execution.

Keywords: multicore, parallel processing, automatic parallelizing compiler, model-based development, MATLAB/Simulink

1. はじめに

現在，自動車制御や航空機制御や医用画像処理等を含む組み込みシステム開発では MATLAB/Simulink [1] のようなモデルベース開発ツールが幅広く普及し始めている [2]．このようなツールではアルゴリズムの設計からコード生成および，検証まで一貫してモデル上で行うことができ，システム開発の生産性向上に大きな貢献を果たしている．

モデルベース開発により組み込みシステムの開発効率が向上した一方で，組み込みシステムの複雑化にともない，組み込みシステムにおけるアプリケーションソフトウェアの高速化の要求が高まっている．たとえば，自動車の Advanced Driving Assistance System（ADAS）[3], [4] のような画像処理や認識処理等を含む高機能な組み込みシステムでは，高速な応答性が要求されるため，いっそうの高速処理が求められている．しかしながら，消費電力の問題からシングルコアのままで要求性能を満たすには限界が来ており，複数コアを集積したマルチコアの採用が組み込み分野においても徐々に進んでいる．現在のマルチコアを搭載した組み込みシステムでは RH850 [5] や Cortex-A [6] や AURIX [7] のように 2 から 4 コアが一般的となりつつあるが，MPPA のようなメニーコア使った取り組み [8] も検討されている．このようなマルチコア・メニーコア上で性能を向上させるには，プログラムの並列化が必須である．プログラムの並列化では，タスク定義，タスク間並列性の抽出とタスクスケジューリングおよび適切な箇所へのデータ転送や同期コードの埋め込みが必要である．そのため，複雑化した組み込みシステムにおけるプログラムではユーザあるいはシステム開発者による手動並列化が難しくなっている．

また，マルチコアの普及にともない，MATLAB/Simulink では Parallel Computing Toolbox [9] のような並列処理ツールをサポートしている．このツールでは parfor による OpenMP [10] のようなユーザ指示文によるループ並列化と，指示された複数のモデル間の並列実行 [11] にとどまるため，ブロック間およびブロック内の並列化あるいはきめ細かい負荷分散は実現できない．また，Simulink モデルのサブシステム間並列処理を行うことができる RTI-MP [12] のようなツールも存在するが，手動による並列化支援のブロックを挿入することが必要である．

このような状況をふまえて，幅広い組み込み分野で普及しているモデルベース開発のアプリケーションにおいて，自動でマルチコアを有効利用できるようなコード生成系やコンパイラ等が必要とされている．そこで，本論文ではモデルベース開発で自動生成された C コードを OSCAR 自動並列化コンパイラ [25], [26] を用いて，マルチグレイン並列化を行い，自動でマルチコアを有効活用するための手法を提案する．具体的には，モデル上に存在するブロック間並列性に加えて，自動生成コードにのみ現れるブロック内並列性を自動で抽出することで，モデル上に現れるブロック間の並列性のみに制約されない，より効果的な並列性能向上を実現することで，本論文で提案手法の有用性や可能性を示す．

以下本論文では，2 章で関連研究について，3 章で提案手法について，4 章でモデルベース開発アプリケーションの並列化の手順について，5 章で OSCAR 自動並列化コンパイラによる並列性の抽出について，6 章でモデルベース開発アプリケーションの並列化手法について，7 章で実マルチコア上での提案手法の性能評価について述べ，8 章で本論文のまとめを述べる．

2. 関連研究

本章では MATLAB/Simulink でモデルベース開発されたアプリケーションの自動並列化に関わる従来研究について述べる．一般に MATLAB/Simulink では細かな Simulink ブロックを組み合わることによりモデルを形成するため，科学技術計算等に代表的な大規模ループが存在せず，従来のループ並列化技術 [13] による大幅な性能向上が難しい．また，モデルの結線情報が依存関係を表し，比較的並列性の抽出が容易なため，モデルベース開発アプリケーションの並列化に関わる研究ではモデル構造上に現れるブロック間並列性を利用した研究が一般的となっている．たとえば，Cha らは Simulink ブロックのデータ依存グラフを利用した Simulink アプリケーションのサブシステムレベルでの並列化を提案している [14], [15]．しかしながら，並列化の単位がサブシステムや関数単位であるため，並列処理による性能向上がサブシステム間で並列性が存在し，負荷が均衡する場合のみに限られ，きめ細かな並列性の抽出が実現困難である．また，久村らは Simulink モデルのファイル情報を入力として，Simulink ブロックの結線情報をデータ依存関係と見立てた CSP 理論によりブロック間の並列性を抽出し，並列化コードを生成する技術を提案している [16], [17]．久村らの手法では Simulink ブロック間の並列性がすべて抽出できるため，文献 [14], [15] と比較して幅広く並列性の抽出が可能である．

また，OSCAR 自動並列化コンパイラ [18] では MATLAB/Simulink の制御アプリケーションの C プログラム上に多く存在する条件分岐やルックアップテーブル等のブロック間の並列性を自動抽出し，オーバヘッドを最小限にして並列化する研究がされている [19]．

上記従来研究 [14], [15], [16], [17], [19] では，ブロック間やタスク間の並列性に着目したもので，並列性がブロック間やタスク間に限られる．そのため，ブロック間で並列性

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 162–0042, Japan
[a)] umedan@kasahara.cs.waseda.ac.jp

が十分に存在しない場合では，マルチコアを十分に活用できない場合がある．また，今後検討されているメニーコアではさらなる並列性の抽出が必要で，そのままでは十分に活用できないという課題がある．

3. 提案手法

モデル上でブロック間並列性が存在する一方で，画像処理を代表としたモデルでは信号線の情報がベクトル信号であり各ブロック内で並列性が存在する場合がある．また，基本的なブロックだけでなく，実用的なモデルではユーザカスタマイズブロックでモデルを構成する場合があり，このユーザカスタマイズブロック内にも並列性が存在する場合がある．そこで，ブロック間の並列性に加えて，ブロック内の並列性を利用することでさらなる並列性の抽出が期待できる．

従来の OSCAR 自動並列化コンパイラでの適用例 [19] では条件分岐構造やルックアップテーブル等のブロック間の並列性に着目して並列化を行っているが，本論文では，ブロック間の並列性に加えて，ブロック内の階層的な並列性に着目し，MATLAB/Simulink から Embedded Coder [20] を使って自動生成された組み込み向け C プログラムのマルチグレイン並列化 [18] を提案する．また，並列化の際には，Simulink の実行中で取得したプロファイル情報をアプリケーションの相対コストとして利用する．このプロファイル情報を利用した実行コストに基づきインライン展開やループ分割を行い，並列性を向上させる．並列性を向上させた後に，プロファイル情報を利用した実行コストを使って，マルチコアへのタスクスケジューリングを適用し，OSCAR API [21] による並列化コードの生成を行う．

従来研究のブロック間並列性を利用した並列化に対して，本論文ではブロック間並列性に加え，自動生成コードにのみ現れるループ並列性を利用したマルチグレイン並列化を適用し，より優れた並列化効果を実現する．

4. モデルベース開発アプリケーションの並列化手順

提案手法では，モデルベース開発で最も用いられているツールの 1 つの MATLAB/Simulink で開発されたアプリケーションを対象とし，Simulink モデルから Embedded Coder を使って自動生成された C プログラムに対して，自動プロファイリングおよび自動並列化を行う．

具体的には，まず図 **1** (1) のように Embedded Coder を使って自動生成された C プログラムに対して，OSCAR 自動並列化コンパイラのプロファイル機能を用いて，プロファイル関数と Simulink 上で動作可能とする MEX API [22] を挿入したプロファイル向け逐次 C プログラムを自動生成する．次に図 1 (2) のように，このプロファイル関数と MEX API が挿入された C プログラムをバイナリ MEX ファイルとしてビルド後，S-Function [23] として，Simulink 上で実行することで，プロファイル情報が自動生成される．

その後，図 1 (3) のようにこのプロファイル情報とコア数やキャッシュサイズ等のターゲットアーキテクチャのパラメータと自動生成された C プログラムを再度 OSCAR 自動並列化コンパイラに入力し，図 1 (4) のようにマルチグレイン並列性の抽出を行う．並列性の抽出後，図 1 (5) のようにインライン展開 [24] を適用し，階層構造を除去し，

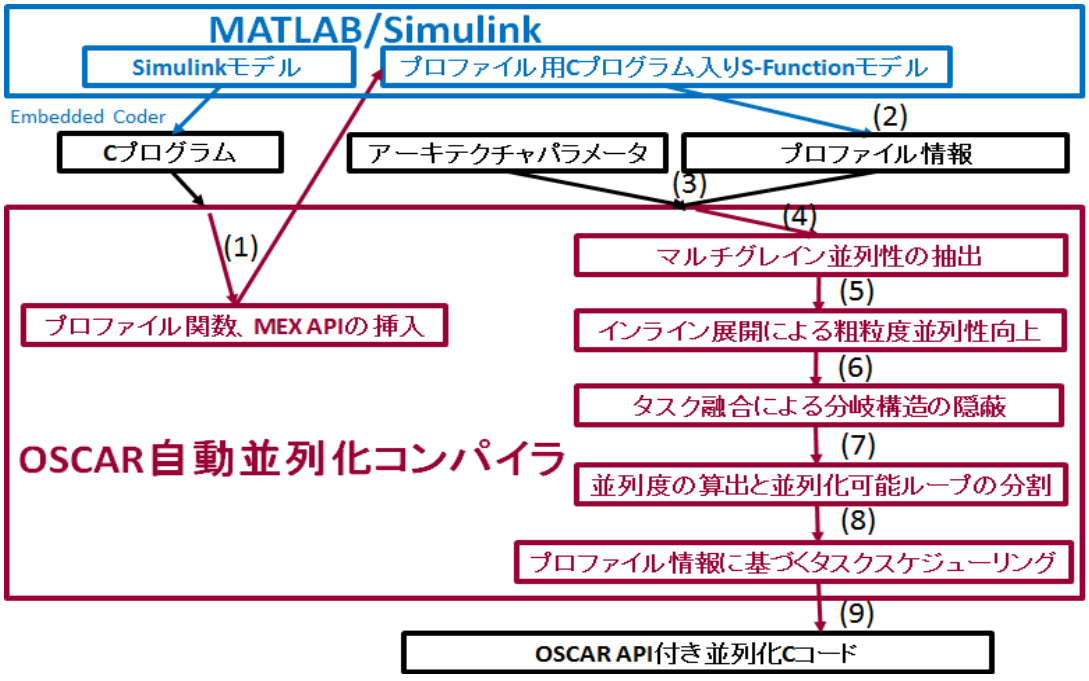

図 1 モデルベース開発アプリケーションの並列化フロー
Fig. 1 Parallelizing flow of applications which is model-based developed.

粗粒度タスク並列性を向上させる．次に，図 1 (6) のように条件分岐構造を持つタスクに対して，スタティックスケジューリングが適用できるように条件分岐の集合を 1 つのタスクとして扱うようにするタスク融合 [19] を適用する．図 1 (7) のようにプロファイル情報に基づく並列度の算出を行った後に，並列化可能なループを分割し，粗粒度タスク並列性に加えて，並列性を向上させる．また，図 1 (8) のように，並列タスクのスタティックスケジューリングの際にはスケジューリング精度を高めるために，プロファイル情報を用いる．並列化コード生成の際には，図 1 (9) のようにターゲットアーキテクチャに合わせて，OSCAR API で定義された並列化指示文の挿入を行う．これにより，マルチプラットフォームな並列化プログラムの生成が可能となる．

5. OSCAR 自動並列化コンパイラによる並列性の抽出

本手法では OSCAR 自動並列化コンパイラを用いて，粗粒度タスク（Macro Task）間の並列性を抽出し，それを Macro Task Graph（MTG）の形で表現する [25], [26]．代表的なエッジ検出フィルタの 1 つである Sobel フィルタの Simulink モデルを例として，OSCAR 自動並列化コンパイラによる並列性抽出の過程を説明する．

5.1 Sobel フィルタ Simulink モデルの具体例

Sobel フィルタは水平方向および垂直方向に対して，Sobel オペレータを用いて一次微分を行い，その後，正規化と閾値処理を行うことで実現できる．例として，Sobel フィルタを構築した Simulink モデルを**図 2** に示す．図 2 の CalcGx, CalcGy は Sobel オペレータを用いた水平方向および垂直方向に微分を行うユーザカスタマイズの MATLAB Function [27] である．EuclideanNorm は正規化と閾値処理を行うサブシステムである．また，図 2 の Simulink モデルに対して，Embedded Coder を使って自動生成された概略 C プログラムを**図 3** に示す．図 2 の Simulink モデルの入力として，648 × 484 の入力画像を使用しているため，図 3 の C プログラム中の 4，9 行目のようにループ回転数が 313,632 となっている．4–6 行目が入力を uint8 型から double 型への変換，7，8 行目が水平方向および垂直方向への微分（CalcGx，CalcGy），9–12 行目が正規化と閾値処理を行うサブシステム（EuclideanNorm）に該当する．図 2 より CalcGx と CalcGy の間でブロック結線が存在しないため，CalcGx と CalcGy が並列処理可能なことが明らかに分かる．

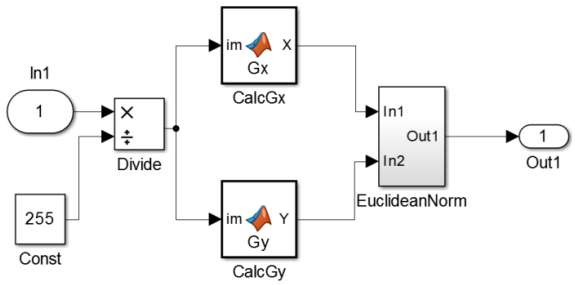

図 2 エッジ検出の Sobel フィルタ Simulink モデル
Fig. 2 Simulink model of Sobel filter for edge detection.

5.2 粗粒度タスク間並列性抽出と MTG

OSCAR 自動並列化コンパイラでは MATLAB/Simulink から自動コード生成された C プログラムを基本ブロック（bb），繰返しブロック（rb），サブルーチンブロック（sb）の 3 種類の粗粒度タスク（Macro Task（MT））に階層的に分割する [25], [26]．この MT が後にマルチコアへの割当て単位となる．図 3 の例では，**表 1** のように Divide に該当する 4–6 行目のループ処理が rb，MATLAB Function に該当する 7，8 行目の関数呼び出しが sb，EuclideanNorm に該当する 9–12 行目のループ処理が rb に変換される．また，OSCAR 自動並列化コンパイラの並列化可能ループ解析機能により，上記 rb が並列化可能ループを示す doall ループとして解析される．一方で，並列化が不可能なループの場合には loop として解析される．

```
1.  void Sobel_step(void)
2.  {
3.  ...
4.    for (i = 0; i < 313632; i++) {
5.        Sobel_B.Divide[i] =   ...;
6.    }
7.    Sobel_CalcGx();
8.    Sobel_CalcGy();
9.    for (i = 0; i < 313632; i++) {
10.   ...
11.     Sobel_Y.Out1[i] = ...;
12.   }
13. }
```

図 3 Sobel フィルタ Simulink モデルから Embedded Coder を使って自動生成された C プログラム
Fig. 3 C program from Simulink model of Sobel filter using the Embedded Coder.

表 1 図 2 の Simulink モデルと図 3 の C プログラムと MT の対応関係
Table 1 Relationship between Simulink mode in Fig. 2, C code in Fig. 3 and MT.

	rb1(doall1)	sb2	sb3	rb4(doall4)
Model	Divide	MATLAB Function	MATLAB Function	EuclideanNorm
C 行数	4-6	7	8	9-12

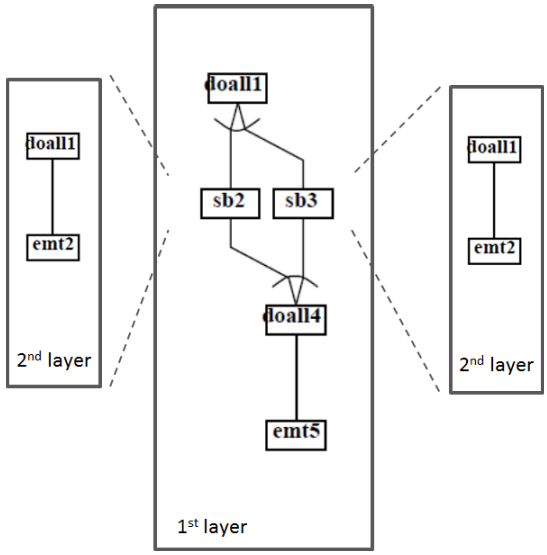

図 4 Sobel フィルタの MTG
Fig. 4 MTG of Sobel filter.

MT 生成後，OSCAR 自動並列化コンパイラは bb，rb（loop，doall），sb 等の MT 間のコントロールフローとデータ依存関係を表現した Macro Flow Graph（MFG）を生成する．さらにこの MFG から MT 間のコントロールフローとデータ依存を考慮し，各 MT が最も早く実行可能になる条件である最早実行可能条件の解析 [25], [28] を行い，並列性を表した階層的な Macro Task Graph（MTG）の生成を行う．

図 3 の C プログラムに該当する MTG を図 4 に示す．Sobel フィルタの MTG は 2 階層に分かれた階層的な MTG で表現される．図 4 では実線エッジがデータ依存関係を示している．そのため，横に並んだ MT は並列実行可能であることを示している．sb2 と sb3 の間でデータ依存関係がないため，第 1 階層（図 4 中の 1st layer）では Sobel_CalcGx と Sobel_CalcGy の関数間（sb2 と sb3 の間）で粗粒度タスク並列性が抽出されていることが分かる．すなわち，図 2 のブロック結線の構造上現れる CalcGx と CalcGy のブロック間の並列性を抽出できていることが分かる．また，第 1 階層の MTG では doall1 や doall4 のような doall と記述された MT がループ並列化可能な MT を示しており，Simulink のブロック内のベクトル演算でループ並列性を抽出できている．さらには，第 2 階層（図 4 中の 2nd layer）においても doall1 と解析され，MATLAB Function に該当する sb2 と sb3 の関数内でループ並列性の抽出ができており，ユーザカスタマイズブロックの MATLAB Function 内部でループ並列性の抽出ができている．

MTG の生成後，階層的な MTG を効果的に並列処理するために，MTG の並列性に応じて，複数のプロセッサコア（PC）をソフトウェア的にグループ化した，プロセッサグループ（PG）を定義する．階層的に処理するために，sb や rb（loop，doall）等の内部に MTG が存在する場合

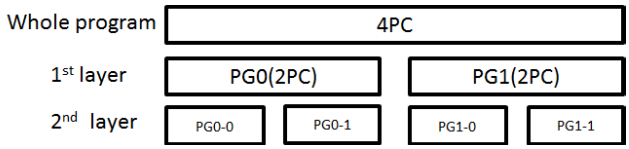

図 5 プロセッサグループ・プロセッサコアの階層適宜
Fig. 5 Hierarchical definition of PG and PC.

には，PG 内の PC をさらにソフトウェア的にグループ化を行う．図 5 に 4 コアの場合の階層的なグループ化の例を示す．全体としては 4 つのプロセッサコア（PC）を持つが，第 1 階層で 2 つのコアを，第 2 階層で 2 つのコアを割り当てた 2PG2PC の構成となる．このようにして，階層並列化が実現される．

5.3 マクロタスクスケジューリング

OSCAR 自動並列化コンパイラでは MTG で表現した並列性により，マルチコアへのタスクスケジューリングを MT 単位で行う．タスクスケジューリングの際には，MTG の形状，条件分岐等の実行時非決定性を基にダイナミックスケジューリングまたはスタティックスケジューリングが選択される．ダイナミックスケジューリングが選択される場合には，実行時にスケジューリングを行うダイナミックスケジューリング関数が並列化プログラムに埋め込まれる．一方，スタティックスケジューリングが選択される場合には，OSCAR 自動並列化コンパイラの解析時にデータ転送と同期オーバヘッドを考慮して実行時間が最小化できるように CP/ETF/MISF 法，ETF/CP/MISF 法，DT/CP/MISF 法および CP/DT/MISF 法 [29] のヒューリステックスケジューリングの結果中から最良のスケジューリングを採用し，静的にマルチコアへのタスクスケジューリングを行う．図 4 では条件分岐等の実行時非決定性が MTG 上に現れないため，実行時オーバヘッドがないスタティックスケジューリングが選択される．

また，タスクスケジューリング後，プロセッサコア間で同期処理やデータ転送が必要な箇所には，OSCAR 自動並列化コンパイラが共有変数を用いたデータ転送命令列と同期用共有変数と同期フラグを用いた同期命令列を挿入する．

6. 組み込み向けモデルベース開発アプリケーションの並列化手法

本章では MATLAB/Simulink でモデルベース開発された組み込み向けアプリケーションのプロファイル情報を用いたマルチグレイン並列化手法について述べる．

6.1 Simulink 上でのプロファイル情報の取得

通常，OSCAR 自動並列化コンパイラでは MT のコスト情報を用いて，並列化階層の決定やマルチコアへのタスクスケジューリングを行う．このコスト情報は，アーキテク

```
1   void func(void)
2   {
3       mp_prof_count(1);
4       mp_start_clock(1);
5           /*Processing MT1*/
6       mp_end_clock(1);
7        ...
8       mp_prof_count(4);
9       mp_start_clock(4);
10          /*Processing MT4*/
11      mp_end_clock(4);
12      ...
13  }
```

図 6 プロファイリング API の例
Fig. 6 An example of profiling API.

クチャごとの命令コストテーブルを用いて計算される．しかしながら，条件分岐やループ回転数等の実行時にならないと確定しない処理に対しては静的なコスト計算が困難である．本論文ではプロファイルより取得したコストを後述の最適化やタスクスケジューリングに利用するために，プロファイル取得用のプログラムを自動生成する．

OSCAR 自動並列化コンパイラのプロファイル機能ではターゲットのマルチコア上でプロファイル情報の取得が可能であるが，本提案手法では，Simulink の中で行える容易性とターゲットアーキテクチャへの移植性の面から Simulink の中でプロファイル情報を取得し，得られたプロファイル情報を相対コストとして使用する．

まず，OSCAR 自動並列化コンパイラが分割した各 MT にプロファイリング関数を挿入する．プロファイリング関数挿入例を**図 6** に示す．この関数では MT ごとの実行回数および実行サイクルを取得する．3，8 行目の mp_prof_count(arg) は MT ごとの実行回数を計測し，4–6，9–11 行目の mp_prof_start(arg) と mp_prof_end(arg) で挟まれた MT の実行サイクルを計測する．この引数 arg は MT 番号を示している．また，Simulink のシミュレーションの中でプロファイル情報が取得できるように，MEX API を挿入する．このプロファイリング関数と MEX の API 付きの C プログラムをバイナリ MEX ファイルとしてビルドし，S-Function として Simulink 上で実行することにより，シミュレーションを通して MT ごとの実行回数および実行サイクルが得られる．この情報に対して，プロファイル統計ツールを用いて，必要に応じて，平均実行サイクルの算出または最長パス上の実行サイクルの算出が可能であるが，本論文では平均実行サイクルをコスト情報として使用し，後述の各種の並列性向上のための最適化やマルチコアへのタスクスケジューリングに利用する．

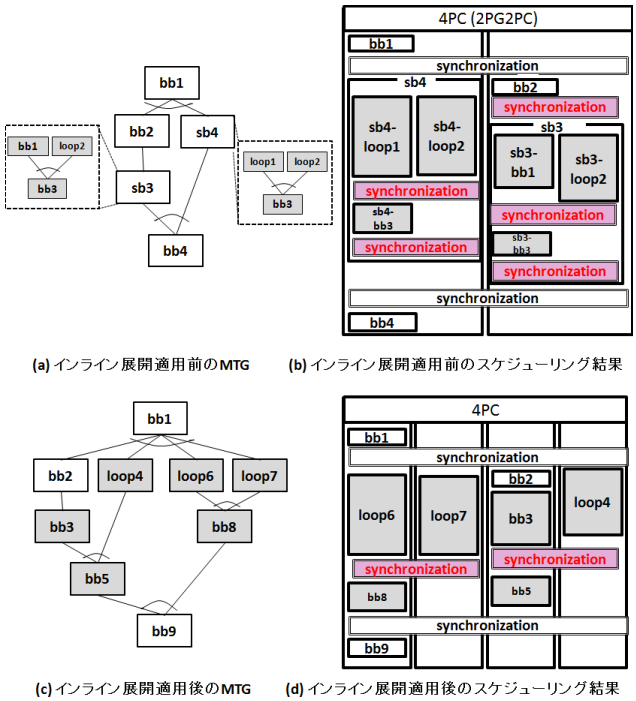

図 7 インライン展開による階層構造除去
Fig. 7 Elimination of hierarchical structure using inline expansion.

6.2 プロファイル情報を用いたインライン展開

5.2 節で前述の OSCAR 自動並列化コンパイラの階層並列化 [30] では第 1 階層で 2 つの PG 間で粗粒度タスク並列化，第 2 階層でループ並列化といったように並列化が可能であるが，階層並列化にともない PG 内および PG 間での階層内，階層間で同期処理を要し，オーバヘッドが生じる．本論文で対象となるアプリケーションでは階層並列化によるオーバヘッドが大きく見えてしまうため，同期処理オーバヘッドを最小化し，他階層の並列性を上位階層の並列性にあわせて効果的に並列化を行うことができるように，階層的に関数のインライン展開 [24] を行い，階層構造を除去する．本手法では，精度向上のため，プロファイル情報のコストを利用し，インライン展開を適用する．

インライン展開による階層構造除去の例を**図 7** に示す．図 7 (a) の MTG の例では sb3，sb4 内に階層構造を持ち，この MTG に対して 4 プロセッサコアで階層並列化を行う場合，図 7 (b) に示すような 2PG2PC のスケジューリング結果となる．図 7 (b) における縦方向が時系列を示し，MT の長さが実行時間を示す．また，synchronization は OSCAR 自動並列化コンパイラが挿入した同期命令，空白箇所がタスクの割当てがないことを示す．図 7 (b) では bb2，sb3 と sb4 を PG 間で並列化を行い，さらには各 PC 内で sb4-loop1 と sb4-loop2，sb3-bb1 と sb3-loop2 の間で階層並列化を行っている．階層並列化を行うため，階層内の実行（sb4，sb3）が終了後，同期処理を行っていることが分かる．これに対して，図 7 (a) に対して，プロファイル

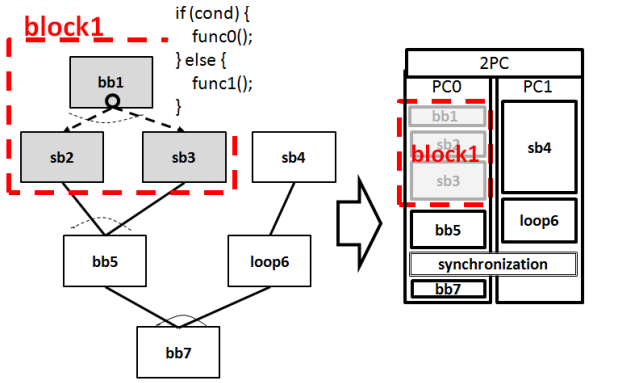

図 8 タスク融合を用いたスタティックスケジューリング
Fig. 8 Static scheduling using task fusion.

情報を基にインライン展開を適用した MTG が図 7 (c) である．図 7 (a) の sb3，sb4 が展開され，関数内の MT が最上位階層に表示されている．このインライン展開適用後の MTG のスケジューリング結果を図 7 (d) に示す．図 7 (d) のように loop6，loop7，bb3，loop4 が最上位階層で並列実行され，階層並列化による同期命令が減っていることが分かる．このようにして，階層構造を除去し，同期オーバヘッドを最小化する．

6.3 スタティックタスクスケジューリングのためのタスク融合

OSCAR 自動並列化コンパイラでは MTG の形状，条件分岐等の実行時非決定性を基にダイナミックスケジューリングまたはスタティックスケジューリングが選択される．しかしながら，Relational Operator や Switch ブロックを使用する Simulink モデルから生成される C プログラムには条件分岐が存在する．並列化を行う階層に条件分岐が存在すると，スタティックスケジューリングを適用できず，ダイナミックスケジューリングを適用しなければならず，実行時オーバヘッドが生じる．そこで，本手法では実行時オーバヘッドがないスタティックスケジューリングが適用できるようにタスク融合 [19] により条件分岐を並列化階層から隠蔽する．

タスク融合を用いたスタティックスケジューリングの例を**図 8** に示す．図 8 (a) では分岐を持つ MTG を示している．この MTG では小円が分岐を示しており，点線が分岐先を示している．図 8 (a) では実行時に bb1 からの分岐により，sb2 もしくは sb3 が実行される．sb2 もしくは sb3 の実行が bb1 の実行時まで決めることができないため，分岐を持つ場合はスタティックスケジューリングが適用できない．そのため，分岐を持つ MTG ではダイナミックスケジューリングを適用することになり，実行時オーバヘッドが生じる．そこで，実行時オーバヘッドのないスタティックスケジューリングが適用できるように，タスク融合を行い，分岐の集合を単一 MT に集約する．図 8 (a) では点線囲われた bb1，sb2，sb3 を単一 MT（block1）として扱い，単一プロセッサコアに割り当てる．タスク融合を用いて 2 プロセッサコアにスケジューリングした結果を図 8 (b) に示す．図 8 (b) ではタスク融合された block1 が単一プロセッサコアに割り当てられていることが分かる．

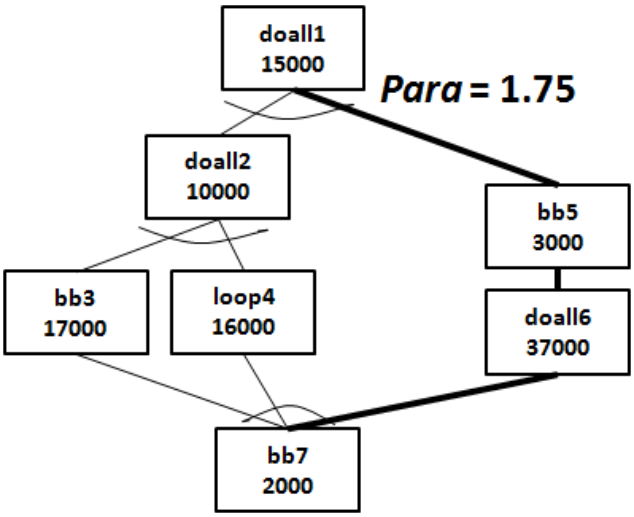

図 9 粗粒度並列度 Para の計算
Fig. 9 Calculation of coarse grain parallelism.

6.4 ループ分割による並列度向上

本手法では粗粒度タスク並列性に加え，ループ並列性を有効利用するために，並列化可能ループの分割を行う．

階層構造を除去，タスク融合後，6.1 節の Simulink 上で得られたプロファイル情報を用いて粗粒度並列度 Para [30] を算出する．Para とは下記のように逐次処理コスト Seq を対象 MTG のクリティカルパス長 CP で割って算出したオーバヘッドがないと仮定した場合の理想的な並列度である．

$$\mathrm{Para} = \mathrm{Seq}/\mathrm{CP}$$

例として，**図 9** の MTG の Para を求める．図 9 中の doall は並列実行可能な繰返しブロック（rb）を示し，loop は並列実行不可能な rb を示す．また，実線エッジはデータ依存，太線はクリティカルパス，ノード内の数値は逐次処理相対コストを表す．MTG 上の逐次処理コストは

$$\mathrm{Seq} = 15000 + 10000 + 17000 + 16000 + 3000 + 37000 + 2000 = 100000$$

太線上のクリティカルパス長は

$$\mathrm{CP} = 15000 + 3000 + 37000 + 2000 = 57000$$

となる．したがって，粗粒度並列度は

$$\mathrm{Para} = 100000/57000 \cong 1.75$$

となる．すなわち，従来のモデルベース開発アプリケーションの並列化手法に該当するブロック間並列化ではたかだか 1.75 倍までしか並列処理による性能向上が得られない．そこで，スケジューリングするコア数に合わせて並列処理の効果がある最小のコスト Tmin を超えるコストになるように並列化可能ループ Doall を分割し並列性を向上させる [30]．

4 プロセッサコアをターゲットに並列化可能ループの分割後の MTG の例を**図 10** に示す．図 9 における doall1，doall2，doall6 が 4 分割され，太字のクリティカルパスが変わっていることが分かる．この際のクリティカルパス長は

$$CP = 3750 + 2500 + 17000 + 2000 = 25250$$

となる．したがって，分割後の粗粒度並列度は

$$Para = 100000/25250 = 3.96$$

となる．すなわち，並列化可能なループを分割することで並列性を向上させることができる．

また，図 9 と図 10 の MTG を 4 プロセッサコアにスケジューリングを行った結果を**図 11** に示す．図 11 (a) は並列化可能ループ分割前のスケジューリング結果であり，4 つのコア資源が有効活用できていないことが分かる．図 11 (b) では分割されたループ間や他の MT と分割されたループ間で並列実行することにより，コア資源の利用率が改善され，スケジューリング長が短くなっている．

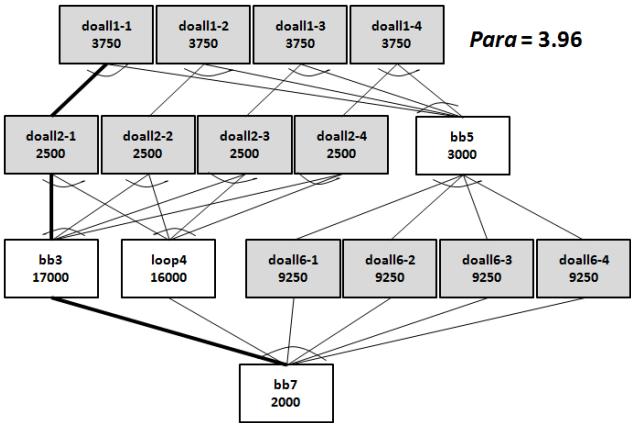

図 10 並列化可能ループ分割後の Para の計算
Fig. 10 Calculation of parallelism after loop division.

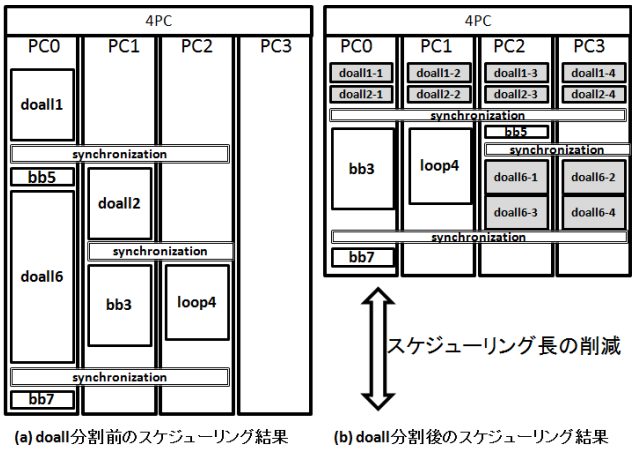

図 11 並列化可能ループ分割によるスケジューリング長の変化
Fig. 11 Transition of scheduling length using loop division.

7. 実マルチコア上での提案手法の性能評価

本章では提案手法を用いたモデルベース開発アプリケーションの実マルチコア上での並列性能評価を行う．

7.1 評価アプリケーション

本評価では MATLAB/Simulink でモデルベース開発された 3 種類のアプリケーションの並列化評価を行う．

7.1.1 道路追従

道路追従は色情報と検出エッジから路肩を検出し，追従するアプリケーションである [31]．入力となる画像サイズは 320×240 である．本評価では，ブロック間の並列性が抽出できるように画像の左右ごとで行う For Iterator ブロックを展開する．また，色彩とエッジ情報に Hough Transform ブロックを用いた直線検出をするが，マルチグレイン並列化による効果を評価するために，Hough Transform ブロックに Chen らが提案する並列アルゴリズムの Hough Transform [32] を S-Function として使用する．

7.1.2 血管抽出

血管抽出は Kirsch オペレータによる網膜の血管を抽出 [33] するアプリケーションである．入力となる画像サイズは 200×170 である．主なブロックは Kirsch オペレータを 45 度ずつ回した 8 方向の MATLAB Function のフィルタ処理と MinMax と Switch ブロックにより構成される．

7.1.3 異常検出

異常検出は株式会社 AandD 提供の画像情報を使った異常検出するアプリケーションである．入力となる画像サイズは 600×600 である．モルフォロジー演算の Opening，Dilation や特徴解析のブロブ解析を用いたブロック等で構成される．モルフォロジー演算やフィルタ処理は MATLAB Function のブロックを使用する．

7.2 評価マルチコア環境

本評価では高性能な組み込みシステムで利用が期待される Intel Xeon と汎用的な組み込みシステムで利用が期待される ARM Cortex のマルチコア上で評価を行う．

7.2.1 Xeon X5670

Xeon X5670 の構成を**表 2** に示す．最大動作周波数が

表 2 Xeon X5670 の構成
Table 2 Parameter of Xeon X5670.

コア数	最大動作周波数	L1 キャッシュ	L2 キャッシュ	L3 キャッシュ
6	2.93GHz	32 Kbyte/core	256 Kbyte/core	12 Mbyte

表 3 Cortex-A15 の構成
Table 3 Parameter of Cortex-A15.

コア数	最大動作周波数	L1 キャッシュ	L2 キャッシュ	L3 キャッシュ
4	1.60GHz	32 Kbyte/core	2Mbyte	

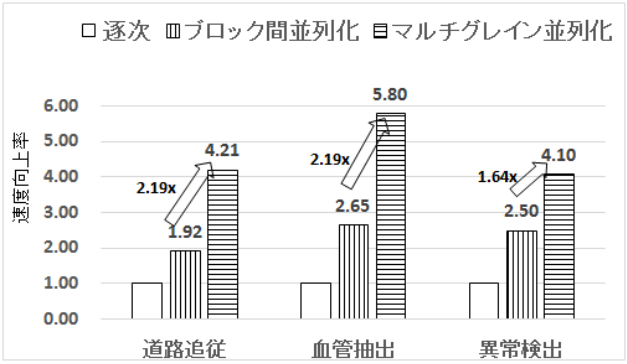

図 12 Xeon X5670 上でのモデルベース開発アプリケーションの並列性能評価
Fig. 12 Evaluation of model-based developed applications on Xeon X5670.

2.93 GHz の 6 コアの構成である．また，L1 キャッシュが 32 Kbyte，L2 キャッシュが 256 Kbyte，L3 が 12 MByte の構成である．

7.2.2 Cortex-A15

Cortex-A15 の構成を**表 3** に示す．最大動作周波数が 1.60 GHz の 4 コアの構成である．また，L1 キャッシュが 32 Kbyte，L2 キャッシュが 2 Mbyte，L3 キャッシュがない構成である．

7.3 Xeon X5670 上での評価

7.1 節の各アプリケーションを Xeon X5670 上で評価した平均速度向上率の結果を**図 12** に示す．縦軸は各アプリケーションの逐次実行時間と比較した速度向上率である．本評価では先行研究 [16], [17], [19] の手法に相当するブロック（タスク）間並列化を行った際の速度向上率と本論文の提案手法であるマルチグレイン並列化を行った際の速度向上率を示す．ブロック間並列化の場合，道路追従では 1.92 倍，血管抽出では 2.65 倍，異常検出では 2.50 倍であるように 6 コアを十分に活用できていない．一方で，マルチグレイン並列化の場合，道路追従では 4.21 倍，血管抽出では 5.80 倍，異常検出では 4.10 倍の速度向上が得られている．従来手法のブロック間並列化のみの実行に比べて，提案手法のマルチグレイン並列化では道路追従では 2.19 倍，血管抽出では 2.19 倍，異常検出では 1.64 倍の速度向上が得られ，速度向上率の改善を確認した．

また，道路追従に関しては，入力によって実行時間の変動が確認されるため，入力フレームごとの実行時間の変動を**図 13** に示す．横軸に動画のフレーム番号，縦軸にフレームごとの実行時間（ms）を逐次，ブロック間並列化およびマルチグレイン並列化についてそれぞれ示す．特に逐次の場合，82 回のフレームの中で，実行時間が 1.705 ms から 4.728 ms と大きく変動している．これは Hough Transform の中のエッジ点に対する処理のループ回転数が入力画像のエッジ点の数により変動するためである．また，ブロック間並列化の場合，実行時間が 0.815 ms から 2.510 ms まで変動し，逐次の最悪実行時（54 回目）では 2.00 倍の速度向上が得られている．一方，マルチグレイン並列化の場合，実行時間が 0.459 ms から 0.983 ms まで変動し，逐次の最悪実行時（54 回目）では 4.81 倍の最大速度向上が得られている．このマルチグレイン並列化の実行では，変動率が小さくなり，実行時間の分散がより小さくなっているのが分かる．そのため，大幅な高速化が実現した結果，入力画像によらず安定した実行が期待できる．

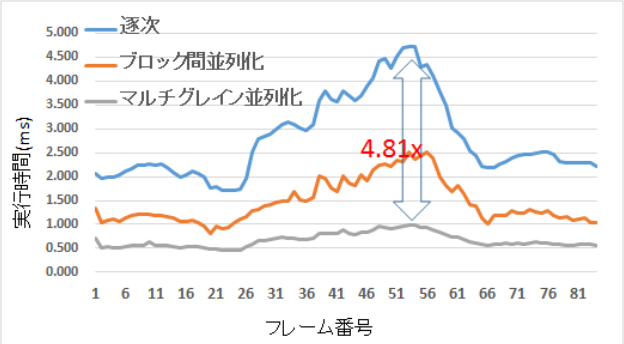

図 13 Xeon X5670 上での道路追従のフレームごとの実行時間
Fig. 13 Execution time per a frame on Xeon X5670.

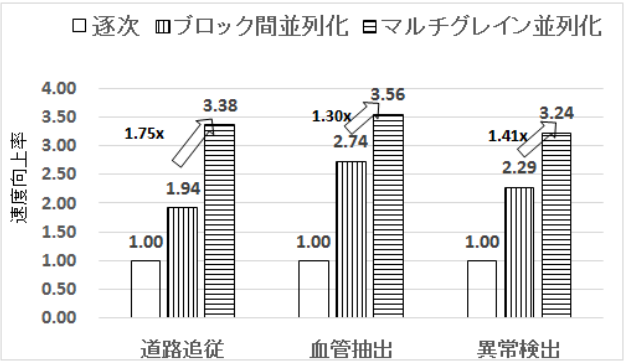

図 14 Cortex-A15 上でのモデルベース開発アプリケーションの並列性能評価
Fig. 14 Evaluation of model-based developed applications on Cortex-A15.

7.4 Cortex-A15 上での評価

7.1 節の各アプリケーションを Cortex-A15 上で評価した平均速度向上率の結果を**図 14** に示す．ブロック間並列化の場合，道路追従では 1.94 倍，血管抽出では 2.74 倍，異常検出では 2.29 倍であるように 4 コアを十分に活用できていない．一方で，マルチグレイン並列化の場合，道路追従では 3.38 倍，血管抽出では 3.56 倍，異常検出では 3.24

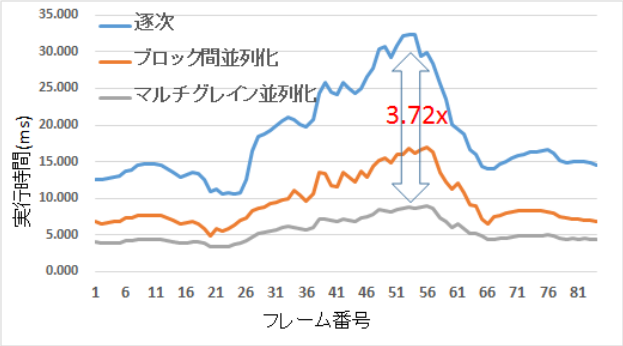

図 15 Cortex-A15 上での道路追従のフレームごとの実行時間
Fig. 15 Execution time per a frame on Cortex-A15.

倍の速度向上が得られている．従来手法のブロック間並列化のみの実行に比べて，提案手法のマルチグレイン並列化では道路追従では 1.75 倍，血管抽出では 1.30 倍，異常検出では 1.41 倍の速度向上が得られ，速度向上率の改善を確認した．

また，Cortex-A15 においても同様に，**図 15** に示すようにマルチグレイン並列化によりフレームごとの実行時間の分散が大幅に削減することができた．具体的には，特に逐次実の場合，82 回のフレームの中で，実行時間が 10.58 ms から 32.36 ms と大きく変動している．また，タスク間並列化の場合，実行時間が 4.88 ms から 17.02 ms まで変動し，逐次の最悪実行時（54 回目）では 2.01 倍の速度向上が得られている．一方，マルチグレイン並列化の場合，実行時間が 3.38 ms から 9.04 ms まで変動し，逐次の最悪実行時（54 回目）では 3.72 倍の速度向上が得られている．このマルチグレイン並列化の実行では，変動率が小さくなり，実行時間の分散がより小さくなっているのが分かる．7.3 節と同様にマルチグレイン並列処理の大きな並列性の抽出により，分散が減少した．

8. まとめ

本論文では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手法を提案した．Embedded Coder を使って自動生成された C プログラムに対して，ブロック間の並列性を粗粒度タスク並列性として，ブロック内の並列性をループ並列性として抽出した．コスト情報の精度を高めるために，Simulink 上で得られたプロファイル情報を利用して，インライン展開やループ分割により，並列性を向上させ，マルチグレイン並列化を行った．Intel Xeon X5670 の 6 コア上では従来手法のブロック間並列化を用いた場合，道路追従で 1.92 倍，血管抽出で 2.65 倍，異常検出で 2.50 倍の速度向上に対して，提案手法を用いた場合，道路追従で 4.21 倍，血管抽出で 5.80 倍，異常検出で 4.10 倍の速度向上が得られた．また，ARM Cortex-A15 の 4 コア上では，従来手法のブロック間並列化を用いた場合，道路追従で 1.94 倍，血管抽出で 2.74 倍，異常検出で 2.29 倍の速度向上に対して，提案手法を用いた場合，道路追従では 3.38 倍，血管抽出では 3.56 倍，異常検出では 3.24 倍の速度向上が得られた．提案手法のプロファイリング情報を用いたマルチグレイン並列処理により，従来のブロック間単階層並列化に比べて，粗粒度タスク間およびタスク内の階層並列化を行うことにより，Intel および ARM 等異なるマルチコアアーキテクチャ上で 1.30 倍から 2.19 倍の速度向上率が得られることが確かめられた．

提案手法により，より並列性を抽出することで，モデルベース開発アプリケーションにおいてより優れた並列化効果が実現した．そのため，提案手法がモデルベース開発されたアプリケーションの並列化による高速化に有効であるといえる．また，ブロック間並列性のみに制約されない並列性の抽出が実現したため，モデルベース開発されたアプリケーションにおいて，より多くのコアを搭載したマルチコアやメニーコアへの移行の可能性を示すことができ，提案手法が今後の組み込みシステムの開発に貢献できるといえる．

謝辞 本論文の性能評価において，モデルベース開発の異常検出アプリケーションを提供していただいた株式会社エー・アンド・デイの皆様に，感謝の意を表します．

参考文献

[1] MathWorks: MATLAB/Simulink, available from ⟨http://jp.mathworks.com/⟩.
[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，入手先 ⟨https://www.ipa.go.jp/sec/softwareengineering/reports/20120323.html⟩.
[3] Okuda, R., Kajiwara, Y. and Terashima, K.: A Survey of Technical Trend of ADAS and Autonomous Driving, VLSI Design, *2014 International Symposium on Automation and Test* (*VLSI-DAT*), pp.1–4 (2014).
[4] Dabral, S., Mody, M., Kamath, S., Buyue Z., Appia, V. and Batur, U.: Trends in camera based Automotive Driver Assistance Systems (ADAS), *2014 IEEE 57th International Midwest Symposium on Circuits and Systems* (*MWSCAS*), pp.1110–1115 (2014).
[5] RENESAS: RH850, available from ⟨http://japan.renesas.com/products/mpumcu/rh850/⟩.
[6] ARM: Cortex-A, available from ⟨http://www.arm.com/ja/products/processors/cortex-a/index.php⟩.
[7] Infinion: AURIX, available from ⟨http://www.infineon.com/cms/jp/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/channel.html?channel=db3a30433727a44301372b2eefbb48d9⟩.
[8] de Dinechin, B.D., Ayrignac, R., Beaucamps, P.-E., Couvert, P., Ganne, B., de Massas, P.G., Jacquet, F., Jones, S., Chaisemartin, N.M., Riss, F. and Strudel, T.: A clustered manycore processor architecture for embedded and accelerated applications, *High Performance Extreme Computing Conference* (*HPEC*), pp.1–6, IEEE (2013).
[9] MathWorks: Parallel Computing Toolbox, available from ⟨http://jp.mathworks.com/products/parallel-computing/index.html⟩.

[10] Open MP: Open MP, available from ⟨http://openmp.org/wp/⟩.
[11] MathWorks: Parallel Simulation, available from ⟨http://jp.mathworks.com/help/simulink/ug/running-parallel-simulations.html⟩.
[12] dSPACE: Real-Time Interface for Multiprocessor Systems (RTI-MP), available from ⟨http://www.dspace.com/ja/jpn/home/products/sw/impsw/rtimpblo.cfm⟩.
[13] Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.1 (1998).
[14] Cha, M. and Kim, K.: Deriving High-Performance Real-Time Multicore Systems based on Simulink Applications, *2011 9th IEEE International Conference on Dependable, Autonomic and Secure Computing* (2011).
[15] Cha, M., Kim, K. and Kim, K.: An Automatic Parallelization Scheme for Simulink-based Real-Time Multicore Systems, *Science & Engineering Reasearch Support Society*, Vol.5 (2012).
[16] 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).
[17] Kumura, T., Nakamura, Y., Ishiura, N., Takeuchi, Y. and Imai, M.: Model Based: Parallelization from the Simulink Models and TheirSequential C Code, *SASIMI* (2012).
[18] Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K. and Narita, S.: A Multi-grain Parallelizing Compilation Scheme for OSCAR, *4th International Workshop on Languages and Compilers for Parallel Computing* (1992).
[19] 梅田 弾，金羽木洋平，見神広紀，林 明宏，谷 充弘，森 裕司，木村啓二，笠原博徳：MATLAB/Simulink で設計されたエンジン制御 C コードのマルチコア用自動並列化，情報処理学会論文誌コンピューティングシステム，Vol.55, No.8 (2014).
[20] MathWorks: Embedded Coder, available from ⟨http://www.mathworks.co.jp/products/embeddedcoder/⟩.
[21] 佐藤卓也，見神広紀，林 明宏，間瀬正啓，木村啓二，笠原博徳：OSCAR API 標準解釈系を用いた Parallelizable C プログラムの評価，情報処理学会研究報告，Vol.2010-ARC-191-2 (2010).
[22] The Mathworks: MEX S-Fucntion, available from ⟨http://jp.mathworks.com/help/simulink/sfg/example-of-a-basic-c-mex-s-function.html⟩.
[23] The Mathworks: S-Function ブロック，入手先 ⟨http://jp.mathworks.com/help/simulink/slref/sfunction.html?refresh=true⟩.
[24] 白子 準，長澤耕平，石坂一久，小幡元樹，笠原博徳：マルチグレイン並列性向上のための選択的インライン展開手法，情報処理学会論文誌，Vol.45, No.5 (2004).
[25] 笠原博徳，合田憲人，吉田明正，岡本雅巳，本多弘樹：Fortran マクロデータフロー処理のマクロタスク生成手法，信学論，Vol.J75-D-I, No.8, pp.511–525 (1992).
[26] 笠原博徳，小幡元樹，石坂一久：共有メモリマルチプロセッサシステム上での粗粒度タスク並列処理，情報処理学会論文誌，Vol.42, No.4 (2001).
[27] The MathWorks: available from ⟨http://jp.mathworks.com/help/simulink/slref/matlabfunction.html⟩.
[28] 本多弘樹，岩田雅彦，笠原博徳：Fortran プログラム粗粒度タスク間の並列性検出法，信学論（D-I），Vol.J73-D-I, No.12, pp.951–960 (1990).
[29] 笠原博徳：並列処理技術，コロナ社 (1991).
[30] 小幡元樹，白子 準，神長浩気，石坂一久，笠原博徳：マルチグレイン並列処理のための階層的並列性制御手法，情報処理学会論文誌，Vol.44, No.4 (2003).
[31] The MathWorks, Color-based Road Tracking, available from ⟨http://jp.mathworks.com/help/vision/examples/color-based-road-tracking.html?refresh=true⟩.
[32] Chen, Y.-K., Li, E., Jianguo, L. and Wang, T.: Novel parallel Hough Transform on multi-core processors, *Acoustics, Speech and Signal Processing* (2008).
[33] Bhadauria, H., Bisht, S. and Singh A.: Vessels Extraction from Retinal Images, *IOSR Journal of Electronics and Communication Engineering* (2013).

梅田 弾 （学生会員）

1989 年生．2012 年早稲田大学基幹理工学部情報理工学科卒業．2013 年同大学大学院基幹理工研究科情報理工専攻修士課程修了．2013 年同大学院基幹理工研究科情報理工専攻博士後期課程進学．2014 年同大学基幹理工学部情報理工学科助手，現在に至る．マルチコアプロセッサのアーキテクチャ，コンパイラ，アプリケーションに関する研究に従事．

鈴木 貴広

1991 年生．2014 年早稲田大学基幹理工学部情報理工学科卒業．2014 年同大学大学院基幹理工研究科情報理工専攻修士課程進学，現在に至る．マルチコアプロセッサのアーキテクチャ，コンパイラ，アプリケーションに関する研究に従事．

見神 広紀 （正会員）

1984 年生．2009 年早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程修了，2011 年同大学基幹理工学部助手，2014 年同大学グリーン・コンピューティング・システム研究機構研究助手，現在に至る．並列アプリケーション，コンパイラ，マルチコアプロセッサアーキテクチャに関する研究に従事．

木村 啓二 （正会員）

1972 年生．2001 年早稲田大学大学院理工学研究科電気工学専攻博士課程修了，1999 年同大学理工学部助手，2004 年同大学理工学部コンピュータ・ネットワーク工学科専任講師，2005 年同助教授，2012 年教授，現在に至る．マルチコアプロセッサのアーキテクチャ，コンパイラ，アプリケーションに関する研究に従事．

笠原 博徳 （正会員）

1957 年生．1980 年早稲田大学理工学部電気工学科卒業，1985 年同大学大学院博士課程修了，工学博士，1986 年同大学理工学部専任講師，1997 年同教授，現在，情報理工学科教授．1987 年 IFAC World Congress Young Author Prize，1997 年情処坂井記念特別研究賞，2010 年 IEEE Computer Society Golden Core Member Award，2014 年文部科学大臣表彰科学技術賞（研究部門），情報処理学会 ARC 主査・会誌 HWG 主査・論文誌 HG 主査，経産省 NEDO 情報家電用マルチコアおよびアドバンスト並列化コンパイラ等のプロジェクトリーダ，文科省情報科学技術委員，IEEE-CS（Computer Society）理事，マルチコア STC 委員長等歴任．電子情報通信学会，IEEE-CS，ACM 等各会員．本会フェロー．