Vol.2015-EMB-38 No.4
2015/8/28

モデルベース開発向け画像処理ソフトウェアの並列化フレームワーク

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]

概要：近年の組み込みシステムはその複雑化に伴い，ソフトウェアの規模が年々増加している．そのため，開発効率の向上の面からモデルベース開発が普及している．このモデルベース開発ではモデル上で設計から実装や検証等までを一貫して行うことで，開発効率を大幅に改善することが可能である．特に自動車分野を始めとする組み込みシステムでは MATLAB/Simulink のようなモデルベース開発ツールがデファクトスタンダードになっている．これらのツールでは Embedded Coder 等のコード生成ツールによる組み込みプロセッサ向けの自動コード生成をサポートしており，設計されたアルゴリズムの人手による実装が不要になっている．しかしながら，このようなコード生成系では主にシングルコアプロセッサをターゲットとしており，各プロセッサ上の最適化は開発者に委ねられる．特に，多くの組み込みプロセッサが持つマルチコアをそのままでは有効利用できないといった問題がある．また，これらのツールは制御系を中心とした分野で幅広く普及している一方で，前述したターゲットプロセッサ上の最適化が十分ではないため，大量データを高速処理する必要がある画像処分野で MATLAB/Simulink のようなモデルベース開発ツールを有効活用するには強力な自動生成コードの最適化が必要である．そこで，本論文ではモデルベース開発の中でマルチコアプロセッサの性能を十分引き出せるように，モデルベース開発された画像処理ソフトウェア向けの並列化コード生成のフレームワークを提案する．具体的には Embedded Coder により自動生成された画像処理ソフトウェアに対して，OSCAR 自動並列化コンパイラを用いて，自動生成コードのプロファイルと並列性をモデルベース開発の中で提示することで，モデルの性能改善に役立てる．さらには，Simulink ブロック間から粗粒度並列性を，Simulink ブロック内や既存コード内からループ並列性を抽出し，得られたプロファイル結果を使って並列化を行う．並列化コードの生成の際に，MATLAB/Simulink のインターフェース関数を併せて出力することにより，Simulink 上での並列化コードの動作が実現した．その結果，ターゲットプロセッサへ実装を行う前工程で，モデルベース開発の画像処理アプリケーションでマルチコアを使った並列性能向上が確認できた．

1. はじめに

近年の組み込みシステムでは要求される様々な機能の導入に伴い，ソフトウェア規模が増大している．そのため，開発効率の改善を目指して，MATLAB/Simulink[1] を代表としたツールを使ったモデルベース開発が普及し始めている．これらのツールではブロックダイヤグラムや状態遷移図等を利用したモデルの設計から Embedded Coder[2] や TargetLink[3] を利用した自動コード生成による実装を一貫して行うことができる．特に自動車分野においては，信頼性や開発効率がより重要であるため，これらのようなモデルベース開発ツールがデファクトスタンダードとなっている．現在までには，主に自動車分野のエンジン制御やモータ制御のコントローラ等の開発でこれらのようなツールが用いられている．一方で，自動車分野に関して今後は画像処理を用いた Adavanced Driver Assistance System(ADAS)[4] や自動運転の開発を効率よく行う必要があり，さらには制御系と一体化した開発が必要なため，画像処理分野にもモデルベース開発の一貫した利用が期待される．

このような画像処理系では，大量データを高速処理することが求められるため，高速化を実現する 1 つの手段としてマルチコア等を使った並列処理が重要となる．しかしながら，自動コード生成ツールから生成されるコードはシングルコア向けで，マルチコア等による並列処理が未だにサポートされていない．

並列処理に関連したツールとして，Mathwork 社では Parallel Processing Toolbox[5]，dSPACE では RTI-MP[6] のようなツールが MATLAB/Simulink 向けにサポートされている．しかしながら，これらは指示された Simulink ブ

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

ロックを並列処理することに留まるため，自動並列処理且つ細かな並列処理が実現できない．また，並列処理の対象が Simulink 上でのシミュレーション向けで，組み込みプロセッサ上の実行がターゲットではない．

これらを踏まえて，モデルベース開発で自動生成されたコードから並列処理を行う研究がいくつか提案されている．例えば，[7] では MATLAB/Simulink 向けの並列処理ライブラリの提供により高速化を行う研究がされている．しかしながら，生成されるコードは特定ハードウェア向けの並列処理コードで，ハードウェアに強く依存する．また，最適化の範囲が特定のライブラリ箇所に留まるため，ソフトウェア全体の最適化の実現が困難である．

一方で，[8], [9] では Simulink モデルから並列化コードを生成する手法を提案している．上記手法ではモデルファイルから Simulink ブロックの結線情報を依存関係に見立て，並列性を抽出し，並列化コードを生成する．自動で汎用マルチコア向けの並列化コード生成が実現できる．しかしながら，並列化の単位が Simulink ブロック単位のため，並列性能がモデルの書き方に制約される．特に画像処理系の Simulink モデルでは各結線情報が行列値であり，各 Simulink ブロック内に並列性を持つことが多い．また，画像処理系の Simulink モデルでは Simulink で用意される画像処理ライブラリだけでは実装に限界があるため，S-Function[10] を利用した既存コードを含んだモデルも想定される．したがって，モデルの結線情報から並列化を行う手法では Simulink ブロック内や既存コード内から並列性の抽出ができないため，Simulink モデル全体で最適な並列化を行うことは困難である．

本手法で使用する OSCAR 自動並列化コンパイラでは，自動生成コードをコードレベルで並列性の解析を行うため，Simulink ブロック間の並列性に加え，Simulink ブロックや既存コード内から並列性を抽出することができる [11]．本論文では，モデル上に現れる Simulink ブロック間の並列性を粗粒度並列性として抽出し，Simulink ブロック内や既存コード内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．一方で，モデルベース開発において，Simulink のようなツール上で一貫して開発を行うことで開発期間短縮につながるため，本論文ではモデル上でのプロファイル結果を使い並列化を行い，モデル上で S-function として動作できるようコード生成を行う．具体的には，まず OSCAR 自動並列化コンパイラのプロファイル機能を使って，プロファイル向けの逐次コードを生成する．そのコードをモデル上で動作させることにより，プロファイル結果として処理負荷の高い Simulink ブロックを開発者にアドバイスする．それに加えて，OSCAR 自動並列化コンパイラの並列化機能を使って，並列解析結果を図示することによりモデルの性能改善に役立てる．次に，モデル上で得られたプロファイル結果を用いて並列化を行い，モデル上で動作できるように MATLAB/Simulink のインターフェース関数を併せて並列化コードの生成を行う．これにより，本論文では下記が実現した．

- モデル開発環境に適した並列化コード生成のフレームワーク
- ターゲットプロセッサに実装を行う前工程での並列化コードの実行及び並列化効果の検証

以下，本論文では，第 2 節で OSCAR 自動並列化コンパイラの概要について述べる．次に，第 3 節で OSCAR 自動並列化コンパイラを用いたモデルベース開発された画像処理ソフトウェアのマルチコア向けフレームワークについて述べる．第 4 節で Simulink 上での並列化コードの性能評価について述べる．第 5 節でまとめについて述べる．

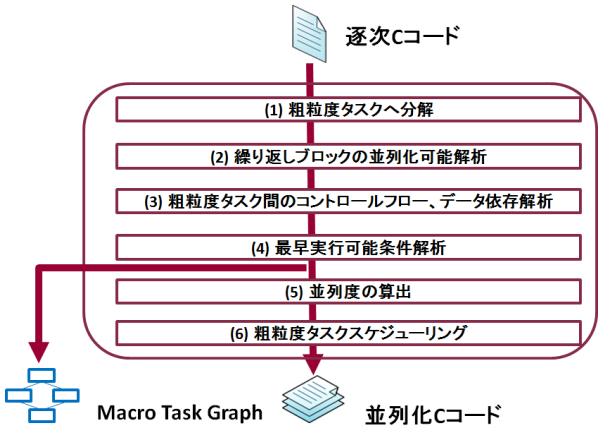

図 1 OSCAR 自動並列化コンパイラの概要

2. OSCAR 自動並列化コンパイラ

OSCAR 自動並列化コンパイラ [12], [13] は逐次 C コードを入力として，並列化 C コードを生成する．本論文では画像処理向け Simulink モデルから Embedded Coder を使って自動生成された逐次 C コードを入力として，OSCAR 自動並列化コンパイラを使って並列化 C コードを生成する．本節では OSCAR 自動並列化コンパイラによる並列化コード生成の概要を述べる．図 1 に OSCAR 自動並列化コンパイラの概要を示す．

2.1 粗粒度タスクの生成と粗粒度タスク間の並列性解析

OSCAR 自動並列化コンパイラでは粗粒度タスク (Macro Task)[12], [13] の単位で並列化を行う．この粗粒度タスクは以下の 3 種類により構成される．

1 基本ブロック (Basic Block : BB)
2 関数呼び出しブロック (Suroutine Block : SB)
3 繰り返しブロック (Repetition Block : RB)

BB は代入文や条件分岐，SB は関数呼び出し，RB はループ処理を示す．初めに図 1(1) で逐次 C コードをこれらの粗粒度タスクの単位に階層的に分割する．粗粒度タスクへ

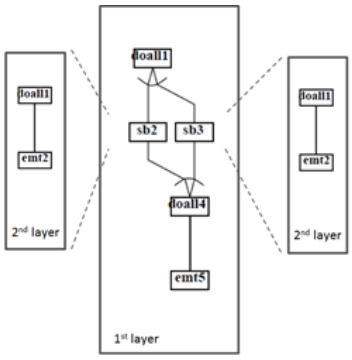

図 2 階層的な Macro Task Graph (MTG)

の分割後，図 1(2) で繰り返しブロック (RB) の並列化可能解析機能により，並列化可能な RB は DOALL，並列化が不可能な RB は LOOP として解析される．

次に，図 1(3) でこれらの粗粒度タスク間のコントロールフローとデータ依存関係を解析し，Macro Flow Graph (MFG)[12], [13] として表現する．さらに図 1(4) でこの MFG から粗粒度タスク間のコントロールフローとデータ依存を考慮し，各粗粒度タスク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 [12], [14] を行い，その結果を階層的な Macro Task Graph (MTG) として生成する．図 2 に MTG の例を示す．図 2 では 2 階層に渡り表現されている．図中の実線がデータ依存関係を示しており，横並びにある粗粒度タスクが並列実行可能であることを示している．また，図中の *doall* は並列実行可能なループを示している．すなわち，この MTG では粗粒度タスク間の並列性と粗粒度タスク内のループ並列性を示しており，マルチグレイン並列性を表現している．この際，図 1(5) で並列性の指針として逐次処理コストを対象 MTG のクリティカルパス長で割って算出したオーバーヘッドがないと仮定した場合の理想的な並列度 Para[15] がコンパイラによって算出される．その後，図 1(6) でマルチコアへの粗粒度タスクスケジューリングを行う．その結果を並列化 C コードとして出力する．

2.2 粗粒度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粗粒度タスク単位で行う．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ー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 法及び CP/DT/MISF 法 [16] 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．本論文では実行時オーバー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，予めタスク融合手法により分岐構造を MTG 形状から隠蔽する [11].

3. モデルベース開発向け画像処理ソフトウェアの並列化フレームワーク

本論文では MATLAB/Simulink を代表とするモデルベース開発内で設計からターゲットプロセッサへの実装までを一貫して行えるように，モデルベース開発ツールと密となった並列化フレームワークを提案する．

OSCAR 自動並列化コンパイラを使ったモデルベース開発向け画像処理ソフトウェアの並列化フレームワークを図 3 に示す．従来の開発手法では自動生成コードを S-Function に組み込んで Software-in-the-Loop Simulation(SILS) を行う一方で, 本手法で提案するフレームワークでは Embedded Coder が生成したコードから OSCAR 自動並列化コンパイラを使ってプロファイル向けコードを生成する．その後，このプロファイル向けコードを S-Function に組み込み動作させる．これにより，SILS 上で自動生成コードのプロファイル情報を得る．さらには OSCAR 自動並列化コンパイラが解析した並列性を図示することで，上流工程でモデルの性能改善に活用する．次に，この SILS で得られたプロファイル情報を使って，再度 OSCAR 自動並列化コンパイラに解析させることにより，並列化コード生成を行う．さらにはこの並列化コードを S-Function に組み込むことで，SILS 上で並列化コードの検証及び並列化による性能向上効果を確認を行う．

3.1 Simulink モデル上でのプロファイル

OSCAR 自動並列化コンパイラのプロファイル機能により分割された各粗粒度タスク間にプロファイラ関数を埋め込み，プロファイル向けのコード生成を行う．このプロファイラ関数では実行回数と実行サイクルの計測を行う．また，モデル上でプロファイル情報を取得できるように，MATLAB/Simulink のインターフェースの MEX 関数 [17] を併せて出力し, S-Function に組み込み動作させる．プロファイル向けコードを S-Function に組み込んだ例を図 4 に示す．図中の `Convert 2-D to 1-D` と `Reshape` を使い，モデル上の Simulink ブロック内で使用する行列と S-Function 内で使用する配列に相互変換する．特にモデルと S-Function の間の入力ポートは Embedded Coder 生成コード内の `Model_U` 変数，出力ポートは `Model_Y` 変数として OSCAR 自動並列化コンパイラで読み取り，MEX 関数内でマッチングさせる．

この S-Function を含んだ Simulink モデルを実行するこ

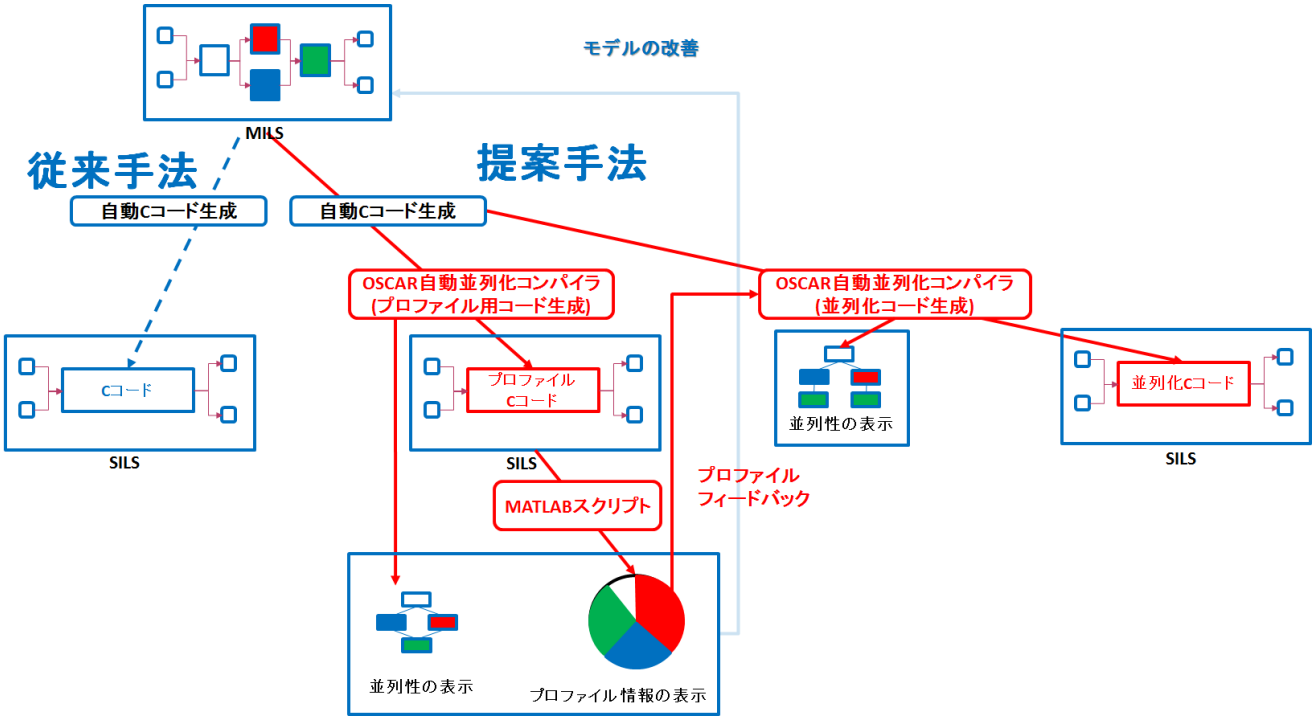

図 3 OSCAR 自動並列化コンパイラを使ったモデルベース開発向け画像処理ソフトウェアの並列化フレームワーク

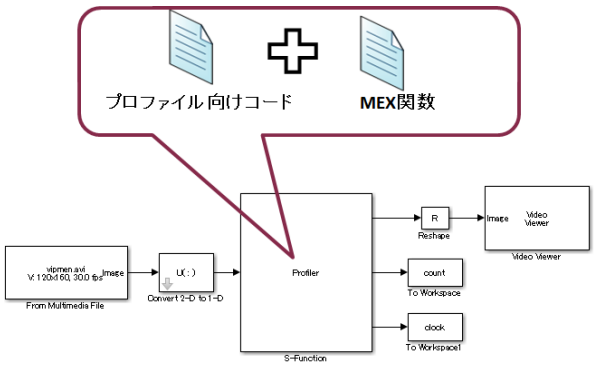

図 4 Simulink モデル上での自動生成コードのプロファイル

とで，プロファイラ関数で計測された各 MT の実行回数を `To Workspace` の `count` で取得し，各 MT の実行サイクルを `To Workspace` の `clock` で取得する．次にワークスペースに保存されたこれらの `count` と `clock` から MATLAB スクリプトを使って，プロファイルデータを生成する．必要に応じて，平均実行時間または最長パス上の実行時間を算出し，モデルの改善や並列化コード生成にフィードバックすることで，生成コードの最適化を行う．

本論文ではより上流工程で極力ターゲットプロセッサに依存がないように，モデル上のプロファイルを OSCAR 自動並列化コンパイラに利用するが，プロセッサ毎に最適化を行う場合，各プロセッサ上のプロファイル結果を活用することも可能である．

3.2 自動コード生成のマルチグレイン並列化

OSCAR 自動並列化コンパイラでは Simulink ブロック間の並列性を粗粒度タスク並列性として抽出する．また，Simulink ブロック内や S-Function を使った既存コード内の並列性をループ並列性として抽出する．例として，図 5 に示すソーベルフィルタを構成したモデルの並列性について述べる．図 5(a) の例では割り算の基本ブロックと畳み込みを行う 2 つの MATLAB Function ブロックと正規化を行うサブシステムモデルより構成される．Simulink ブロックの結線がデータ依存関係を表し，互いにブロック結線が存在しないため，畳み込みを行う 2 つの MATLAB Function ブロック間で並列処理が可能なことがわかる．同様に，図 5(b) に示す OSCAR 自動並列化コンパイラが解析した MTG では 2 つの MATLAB Function に該当する *doall2*，*doall3* が並列実行可能なことを表している．また，図中の *doall* が並列実行可能なループを表しており，各 Simulink ブロック内でループ並列性を抽出していることがわかる．すなわち，Simulink ブロック間の並列性を粗粒度並列性として抽出し，Simulink ブロック内の並列性をループ並列性として抽出できていることがわかる．このマルチグレイン並列性を抽出後，前述のプロファイル結果を基にマルチコアへのスタティックスケジューリングを行い，OSCAR 自動並列化コンパイラで並列化コードを生成する．

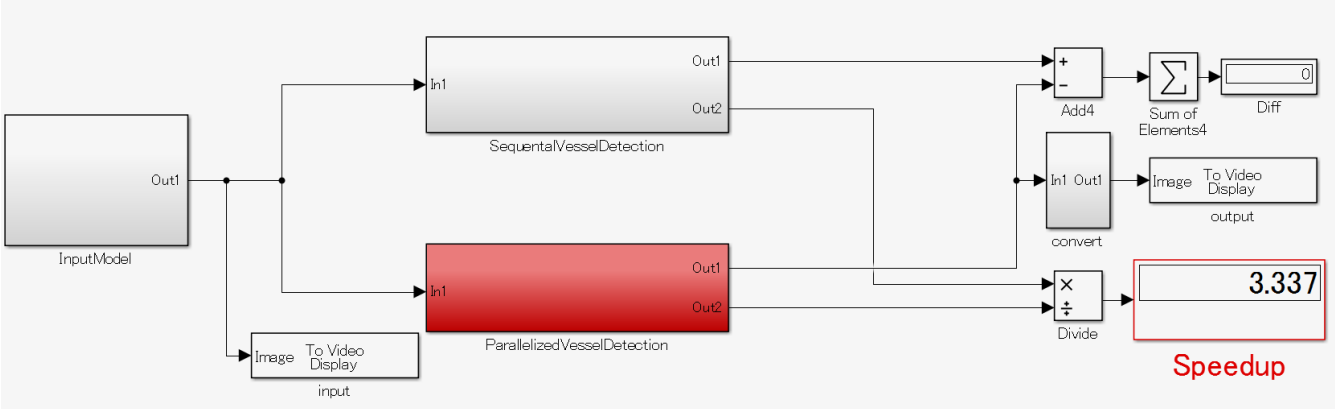

図 6 Simulink 上で並列化コードの動作

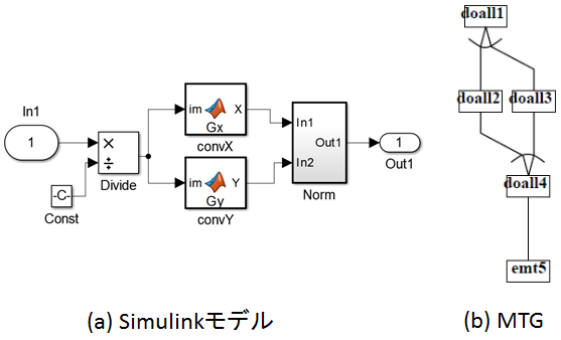

図 5 Simulink モデルと MTG

3.3 Simulink 上での並列化コードの実行

並列化コードをモデル上で動作できるように並列化コードの出力の際に，3.1 節と同様に MEX 関数を出力する．また，実行時間を測定できるように MEX 関数内にタイマー関数を挿入する．その後，S-Function として組み込むことで並列化コードを Simulink 上で並列実行させる．また，逐次コードと並列化コードを Simulink 上で実行することにより，ターゲットプロセッサへ実装を行う前工程で並列化による性能効果を測定することができる．図 6 に Simulink に逐次コードと並列化コードを S-Function として埋め込み，実行時間比を表示したモデルを示す．図 6 の上段では逐次コードを，下段では並列化コードを動作させている．Display ブロックを使って，ステップ毎の出力結果の差分 (*Diff*) と MEX 関数内に挿入されたタイマー関数で測定した実行時間比 *Speedup* を表示している．また，この並列化コードに前述のプロファイル機能を埋め込むことで，並列化コードのプロファイル解析を行うことも可能である．

4. Simulink 上での並列化コードの性能評価

4.1 実行環境

本節では Simulink 上で並列化コードを動作させることで，PC 上で並列化による効果を計測する．

使用する実行環境を表 1 に示す．使用するプロセッサは 3.4GHz で動作する 4 コアの構成である．使用する OS は Windows 7，MATLAB/Simulink のバージョンは R2014a である．また，MEX 関数のコンパイルには Microsoft Visual C++ 2010 (C) を使用する．

表 1 実行環境

プロセッサ	Xeon E3-1240 v3 @3.40GHz (4 core)
OS	Windows 7
MATLAB/Simulink	MATLAB R2014a
MEX コンパイラ	Microsoft Visual C++ 2010 (C)

4.2 評価アプリケーション

評価に使用するアプリケーションを下記に示す．

4.2.1 血管抽出

Kirsch オペレータによる網膜の血管を抽出するアプリケーションである [18]．入力画像サイズは 200 × 170 である．

4.2.2 オプティカルフロー

オプティカルフローは物体追跡を行うアプリケーションで，使用するアルゴリズムは Horn-Schunck である．また，入力画像サイズは 160 × 120 である．

4.2.3 路肩追従

路肩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 [19]．また，入力画像サイズは 320 × 240 である．

4.3 評価結果

本評価では MEX 関数内に挿入されたタイマー関数により，S-Function 内の実行時間を計測する．4 コア用に並列化し，Embedded Coder 出力の逐次コードと比較した並列性能向上率を図 7 に示す．性能向上は 3000 回実行の内の平均速度向上率である．血管抽出に関しては 3.42 倍，オプティカルフローに関しては 3.05 倍，路肩追従に関しては 3.19 倍の並列性能向上が Simulink 上の実行で得られ，OSCAR 自動並列化コンパイラを使ったモデルベース開発

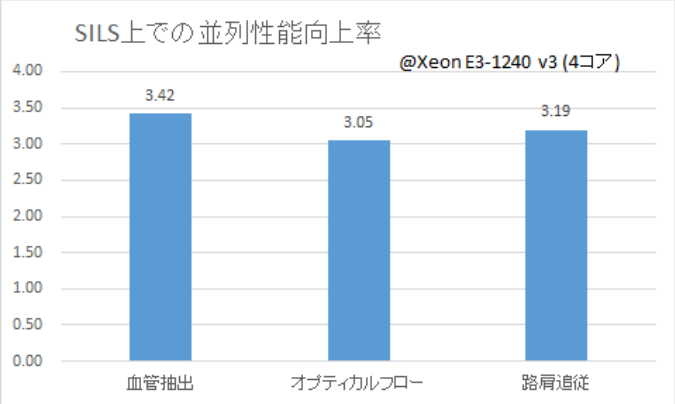

図 7 Simulink(SILS) 上での並列評価結果

の画像処理アプリケーションの高速化が実現した．また，ターゲットとなるプロセッサに実装を行う前工程で，並列性能向上率を確認することができた．

5. おわりに

本論文ではモデルベース開発ツールで最も使用されている MATLAB/Simulink 上で一貫した画像処理向け並列化コード生成フレームワークを提案した．OSCAR 自動並列化コンパイラにより Embedded Coder 生成コードの並列性解析とプロファイル解析をモデルベース開発の中で実現した．また，モデル上で得られたプロファイル情報と Embedded Coder 生成コードから並列化コードと MEX 関数を生成し，S-Function として組み込むことで自動並列化コードを Simulink 上での動作が実現した．これにより，Simulink 内での並列化コードの検証と並列化による効果をターゲットプロセッサに実装を行う前工程で確認できるようになった．したがって，提案するフレームワークによりモデルベース開発に密となった並列化コードを生成することで，自動生成コードの最適化がより容易に行えるようになった．

参考文献

[1] MathWorks: MATLAB/Simulink, http://www.mathworks.com/products/simulink/.
[2] MathWorks: Embededed Coder, http://www.mathworks.com/products/embedded-coder/.
[3] dSPACE: TargetLink, https://www.dspace.com/ja/jpn/home/products/sw/pcgs/targetli.cfm.
[4] Okuda, R., Kajiwara, Y. and Terashima, K.: A Survey of Technical Trend of ADAS and Autonomous Driving (2014).
[5] MathWorks: Parallel Computing Toolbox, http://www.mathworks.com/products/parallel-computing/.
[6] dSPACE: RTI-MP.
[7] 杉村武昭, 野田英行 and 下村英介: モデルベース開発と PILS(Processor In Loop Simulation) を活用した組込み向け超並列プロセッサのソフトウェア開発, **電子情報通信学会誌**, pp. 141–146 (2011).
[8] 久村孝寛, 枝廣正人, 中村祐一, 石浦菜岐佐, 竹内良典 and 今井正治: Simulink モデルにもとづいた並列 C コード生成 (2011).
[9] Kumura, T., Nakamura, Y., ISHIURA, N., TAKEUCHI, Y. and IMAI, M.: Model Based Parallelization from the Simulink Models and Their Sequential C Code (2012).
[10] MathWorks: S-Function, http://www.mathworks.com/help/simulink/slref/sfunction.html.
[11] 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 and 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**, pp. 1817–1829 (2014).
[12] 笠原博徳, 合田憲人, 吉田明正, 岡本雅巳 and 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**, pp. 511–525 (1992).
[13] 笠原博徳, 小幡元樹 and 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**, pp. 910–920 (2001).
[14] 本多弘樹, 岩田雅彦 and 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**, pp. 951–960 (1990).
[15] 小幡 元樹, 白子 準, 神. 浩. 石. 一. 笠. 博.: 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**, pp. 1044–1055 (2003).
[16] 笠原博徳 (ed.): **並列処理技術**, コロナ社 (1991).
[17] MathWorks: MEX, http://www.mathworks.com/help/matlab/call-mex-files-1.html.
[18] Bhadauria, H., Bisht, S. and A, S.: Vessels Extraction from Retinal Images, *IOSR Journal of Electronics and Communication Engineering*, pp. 2278–2834 (2013).
[19] MathWorks: 路肩追従アプリケーション, http://jp.mathworks.com/help/vision/examples/color-based-road-tracking.html.