情報処理学会論文誌 Vol.55 No.8 1817–1829 (Aug. 2014)

# 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

梅田 弾[1,a)] 金羽木 洋平[2] 見神 広紀[1] 林 明宏[1,†1]
谷 充弘[3] 森 裕司[3] 木村 啓二[1] 笠原 博徳[1]

受付日 2013年11月12日, 採録日 2014年5月17日

**概要**：近年の自動車では安全性・快適性・環境適合性が求められ，これらを実現するために自動車制御系のソフトウェアが年々より高度化している．制御の高度化と同時に，これらを実現するソフトウェアをリアルタイムで動作させるために，プロセッサの高速化が必要である．しかし，シングルコアの動作周波数の向上が困難であることから，1コアによる処理性能向上が限界となり，今後の自動車制御系でマルチコアへの移行が進んでいくと考えられる．また，自動車制御系において開発期間の短縮および信頼性の向上のためにMATLAB/Simulinkによるモデルベース設計が普及している．しかし，現時点でこのようなモデルベース設計で自動的にコード生成されるソースコードはマルチコア上で自動的に並列処理できるまでには至っていない．そこで，本論文ではMATLAB/Simulinkによって設計された制御モデルからEmbedded Coderにより自動生成されたエンジン制御Cコードをマルチコア上で動作するための並列化手法を提案する．提案手法を用いて，従来手動ではタスク粒度が細かく並列化が困難であった条件分岐と算術代入文からなるエンジン制御Cコードを OSCAR 自動並列化コンパイラにて自動並列化した．RP2やV850E2R等の組み込みマルチコア上で実行したところ，2コアで最大1.91倍，4コアで最大3.76倍の性能向上が得られた．

**キーワード**：マルチコア，自動並列化コンパイラ，モデルベース設計，自動車エンジン制御

# Automatic Parallelization of Designed Engine Control C Codes by MATLAB/Simulink

Dan Umeda[1,a)] Youhei Kanehagi[2] Hiroki Mikami[1] Akihiro Hayashi[1,†1] Mitsuhiro Tani[3]
Hiroshi Mori[3] Keiji Kimura[1] Hironori Kasahara[1]

Received: November 12, 2013, Accepted: May 17, 2014

***Abstract:*** Recently, more safety, comfort and environmental feasibility are required for the automobile. Accordingly, control systems need performance enhancement on microprocessors for real-time software which realize that. However, the improvement of clock frequency has been limited by power consumption and the performance of a single-core processor which controls power has reached the limits. For these factors, multi-core processors will be used for automotive control system. Recently Model-based Design by MATLAB and Simulink has been used for developing automobile systems because of elimination time of development and improvement of reliability. However, auto-generated-code from MATLAB and Simulink has been functioned on only single core processor so far. This paper proposes a parallelization method of engine control C codes for a multi-core processor generated from MATLAB and Simulink using Embedded Coder. The engine control C code which composed of many conditional branches and arithmetic assignment statements and are difficult to parallelize have been parallelized automatically using OSCAR automatic parallel compiler. In this result, it is succeeded to attain performance improvement on RP2 and V850E2R. Maximum 1.9x speedup on two cores and 3.76x speedup on four cores are attained.

***Keywords:*** multicore, automatic parallelizing compiler, model-based design, automotive engine control

© 2014 Information Processing Society of Japan

## 1. はじめに

近年の自動車ではさらなる安全性，快適性，環境適合性等が求められている．そこで，これらを満足させるために，エンジン制御のようなリアルタイム制御系の高度化が重要となっている．制御系の高度化のためには，それらを実現するためのプロセッサの高性能化が重要となる．たとえば，安全，快適で燃費の良い自動車開発にとって重要なエンジン制御系を高度化するためには，制御アルゴリズムの高度化，新制御機能の実現等による計算負荷の増大を避けられない．そのため，リアルタイム制御を実現しているプロセッサの高速化が必須となる．しかし，従来のように，プロセッサの高速化のために動作周波数を向上させることは，プロセッサの発熱量の増大と，それにともなう長期の信頼性の問題により困難である．このため，現在シングルコアプロセッサで動作する車載チップに対して，今後は1チップ上に低動作周波数のプロセッサコアを複数集積したマルチコアプロセッサへの移行が自動車分野でも進んでいくと考えられる．

たとえば，Seo Kyungil 等によって Unified Chassis Control（UCC）アルゴリズムを利用した電子制御ユニットのマルチコア化の提案がされている [1], [2]．これは電子制御ユニットを supervisor, main controller, fault detection/ isolation/ tolerance control controller の 3 つに機能分散を行い，3 コアで並列処理を行う．また，Aurelien Monot 等は OS レベルで電子制御ユニットの機能分散 [3] を提案している．これらの手法では，機能分散によりシステムスループットを向上することが可能である．しかしながら，各機能のレイテンシ削減は実現できない．

また，車載のマルチコアの適用に関しては，2 コアを集積した SH-Navi3 [4] のマルチコアを用いた走行支援システム等の検討，導入が行われているが，現状ではエンジン制御系でマルチコアの適用が発表された例はない．

エンジン制御系へのマルチコアの適用に関しては，自動車エンジン制御プログラムへ OSCAR コンパイラ [5] を適用する試みも行われている．これまでに従来の手書きの自動車エンジン制御に対しての並列性の抽出方法等を提案している [6]．

一方，近年の自動車分野のソフトウェア開発では，信頼性や開発コスト削減の面で，MATLAB/Simulink [7] によるモデルベース設計が普及している．MATLAB/Simulink によるモデルベース設計では，設計したモデルから Embedded Coder [8] を用いた C コードの自動生成がサポートされている．しかし，MATLAB/Simulink によって設計されたモデルから Embedded Coder により自動生成される C コードはシングルコア向けのコードであり，マルチコアを十分に活用できない．そのため，モデルベース設計において，マルチコアを有効利用するコード生成系や並列化コンパイラ等が必要である．しかし，マルチコア上で，このようなモデルベース設計されたエンジン制御 C コードの計算を，従来の 1 コアより高速に行うためには，ソースコードを並列処理単位であるタスクへ分割し複数のプロセッサに，各タスク実行時間とプロセッサの通信時間を考慮して割当てを行い，並列化する必要がある．

特に，MATLAB/Simulink によってモデルベース設計された制御モデルから Embedded Coder により自動生成されたエンジン制御 C コードには並列化が適した大きなループ構造がなく，条件分岐と代入文の連続によりその構造が複雑なため，通常のプログラム開発者が手動で並列化を行うことは困難で長期間を要する．

商用の並列化コンパイラとしては，Intel コンパイラや IBM コンパイラが利用されているが，動作環境が汎用マシンやサーバマシンであるため，組み込みプロセッサをサポートしていない．また，市販の並列化コンパイラは並列化の対象がループ構造であるため，本論文が対象とする Embedded Coder により自動生成される自動車エンジン制御 C コードからはループ構造以外での並列性の抽出ができない．

このような状況をふまえ，MATLAB/Simulink によって設計されたモデルからコード生成系より生成される C コードの並列化に関する従来技術がいくつか存在する．MathWork 社ツールでの同時実行設定のカスタマイズ [9] や Minji Cha 等によるサブシステム（機能）レベルでの並列化 [10], [11] 等があげられる．しかし，これらは 1 つの Simulink モデル内のサブシステム間で並列実行するものであり，大規模シミュレーション向けの技術である．そのため，リアルタイム制御のエンジン制御系では不向きである．また，上記手法では Simulink 上の各サブシステムレベルで並列性が存在し，かつ各サブシステムの実行負荷が均等となっていることが前提である．そのため，並列処理により得られる性能向上がモデルの書き方やアルゴリズムに依存してしまう．

また，組み込みをターゲットとしている技術としては，久村等による Simulink モデルの mdl ファイルを入力として Simulink ブロックの並列性を抽出して，自動生成された C コードから並列化 C コードを生成する技術 [12], [13] がある．この技術ではサブシステムによる階層構造とフィードバックループ構造を除去した中間モデルから CSP 理論に

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] 富士通株式会社
Fujitsu Corporation, Kawasaki, Kanagawa 211–8588, Japan
[3] 株式会社デンソー
DENSO Corporation, Kariya, Aichi 448–8661, Japan
[†1] 現在，ライス大学
Presently with Department of Computer Science, Rice University
[a)] umedan@kasahara.cs.waseda.ac.jp

© 2014 Information Processing Society of Japan

より，並列化 C コードを生成し，パイプライン式に並列実行を行う．ここで，モデルベース設計ではトップアーキテクチャの抽象化レベルが高く，理解しやすい制御構造であり，より抽象度を高くすることが求められる（モデルの最適化）．一方，実際に動作させる組み込みマイコン上のソフトウェアではより小さく，速く，効率的なコード生成でより CPU アーキテクチャに近い記述が求められる（CPU 最適化）．モデルの最適化と CPU 最適化という 2 つの要件の間には方向性に大きな差があり，文献 [12], [13] の手法ではこの差分解決する手段として，Simulink 上での階層化設計を提案している．上記手法では抽象度の高いモデル上で最適化を行うため，トップアーキテクチャから順にターゲットの構成に近い Simulink モデルを作成して，トップアーキテクチャとの違いをつねに検証する必要があるが，CPU アーキテクチャのシミュレーション要素が存在するため，実用的な時間で検証することが困難となる．また，CPU やメモリ構成等を考慮したターゲットマルチコア CPU に最適なコード生成結果を得るためにはモデル CPU の概念を取り入れたアーキテクチャを考慮したモデルからコード生成が必要である．

一方で，本論文では CPU 最適化として，抽象度の高い段階で C コード生成を行い，OSCAR コンパイラで得られた数々の最適化手法の適用する．ターゲットマルチコア CPU に合わせてコードレベルで最適化を行うため，Simulink 関連ツールとの依存性がなく，ターゲットマルチコア CPU に最適なコード生成が可能である．また，Simulink 関連ツールとは依存が存在しないため，従来の mdl 形式の Simulink モデルおよび MATLAB 2012a より導入された slx 形式の Simulink モデルでも本手法は適用可能である．

本論文では，MATLAB/Simulink によって設計されたエンジン制御 C コードの自動並列化手法と，自動並列化によるマルチコア上での並列処理性能評価について述べる．具体的には，エンジン制御 Simulink モデルから Embedded Coder により変換した C コードを OSCAR 並列化コンパイラで，マルチコア用の並列コードに自動的に変換するための方式の開発およびその並列処理性能評価を行った．また，本並列化手法ではコンパイラが算出した並列度に適したコア数を使った並列化プログラム生成する．特に次世代の車載チップでは初めの段階として，2 から 4 コアの構成が想定されるため，2 から 4 コアをターゲットとして並列化プログラムを生成する．これにより本論文の手法で

- エンジン制御特有の処理負荷の大きなループ構造がなく，ほとんどが条件分岐と算術代入文で構成されるアプリケーションの自動並列化・高速化
- Simulink 関連ツールと依存がなく，Simulink モデルより Embedded Coder から自動生成される C コードレベルでの自動並列化・高速化

に成功した．本論文で提案する手法では，MATLAB/Simulink により設計されたエンジン制御 C コードに対して，OSCAR コンパイラを用いて，ソースコード全域から並列性を抽出するため，Simulink 関連ツールに依存がなく，モデルの最適化とは独立して，MATLAB/Simulink から自動生成されるエンジン制御 C コードのマルチコア上での負荷分散，レイテンシの削減を自動的に行うことができる．また，OSCAR API [14] を用いることによって，任意のマルチコアアーキテクチャで動作可能となるコード生成をサポートすることができる．

## 2. OSCAR コンパイラ

本章では，OSCAR コンパイラによるマルチグレイン並列処理技術 [15], [16] について述べる．マルチグレイン並列処理とは，ループやサブルーチン等の粗粒度タスク間の並列処理を利用する粗粒度タスク並列処理，ループイタレーションレベルの並列処理である中粒度並列処理，基本ブロック内部のステートメントレベルの並列性を利用する近細粒度並列処理を階層的に組み合わせてソースコード全域にわたる並列処理を行う手法である．

### 2.1 粗粒度タスク並列処理（マクロデータフロー処理）

粗粒度タスク並列処理では，ソースコードを基本ブロック（BB），繰返しブロック（RB），サブルーチンブロック（SB）の三種類の粗粒度タスク（マクロタスク（MT））に階層的に分割する [17]．MT 生成後，OSCAR コンパイラは BB，RB，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マクロフローグラフ（MFG）を生成し，さらに 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 [18], [19] を行い，並列性の抽出を行う．その後 OSCAR コンパイラは MTG 上の MT を 1 つ以上のプロセッサエレメント（PE）をグルーピングしたプロセッサグループ（PG）に割り当てる．

MFG および MTG の例を**図 1** (a) の MFG においてノードは MT を表し，実線エッジはデータ依存を，点線エッジはコントロールフローを表す．図中の BB 中の小円が条件分岐を示している．図 1 (b) の MTG におけるノードも MFG 同様 MT を表し，MT 内の小円は条件分岐を表す．また，実線エッジはデータ依存を表し，点線エッジは拡張されたコントロール依存を表す．拡張されたコントロール依存とは，通常のコントロール依存，データ依存とコントロール依存を複合的に満足させるため，先行ノードが実行されることを確定する条件分岐を含んでいる．また，エッジを束ねるアークには 2 つの意味があり，実線アークはアークによって束ねられたエッジが AND 関係にあることを，点線アークは束ねられたエッジが OR 関係にあることを示している．

© 2014 Information Processing Society of Japan

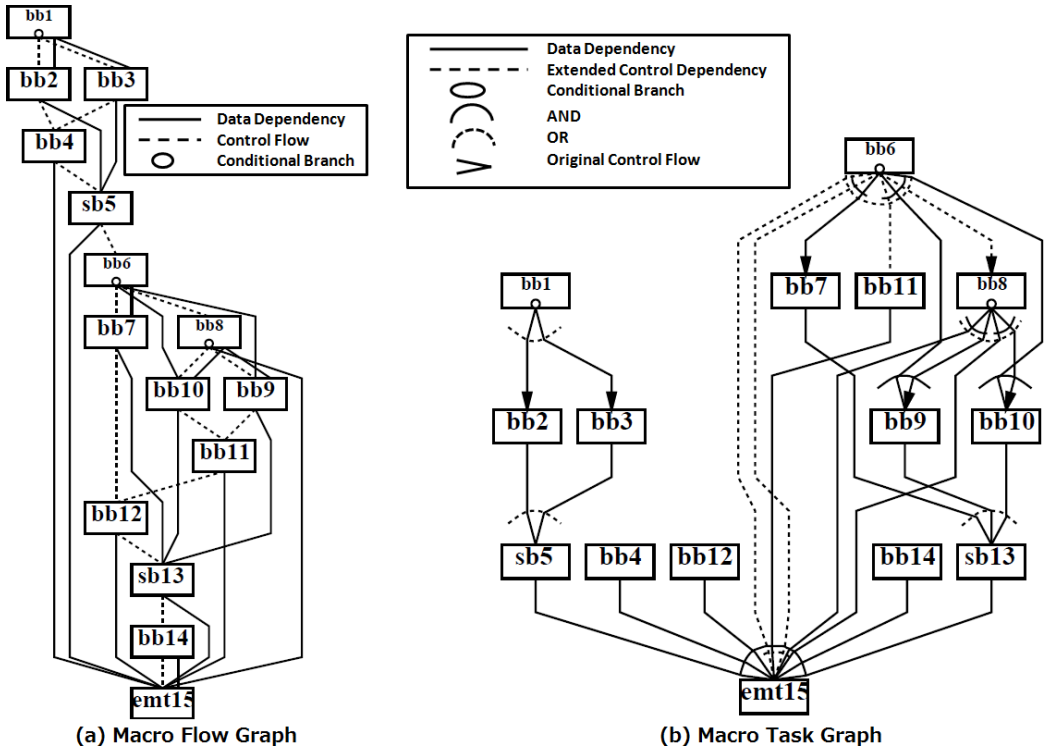



図 1 リネーミング前の MFG，MTG とリネーミング後の MTG
**Fig. 1** MFG and MTG before renaming, and MTG after renaming.

### 2.2 マクロタスクスケジューリング

粗粒度タスク並列処理では，各階層で生成されたマクロタスクは PG に割り当てられて実行される．どの PG にマクロタスクを割り当てるかを決定するスケジューリング手法として，実行時に実行タスクの割当てを行うダイナミックスケジューリングと OSCAR コンパイラが解析時に静的に実行タスクの割当てを行うスタティックスケジューリングがあり，マクロタスクグラフの形状，実行時非決定性等を元に選択される．通常，マクロタスクグラフが条件分岐等の実行時不確定性を持つ場合は，実行時にタスクの割当てを行うダイナミックスケジューリングを使用する必要がある．

### 2.3 並列化コードの生成

OSCAR コンパイラはソースコードに対して，自動並列化後，並列化 C コードを生成する．このとき，様々な共有メモリタイプのマルチコアに適用可能にするため，OpenMP [20] のサブセットをベースとした OSCAR API を用いる．特にスレッド生成系では OpenMP ディレクティブ “parallel sections” によってプログラムの実行開始時に一度だけ並列スレッドを生成するワンタイムシングルレベルスレッド生成手法 [21] を用いることでスレッド生成のオーバヘッド削減を実現している．このスレッド間で共有するデータが存在する際は，共有変数を用いて，フラグセットとフラグチェックによる同期処理を行うコードを挿入する．また，OSCAR API では OpenMP 互換のスレッド生成だけでなく，OSCAR コンパイラで拡張したメモリ管理，データ転送，電力制御のような指示文を持っている [14]．さらに，この OSCAR API を持った並列化コードは OSCAR API 標準解釈系 [22] により OSCAR API の各指示文が，ターゲットのマルチコアのマシンで利用している並列ランタイムライブラリの呼び出しコードに変換される．変換後のソースコードは既存の逐次コンパイラでコンパイルされ，前述のランタイムライブラリとリンクすることで並列実行バイナリが得られる．たとえば，POSIX 環境であれば，OSCAR API 記述のスレッド指示文が，逐次コンパイラで動作できるように pthread を用いたスレッド命令に変換される．

## 3. MATLAB/Simulink で自動生成されたエンジン制御 C コードの並列化手法

本章では MATLAB/Simulink で自動生成されたエンジン制御 C コードの特徴について述べる．次に MATLAB/Simulink で自動生成されたエンジン制御 C コードの特徴を考慮した並列化手法について述べる．

### 3.1 MATLAB/Simulink で自動生成されたエンジン制御 C コードの特徴

エンジン制御モデルで特徴的なサンプルモデルを図 **2** に，このモデルに対して，Embedded Coder により変換された C コードを図 **3** に示す．また，この C コードに対して，OSCAR コンパイラが解析した C コードの MFG と

© 2014 Information Processing Society of Japan

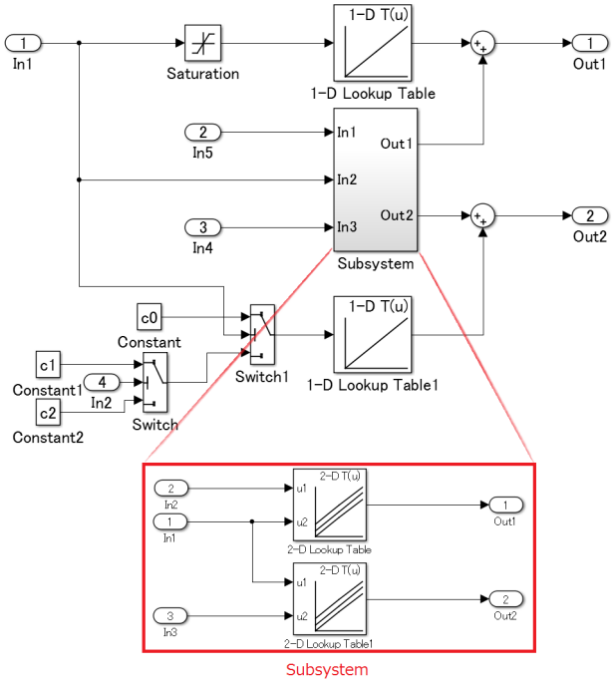



図 2 エンジン制御モデルで特徴的なサンプルモデル
**Fig. 2** Sample model which Engine Control Model has.

MTG を**図 5** に示す．

図 2 の Simulink モデル全体のアルゴリズムが図 3 の Model_step 関数に相当する．図 2 の Saturation ブロックは入力信号に上限と下限を与えるブロック，Switch ブロックは入力値に応じて，出力を切り替えるブロックで，これらは図 3 の 5 から 11 行目と 22 から 28 行目の条件分岐文に変換される．また，図 2 の Lookup Table ブロックはルックアップテーブルを使用して非線形性をモデル化するブロックで，入力値を出力値に近似する関数に変換される．具体的には，図 2 の 1-D Lookup Table ブロックが図 3 の 13，30 行目の look1_binlxpw 関数に，図 2 の Subsystem 内の 2-D Lookup Table ブロックが図 3 の 15，19 行目の look2_binlxp 関数に変換される．

また，図 3 の生成された C コードには多くの箇所で一時変数を繰り返し再利用する特徴を持つ．たとえば，rtb_DLookupTable1_n のようなローカル変数は至るところで繰り返し再利用されている．しかし，並列処理の観点では一時変数を繰り返し再利用すると，ソースコードの並列性解析の際にデータ依存があると解析され，並列性が十分引き出せないという問題がある．そのため，そのままでは図 5 (b) の MTG のように十分な並列性を抽出できていない．たとえば，図 2 の Simulink モデル上ではサブシステム内部も含め，4 つの Lookup Table が各々並列に見える．一方で，図 5 の sb8，sb9，sb11，sb19 が Lookup Table に該当する近似関数であるが，図 5 (b) の MTG では bb10 等の先行 MT からのデータ依存があり，sb11，sb19 が sb8，sb9 と並列ではない．

また，MATLAB/Simulink によってモデルベース設計された C コードにはループである RB がなく，多くが BB であり，条件分岐が多数存在する．実際に，4 章で後述するエンジン制御 C コードにはループである RB が存在せず，ほとんどが条件分岐に関連する BB である．また，各 BB は組み込みプロセッサ上で数十クロックサイクル程度であり，実行粒度は小さい．

以上のように粒度大きな並列性が乏しいプログラムから，最大限の並列性を抽出し，タスク実行時間とプロセッサ間での通信オーバヘッドを考慮した並列処理が求められる．

```
1   void Model_step(void)
2   {
3     real_T rtb_DLookupTable1_n;
4     real_T rtb_DLookupTable1;
5     if (Model_U.In1 >= Model_P.Saturation_UpperSat) {
6       rtb_DLookupTable1_n = Model_P.Saturation_UpperSat;
7     } else if (Model_U.In1 <= Model_P.Saturation_LowerSat) {
8       rtb_DLookupTable1_n = Model_P.Saturation_LowerSat;
9     } else {
10      rtb_DLookupTable1_n = Model_U.In1;
11    }
12
13    rtb_DLookupTable1_n = look1_binlxpw(rtb_DLookupTable1_n,
14      Model_P.DLookupTable_bp01Data, Model_P.DLookupTable_tableData, 2U);
15    rtb_DLookupTable1 = look2_binlxpw(Model_U.In1, Model_U.In5,
16      Model_P.DLookupTable_bp01Data_e, Model_P.DLookupTable_bp02Data,
17      Model_P.DLookupTable_tableData_c, Model_P.DLookupTable_maxIndex, 3U);
18    Model_Y.Out1 = rtb_DLookupTable1_n + rtb_DLookupTable1;
19    rtb_DLookupTable1 = look2_binlxpw(Model_U.In5, Model_U.In4,
20      Model_P.DLookupTable1_bp01Data, Model_P.DLookupTable1_bp02Data,
21      Model_P.DLookupTable1_tableData, Model_P.DLookupTable1_maxIndex, 3U);
22    if (Model_U.In1 >= Model_P.Switch1_Threshold) {
23      rtb_DLookupTable1_n = Model_P.Constant_Value;
24    } else if (Model_U.In2 >= Model_P.Switch_Threshold) {
25      rtb_DLookupTable1_n = Model_P.Constant1_Value;
26    } else {
27      rtb_DLookupTable1_n = Model_P.Constant2_Value;
28    }
29
30    rtb_DLookupTable1_n = look1_binlxpw(rtb_DLookupTable1_n,
31      Model_P.DLookupTable1_bp01Data_o, Model_P.DLookupTable1_tableData_g,
    1U);
32    Model_Y.Out2 = rtb_DLookupTable1 + rtb_DLookupTable1_n;
```

図 3 Embedded Coder により変換されたサンプル C コード
**Fig. 3** Sample C code generated by Embedded Coder.

© 2014 Information Processing Society of Japan

```
1   void Model_step(void)
2   {
3     real_T rtb_DLookupTable1_n,   rtb_DLookupTable1_n_2;
4     real_T rtb_DLookupTable1;
5     if (Model_U.In1 >= Model_P.Saturation_UpperSat) {
6     rtb_DLookupTable1_n = Model_P.Saturation_UpperSat;
7     } else if (Model_U.In1 <= Model_P.Saturation_LowerSat) {
8     rtb_DLookupTable1_n = Model_P.Saturation_LowerSat;
9     } else {
10    rtb_DLookupTable1_n = Model_U.In1;
11    }
12
13   rtb_DLookupTable1_n = look1_binlxpw(rtb_DLookupTable1_n,
14       Model_P.DLookupTable_bp01Data, Model_P.DLookupTable_tableData, 2U);
15    rtb_DLookupTable1 = look2_binlxpw(Model_U.In1, Model_U.In5,
16       Model_P.DLookupTable_bp01Data_e, Model_P.DLookupTable_bp02Data,
17       Model_P.DLookupTable_tableData_c, Model_P.DLookupTable_maxIndex, 3U);
18    Model_Y.Out1 = rtb_DLookupTable1_n + rtb_DLookupTable1;
19    rtb_DLookupTable1 = look2_binlxpw(Model_U.In5, Model_U.In4,
20       Model_P.DLookupTable1_bp01Data, Model_P.DLookupTable1_bp02Data,
21       Model_P.DLookupTable1_tableData, Model_P.DLookupTable1_maxIndex, 3U);
22    if (Model_U.In1 >= Model_P.Switch1_Threshold) {
23     rtb_DLookupTable1_n_2 = Model_P.Constant_Value;
24    } else if (Model_U.In2 >= Model_P.Switch_Threshold) {
25     rtb_DLookupTable1_n_2 = Model_P.Constant1_Value;
26    } else {
27     rtb_DLookupTable1_n_2 = Model_P.Constant2_Value;
28    }
29
30   rtb_DLookupTable1_n_2 = look1_binlxpw(rtb_DLookupTable1_n_2,
31       Model_P.DLookupTable1_bp01Data_o, Model_P.DLookupTable1_tableData_g, 1U);
32    Model_Y.Out2 = rtb_DLookupTable1 + rtb_DLookupTable1_n_2;
33  }
```

図 4　リネーミング後サンプル C コード
**Fig. 4**　Sample C code using renaming variable.

### 3.2　MATLAB/Simulink で自動生成されたエンジン制御 C コードの並列化手法

MATLAB/Simulink で自動生成された C コードは上記のようにループを持たず，粒度の小さいタスクで構成されるため，本論文では粗粒度タスク並列処理を使用する．

粗粒度タスク並列処理において，プログラムの意味上から最大限の並列性を引き出すために，OSCAR コンパイラにて図 3 の rtb_DLookupTable1_n のような再利用される一時変数を**図 4** のようにリネーミングを行う．リネーミングを行うことで，逆依存および出力依存を解消することができ，図 5 (c) のような MTG が得られ，多くの並列性が抽出できる．

また，各プロセッサコアへのマクロタスクスケジューリングにあたっては，本 C コードは多数の条件分岐を持ち，実行時不確定性が存在するため，スタティックスケジューリングが適用できず，ダイナミックスケジューリングを適用しなければならない．しかし，各実行タスクのコストはたかだか数十クロックサイクルから百クロックサイクル程度であり，OSCAR コンパイラの生成するダイナミックスケジューリングは各実行タスクにつき，通常数百クロックサイクルのオーバヘッドを要するため，ダイナミックスケジューリングの相対的なオーバヘッドが高くなってしまう．そのため，従来の OSCAR コンパイラでは本論文で対象とする条件分岐を多数持ち，タスク粒度が細かいエンジン制御 C コードを高速化することができなく，ダイナミックスケジューリングを適用すると，スケジューリングオーバヘッドが大きくなり遅くなってしまう問題がある．

そこで，実行時スケジューリングオーバヘッドが必要ないスタティックスケジューリングを適用できるように，条件分岐を持つタスクとその分岐先のタスクまでを 1 つの粗粒度タスクに融合するタスク融合を行う．また，OSCAR コンパイラではスタティックスケジューリングの際，タスク実行コストを考慮してスケジューリングするため，逐次実行コストのプロファイリング結果を活用する．

#### 3.2.1　条件分岐隠蔽のためのタスク融合

エンジン制御 C コードに対してスタティックスケジューリングを適用するために，MTG 上から条件分岐すなわちコントロール依存をすべて隠蔽し，データ依存エッジのみの MTG を作る必要がある．下記に条件分岐隠蔽のためのタスク融合のアルゴリズムを示す．本 C コードではすべてのタスク粒度が小さいため，すべての条件分岐を隠蔽することによって，依存関係を損なわない範囲で粒度を大きくした MT を生成することが可能である．

step1　マクロフローグラフの出口ノードから入口ノードの方向に各パスをスキャンし，条件分岐を含む BB を検出する．検出しなければ，step5 へ
step2　step1 で検出された条件分岐を含む BB とその BB から出力された複数のコントロールフローが集約する BB までの連続した MT を 1 つの融合ブロック（block）として再定義する．
step3　MFG を再構築する．
step4　step1 へ
step5　終了

**図 6** に条件分岐隠蔽のためのタスク融合適用例を示す．図 6 (a) はタスク融合適用前の MFG で，bb1，bb3，bb12，bb14 内の小円が条件分岐である．これらの条件分岐隠蔽のために上記のタスク融合アルゴリズムを適用すると，初めに step1 により出口ノードすなわち emt21 からスキャンしていき，図 6 (a) の小円を持つ bb14 が条件分岐を含む BB

© 2014 Information Processing Society of Japan

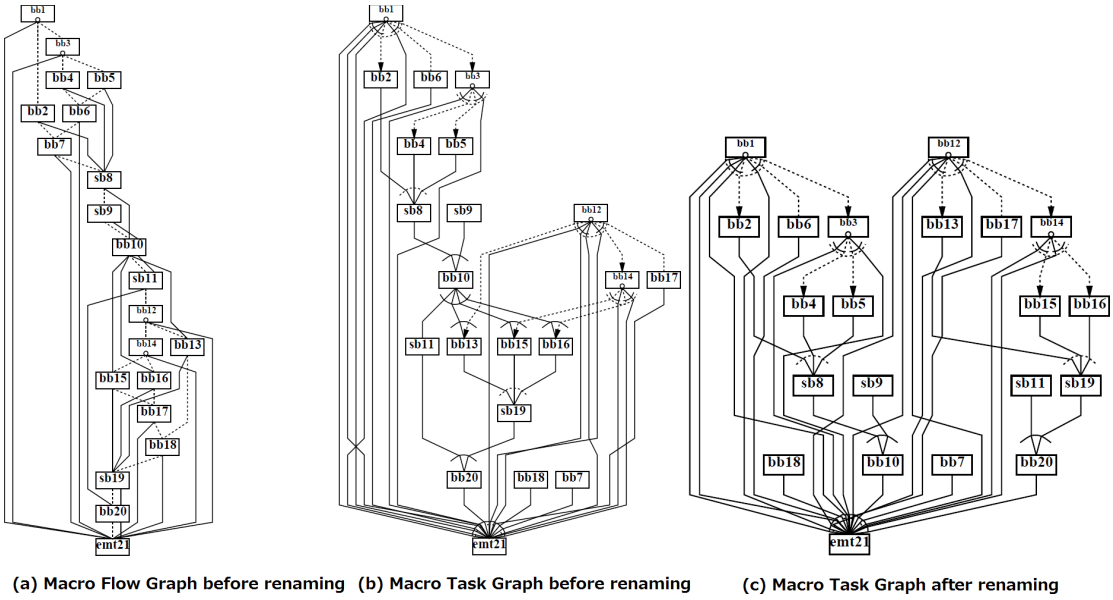



図 5 リネーミング前の MFG，MTG とリネーミング後の MTG
**Fig. 5** MFG and MTG before renaming, and MTG after renaming.

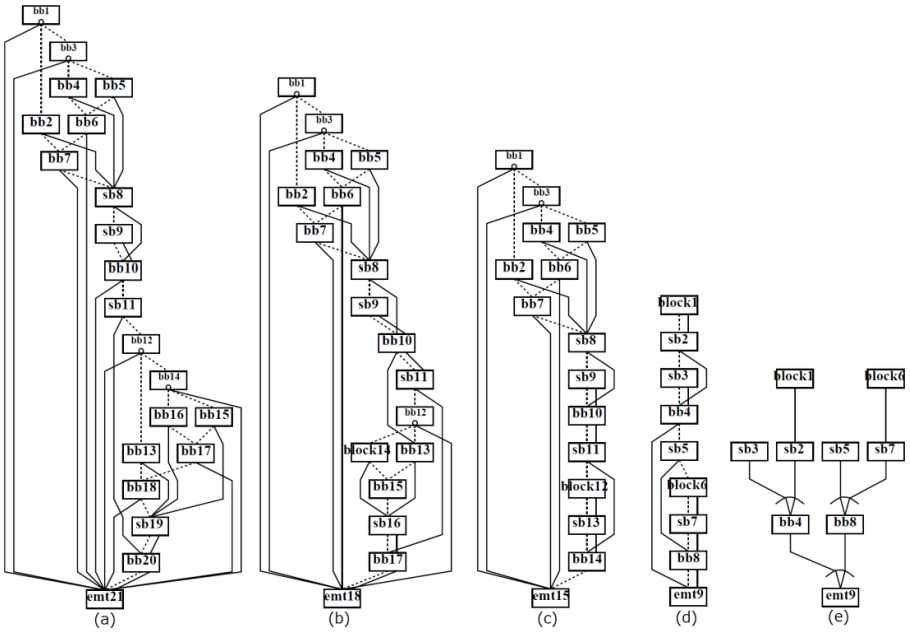



図 6 タスク融合適用例
**Fig. 6** Example of Macro-Flow Graph and Macro-Task Graph before task fusion.

として判定される．step2 により bb14 から出力された複数のコントロールフローが集約する MT として bb17 が判定され，bb14，bb15，bb16，bb17 の連続した MT が 1 つの融合ブロックとして再定義される．step3 で図 6 (b) の MFG が形成される．図 6 (b) の block14 がタスク融合を適用した MT である．step4 で step1 に戻る．次に図 6 (b) に対して，step1 により emt18 からスキャンしていき，bb12 が条件分岐を持つ BB として判定される．step2 により bb12 から出力された複数のコントロールフローが集約する MT として bb15 が判定され，bb12，bb13，block14，bb15 の連続した MT が 1 つの融合ブロックとして再定義される．step3 で図 6 (c) の MFG が形成される．同様にして，図 6 (c) の bb1 から bb7 の中で 2 度タスク融合され，図 6 (d) の MFG が形成される．step1 で条件分岐を含む BB を検出しないため，step5 に飛び，終了となる．

上記の条件分岐隠蔽のためのタスク融合を行うことで，図 6 (d) のような MFG 上において，条件分岐がない MFG に変形できる．この MFG に対して，最早実行可能条件解析を行うことにより図 6 (e) に示すコントロール依存がない MTG が形成される．このようにしてコントロール依存がなく，データ依存エッジのみの MTG を実現できたため，スタティックスケジューリングが適用可能となる．

© 2014 Information Processing Society of Japan

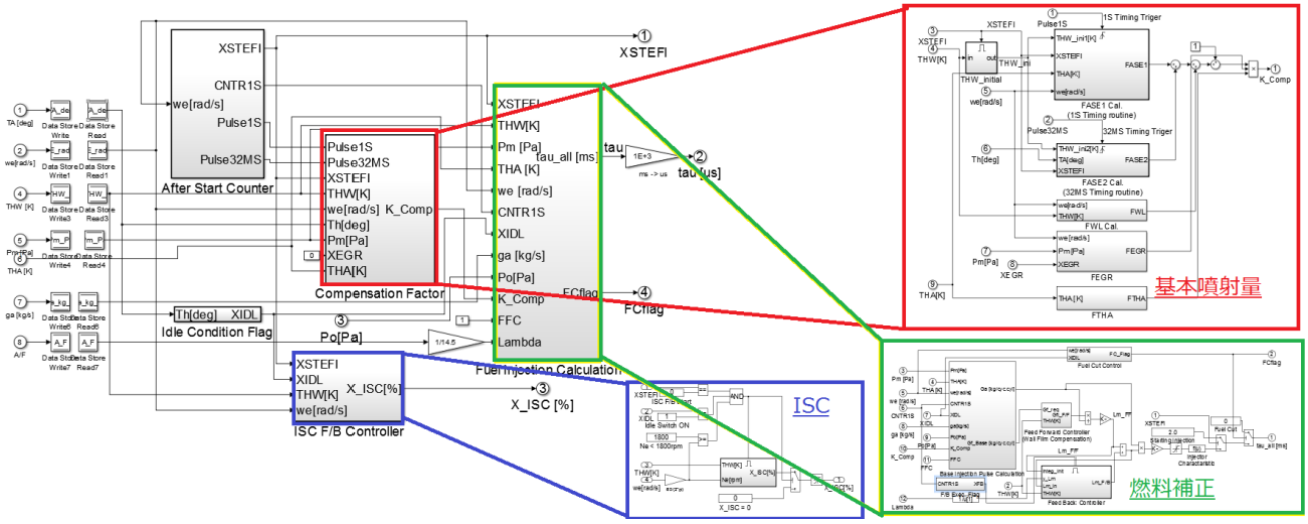



図 7　エンジン制御基本モデルとサブシステムモデル

**Fig. 7**　Basic model of engine control and subsystem model.

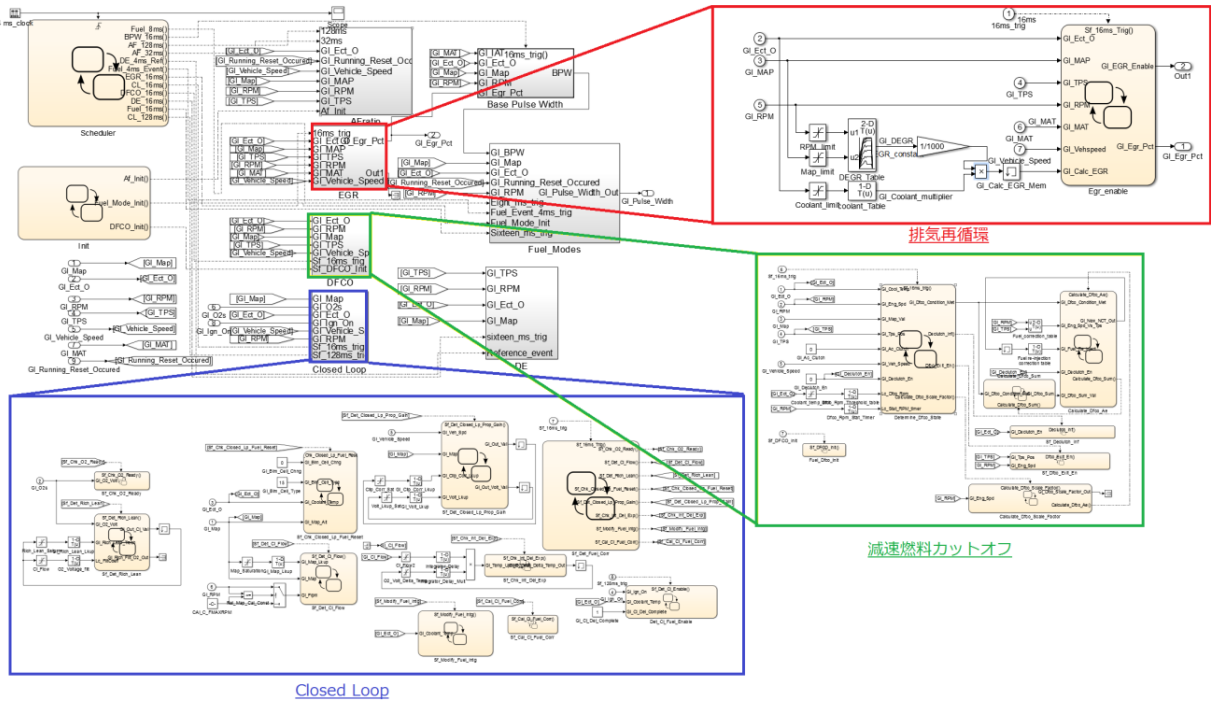



図 8　エンジン燃料噴射制御モデル

**Fig. 8**　Model of engine fuel control.

また，図 6 (e) の block1 が図 2 の Saturation ブロック，sb2 が図 2 の 1-D Lookup Table ブロック，sb3 が図 2 のサブシステム内部の 2-D Lookup Table ブロック，bb4 が図 2 の上段の加算ブロック，sb5 が図 2 のサブシステム内部の 2-D Lookup Table1 ブロック，block6 が図 2 の Switch ブロックと Switch1 ブロック，sb7 が図 2 の 1-D Lookup Table1 ブロック，bb8 が図 2 の下段の加算ブロックを示す．図 2 では Saturation ブロックと 2 つの Switch ブロック，各 Lookup Table ブロック，各加算ブロックのそれぞれが Simulink モデル上並列であるため，ソースコードレベルで図 2 のサブシステム内の並列性を含めた Simulink モデルと同等の並列性を引き出すことを実現した．また，図 6 (a) でコントロールフローを抽出した上で，タスク融合を行うことにより，コントロールフローの条件を満たしつつ，図 6 (e) の block1 と block6 のような分岐間の並列性を抽出することを可能とした．図 2 ではサンプルモデルのため，手動による並列性の抽出は容易であるが，後述する**図 7**，**図 8** のように複雑で階層的に定義され，手動で並列性解析ができないような Simulink モデルに対しても，コンパイラでは代入文や分岐や関数等の間で並列性の抽出が瞬時に可能である．

© 2014 Information Processing Society of Japan

## 4. 性能評価

本章では評価対象 Simulink モデルの概要について述べ，性能評価に用いる 2 種類の組み込みマルチコアプロセッサについて述べる．次に，各マルチコア環境でモデルベース設計されたエンジン制御 C コードの並列処理性能評価について述べる．現在，自動車のプロセッサが 2 コアへの移行の段階のため，2 コアでの並列化コードの生成が求められているが，プログラムの並列度に応じて 4 コアまでの並列処理性能を評価する．

### 4.1 評価対象 Simulink モデル

評価対象 Simulink モデルは株式会社デンソー提供の図 7 に示すエンジン制御基本モデルと図 8 に示すエンジン燃料噴射制御モデルの 2 つである．これらのエンジン制御モデルでは，スロットル開度やクランクの回転角，吸気温の排気中の酸素温度等各種センサからの入力データ情報をもとに，燃料の噴射量の計算等を行う．

図 7 のモデルでは基本噴射量を制御する Fuel Injection Calculation と燃料補正を行う Compensation Factor とアイドル時の制御を行う ISC（Idle State Controller）F/B Controller とアイドル時の判定を行う idle Condition Flag 等のサブシステムから構成されている．中でも使用されるブロック中で実行負荷割合の高い Lookup Table は制御全体で 20 個存在し，全体の約 30%の実行サイクルを占めている．

並列性に関しては，Fuel Injection Calculation や ISC（Idle State Controller）F/B Controller のようなサブシステムどうしが並列であるが，ISC の制御は車がアイドル時に実行されるモデルで，基本噴射量モデルと比較して，実行負荷が小さい．また，Compensation Factor のサブシステムと各サブシステムとの間には並列性がない．そのため，図 7 では上位のサブシステムレベルでの並列処理が有効ではない．図 7 のエンジン制御基本モデルは OSCAR コンパイラにてクリティカルパス長と全体のタスク長から算出された並列度 [23] が 2.1 と推定されたため，本評価では 2 コアまでの並列化を行う．

また，図 8 のエンジン燃料噴射制御モデルでは，排気再循環を行う EGR や減速燃料カットオフを行う DFCO や空燃比等に基づく燃料制御を行う AFratio や閉ループな制御を行う Closed loop 等から構成される．中でも使用されるブロック中で実行負荷割合が高い Lookup Table は全体 40 個あり，全体の約 60%の実行サイクルを占めている．

並列性に関しては，EGR や DFCO や Closed Loop のようなサブシステム間に並列性があるように見える．しかしながら，各サブシステムで行う処理内容および規模が異なる．たとえば，使用されるブロック中で実行負荷割合の高い Lookup Table が EGR では 2 個，DFCO では 4 個，Closed Loop では 7 個，AFratio では 10 個以上存在し，サブシステム間で並列化すると負荷バランスに偏りが生じる．またサブシステム毎でコード生成して並列化する場合，サブシステム内部で並列化することになる．したがって，より効果的な並列処理性能を最大限引き出すためには，コードを一括で生成し，サブシステムレベルのみならず，サブシステム内の並列性を階層的に活用し，プロセッサコアへの実行負荷を均衡させる必要性がある．図 8 のエンジン燃料噴射制御モデルでは OSCAR コンパイラにて並列度が 4.3 と判定されたため，本評価では 4 コアまでの並列化を行う．

本論文では図 7 と図 8 の Simulink モデル全体から Embedded Coder により C コードを自動生成する．本論文では，プログラム全域から大きな並列性を抽出するため，制御モデル全体をインライン化して，一括コード生成を行う．生成された C コードを OSCAR コンパイラにより 3 章で述べた手法を使って自動並列化を行う．さらには OSCAR API 標準解釈系を用いて，逐次コンパイラで並列実行バイナリを生成できるように，ランタイムライブラリで構成された並列化 C コードに変換する．

### 4.2 評価環境

本章のエンジン制御 C コードの並列処理性能評価では，下記のルネサスエレクトロニクス/日立/早稲田大学で開発した情報家電用マルチコア RP2 および，車載用マルチコア V850E2R 上を用いる．各プログラムの入力データは米国，カナダ，オーストラリアにおける排ガス測定用の走行パターンのうち市街地想定（LA#4 City [24]）走行パターンをトレースするように速度が制御されているテストデータを用いる．本論文では OS はスレッド生成とコアのバインディングのみを行う簡易的なものを用い，ワンタイムシングルレベルスレッド生成手法により，単一ソフトウェアを複数 CPU コアで動作させる．また，データを共有する際には OS での排他制御を行わず，OSCAR コンパイラより生成された 1 対 1 同期処理をプログラム中で行うことにより，データのアクセス順番の保持を行う．

#### 4.2.1 情報用家電マルチコア RP2

情報家電用 RP2 のブロック図を**図 9** に示す．RP2 は SH4A コアを 8 基搭載したホモジニアスマルチコアである．4.2.2 項で後述するように，市販化されている車載用マルチコアが 2 コアであり，次世代の車載マルチコアが 2 コアから 4 コアであることを想定して，RP2 上では 2 コアから 4 コアまでを評価で用いる．また，各コアには，中央演算処理装置（CPU），ローカルデータメモリ（OLRAM），分散共有メモリ（URAM）を持つ．分散共有メモリは各コアがローカルに持つが，他コアからもデータの書き込みができ，同期やデータ転送を低オーバヘッドで行うことができる．チップ上のすべてのコアは，SHwy バスによってオン

© 2014 Information Processing Society of Japan

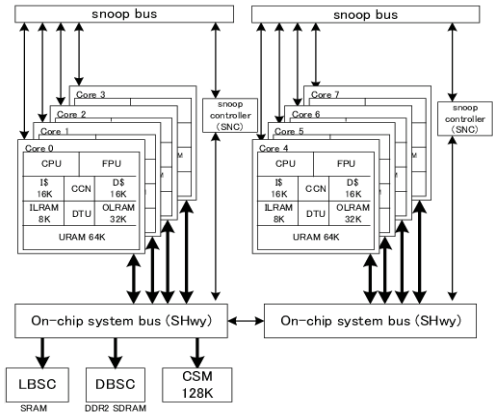



図 9 情報家電用マルチコア RP2 の構造
**Fig. 9** Architecture of RP2.

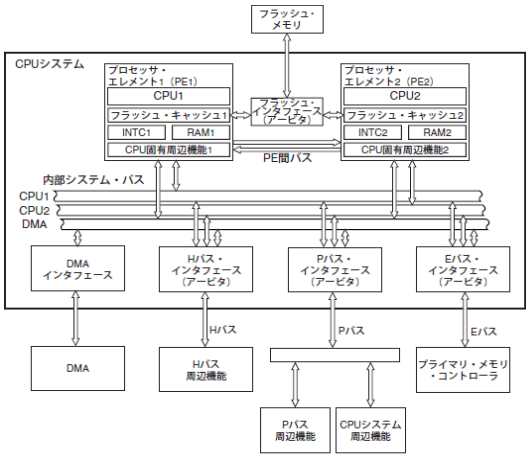



図 10 V850E2R の構造
**Fig. 10** Architecture of V850E2R.

チップ集中共有メモリ（CSM）やオフチップ集中共有メモリ（DDR2 SDRAM）に接続されている．

RP2 上での評価ではメモリアクセスオーバヘッドを削減するために，入力データ等はそれぞれのローカルデータメモリに配置する．また，スタティックスケジューリング時の MT 間の同期情報の授受には，分散共有メモリを使用する．分散共有メモリの使用により，同期フラグチェックによるビジーウェイトが SHwy バスを使用せずに PE 内部で行われるため，効率の良い同期やスケジューリング情報の授受が可能である．ほかはオンチップ集中共有メモリに配置し，SHwy バスを使い，供給を行う．また，RP2 上では逐次コンパイラとして SH C コンパイラを用いる．

#### 4.2.2 V850E2R

V850 プロセッサをベースとしたマルチコア V850E2R は**図 10** に示すような市販化されている車載用のアーキテクチャ [25] である．V850E2R マルチコアは 2 コアでそれぞれの CPU において，他の CPU から PE 間バスを経由して，アクセス可能な分散共有メモリ（RAM1，RAM2）を持ち，数サイクルの低レイテンシなデータアクセスが可能である．

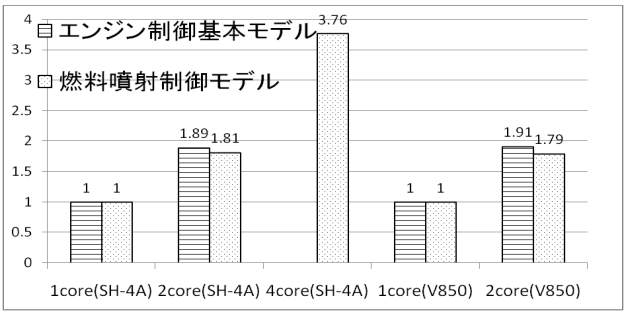



図 11 各環境でのモデルベース設計された自動車制御ソフトウェアの並列処理性能評価結果
**Fig. 11** Evaluation of automotive control software developed by Model-Based Design on multi-core embedded processors.

特に入力データに関しては，各 RAM に配置し，PE 間バスアクセスが生じないようにする．同期命令や他のデータは PE 間バスアクセスを最小化するように各 RAM に分散して配置を行う．また，命令に関してはフラッシュメモリとキャッシュメモリを利用しながら命令供給を行う．フラッシュメモリは共有で，両 PE に命令供給するが，命令キャッシュメモリは独立である．また，V850 上では逐次コンパイラとして Green Hills コンパイラを用いる．

### 4.3 組み込みマルチコアプロセッサ上での評価結果

RP2 や V850E2R 上で図 7 と図 8 の株式会社デンソー提供の Simulink モデルより生成されたエンジン制御 C コードの OSCAR コンパイラを用いた並列処理性能を評価し，3.2 節で提案した並列化手法の有効性を示す．3.2 節で提案した並列化手法であるタスク融合用いずに，ダイナミックスケジューリングで図 7 と図 8 の Simulink モデルより生成されたエンジン制御 C コードを並列化すると，ともに 10 倍以上の速度劣化が得られている．これは数十クロックサイクルから百クロックサイクル程度の粒度の小さな各 MT に対して，実行時にスケジューリングすることで，スケジューリングオーバヘッドが相対的に大きくなってしまうからである．

**図 11** に 3.2 節の並列化手法を用いた各環境でのモデルベース設計された自動車エンジン制御 C コードの並列処理性能評価結果を示す．エンジン制御基本モデルに関しては，オリジナル C コードの逐次実行に比べて，OSCAR コンパイラで並列化した 2 コア用の C コードでは RP2 上で 1.89 倍の性能向上が得られた．同様に車載用マルチコアの V850E2R 上で，1.91 倍の性能向上が得られた．

また，エンジン燃料噴射制御モデルに関しては，オリジナル C コードの逐次実行に比べて，RP2 上では OSCAR コンパイラが並列化した 2 コア用のコードで 1.81 倍，4 コア用のコードで 3.76 倍の性能向上が得られた．同様に V850E2R 上で，1.79 倍の性能向上が得られた．

© 2014 Information Processing Society of Japan

図 7 と図 8 のようなサブシステムレベルで並列化が困難なモデルに対して，サブシステムレベルではなく，ソースコード全域にわたって並列性を抽出し，オーバヘッドをかけずに各コアにタスクを静的に割り当てることで，性能向上が得られ，提案する並列化手法の有効性を示すことができた．また，提案する手法によりモデルベース設計されたエンジン制御 C コードのマルチコア上での高速化が実現したため，エンジン制御におけるマルチコアの利用の可能性を示すことができた．

## 5. おわりに

本論文では MATLAB/Simulink でモデルベース設計されたエンジン制御の C コードの並列化手法を提案した．条件分岐が連続しタスク粒度の細かいエンジン制御 C コード対して，提案手法では OSCAR コンパイラを用いてリネーミングを行い，ソースコード全域にわたって並列性を抽出し，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，最早実行可能条件解析を利用して，自動並列化を行った．

さらに OSCAR API 標準解釈系を用いたコード生成を行うことにより SH および V850 用の逐次コンパイラを用いて並列オブジェクトを生成することを可能とし，組み込みマルチコアプロセッサ上での並列処理を実現した．

その結果，多くが条件分岐で粒度が細かく，マルチコア上での並列化が困難であった MATLAB/Simulink モデルベース設計されたエンジン制御 C コードにおいて，情報家電用マルチコア RP2 上と車載用マルチコア V850E2R 上で OSCAR コンパイラを用いて性能向上を得ることに成功した．エンジン制御基本モデルに関しては，RP2 上で 1.89 倍，V850E2R では 1.91 倍の性能向上が得られた．また，エンジン燃料噴射制御モデルに関しては，RP2 上で 3.76 倍，V850E2R 上で 1.79 倍の性能向上が得られた．

これにより，OSCAR コンパイラを用いることで，MATLAB/Simulink によってモデルベース設計された自動車エンジン制御 C コードの自動並列化およびその高速化が可能であることが確認できた．エンジン制御のマルチコアによる高速化によって，リアルタイム処理の制約で従来車載できなかったより高度な制御機能等の実装が可能になり，エンジン制御のさらなる高度化の可能性を示すことができた．

今後は Embedded Coder 以外の dSPACE 社の Target Link 等の別のコード生成ツールに対して，同手法の適用を試みる予定である．

### 参考文献


[1] Seo, K., Chung, T., Heo, H. and Yi, K.: Coordinated implementation and processing of a unified chassis control algorithm with multi-central processing unit, JAUTO1346 IMechE, Vol.224 Part D: J. Automobile Engineering (2009).

[2] Seo, K., Chung, T., Heo, H. and Yi, K.: An Investigation into Multi-Core Architectures to Improve a Processing Performance of the Unified Chassis Control Algorithms, *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.*, Vol.3, No.1, pp.53–62 (2010), doi:10.4271/2010-01-0662 (2010).

[3] Monot, A., Navet, N., Bavoux, B. and Simonot-Lion, F.: Multisource Software on Multicore Automotive ECUs Combining Runnable Sequencing With Task Scheduling, IEEE Trans. on Industrial Electronics, Vol.59, No.10 (2012).

[4] Renesas: SH-Navi3, available from ⟨http://japan.renesas.com/edge/vol.24/special05/index.jsp⟩.

[5] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. 13th International Workshop on Languages and Compilers for Parallel Computing* (2000).

[6] 金羽木洋平，梅田 弾，見神広紀，林 明宏，沢田光男，木村啓二，笠原博徳：自動車エンジン制御ソフトウェアにおけるマルチコア上での並列処理，Vol.2013-ARC-203, No.2 (2013).

[7] MathWorks: Simulink, available from ⟨http://www.mathworks.co.jp/products/simulink/⟩.

[8] MathWorks: Embedded Coder, available from ⟨http://www.mathworks.co.jp/products/embeddedcoder/⟩.

[9] MathWorks: SimulinkCustomize, available from ⟨http://www.mathworks.co.jp/jp/help/simulink/multicore-processor-targets-1.html⟩.

[10] 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).

[11] Cha, M., Kim, K. and Kim, K.: An Automatic Parallelization Scheme for Simulink-based Real-Time Multicore Systems, Science & Engineering Research Support Society, Vol.5 (2012).

[12] 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).

[13] Kumura, T., Nakamura, Y., Ishiura, N., Takeuchi, Y. and Imai, M.: Model Based: Parallelization from the Simulink Models and Their Sequential C Code, SASIMI (2012).

[14] Kimura, K., Mase, M., Mikami, H., Miyamoto, T. and Kasahara, J.S.H.: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers, *Proc. 22nd International Workshop on Languages and Compilers for Parallel Computing* (2009).

[15] 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* (2010).

[16] 木村啓二，小高 剛，小幡元樹，笠原博徳：OSCAR チップマルチプロセッサ上でのマルチグレイン並列処理，Arc2002-150-7, 情報処理学会 (2002).

[17] Mase, M., Onozaki, Y., Kimura, K. and Kasahara, H.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *15th Workshop on Compilers for Parallel Computing* (2010).

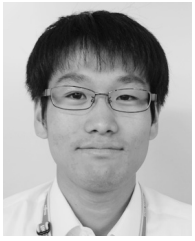
[18] 本多弘樹，岩田雅彦，笠原博徳：Fortran プログラム粗粒度タスク間の並列性検出法，信学論（D-I），Vol.J73-D-I,


© 2014 Information Processing Society of Japan

No.12, pp.951–960 (1990).
[19] 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).
[20] Open MP: Open MP, available from ⟨http://openmp.org/wp/⟩.
[21] 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).
[22] 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).
[23] 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, pp.1345–1356 (2004).
[24] EPA: Dynamometer Drive Schedules, available from ⟨http://www.epa.gov/nvfel/testing/dynamometer.htm⟩.
[25] RENESAS: V850, available from ⟨http://japan.renesas.com/products/mpumcu/v850/⟩.

**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 年同大学基幹理工研究科情報理工専攻修了．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．



**金羽木 洋平** （正会員）

1988 年生．2011 年早稲田大学基幹理工学部情報理工学科卒業．2013 年同大学基幹理工研究科情報理工専攻修了．2014 年富士通株式会社入社，現在に至る．

**見神 広紀** （正会員）

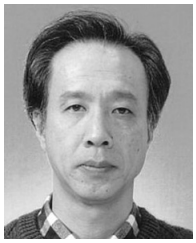
1984 年生．2009 年早稲田大学大学院基幹理工学研究科情報理工学専攻修士課程修了，2011 年同大学基幹理工学部助手，2014 年同大学グリーン・コンピューティング・システム研究機構研究助手，現在に至る．並列アプリケーション，コンパイラ，マルチコアプロセッサアーキテクチャに関する研究に従事．

**林 明宏** （正会員）

1984 年生．2007 年早稲田大学理工学部コンピュータ・ネットワーク工学科卒業．2008 年同大学大学院理工学研究科情報・ネットワーク専攻修士課程修了．2012 年同大学院基幹理工学研究科情報理工学専攻博士（工学）学位取得．2010 年同大学基幹理工学部情報理工学科助手．2012 年同助教．2013 年アメリカ合衆国 Rice University 博士研究員，現在に至る．マルチコア，アクセラレータ，スーパーコンピュータ向けのプログラミング言語およびコンパイラ・ランタイムに関する研究に従事．

**谷 充弘** （正会員）

1964 年生．1986 年愛知工業大学工学部応用化学科卒業．1986 年日立マイクロコンピュータエンジニアリング株式会社．1996 年より株式会社デンソー，エンジン，トランスミッション制御用リアルタイム OS，コンパイラの企画，品質確認，数値計算ライブラリ作成に従事．IEEE, IEEE-CS，ACM 各会員．



**森 裕司** （正会員）

1957 年生．1983 年早稲田大学大学院理工学研究科修士課程修了．同年株式会社デンソー，研究開発部門においてメカ制御系の研究開発に従事．現在，電子制御システムに関する先行技術の企画を担当，主にモデルベース開発，シミュレーション，アーキテクチャに関する開発に従事．

**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．

© 2014 Information Processing Society of Japan

**笠原 博徳** （正会員）

1957 年生．1980 年早大理工電気卒業，1985 年同博士修了，工学博士，1986 年早大理工専任講師，1997 年教授，現在，情報理工学科教授．1987 年 IFAC World Congress Young Author Prize，1997 年情報処理学会坂井記念特別，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）受賞．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等歴任．電子情報通信学会，IEEE，IEEE-CS（Computer Society），ACM 等各会員．現在，IEEE-CS 理事，マルチコア STC 委員長．

© 2014 Information Processing Society of Japan