# モデルベース設計により自動生成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 † 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†

自動車の安全性・快適性・環境適合性の向上を目指し，自動車制御系は年々より高度化している．これに伴い，制御プロセッサには高い性能が求められている．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，マルチコアへの移行が進んでいくと考えられる．しかし，マルチコアの性能を有効に引き出すようなソースコードの並列化は困難である．一方，自動車制御系において開発期間の短縮及び信頼性の向上のためにモデルベース設計が普及している。しかし，現時点でこの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．本論文ではモデルベース設計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 コードを自動並列化し，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 倍の性能向上することに成功した．これにより，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自動並列化が可能であることが確認できた．

# Automatic Parallelization of Automatically Generated Engine Control C Codes by Model-based Design

Dan Umeda† Youhei Kanehagi† Hiroki Mikami† Akihiro Hayashi† Mitsuhiro Tani††
Hiroshi Mori†† Keiji Kimura† Hironori Kasahara†

Recently, the automobile control system is advancing to achieve more safety, comfort and fuel efficiency. Accordingly, control systems need performance enhancement on microprocessors. 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. However, it is difficult to parallelize source code and attain speedup on multi-core processors. Recently Model-based Design has been used for developing automobile systems because of elimination time of development and improvement of reliability. However, auto-generated-code from Model-based Design 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 developed by Model-Based Design. The engine control C code which composed of only conditional branches and arithmetic assignment statements and are difficult to parallelize have been parallelized automatically and 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. It is confirmed that parallelization of engine control C code developed by model based design on multi-core processor is possible.

† 早稲田大学
WASEDA UNIVERY
†† 株式会社デンソー
DENSO CORPORATION

## 1. はじめに

より安全，快適，かつ省エネで環境適合性の高い次世代自動車の開発のため，エンジン制御のようなリアルタイム制御系，人認識・他車認識のような外界認識，運

転に必要な情報の提示といった，それぞれの高度化及びそれらの統合化が重要となっている．このような制御系，及び統合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，消費電力の増大と，それに伴う長期の信頼性の問題により困難である．このため，現在は車載チップのほとんどがシングルコアプロセッサであるが，今後は1チップ上に低動作周波数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．例えば，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 プロセッサで並列処理を行う．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困難である．

また，現在カーナビ用に SH4A を 2 基搭載した SH-Navi3[4]などのマルチコアを用いた白線検知などの走行支援システム等の一部で検討，導入が行われているが，現状ではエンジン制御系で実現された例は存在しない．

また，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を目的にモデルベース設計が普及している．特に，エンジン制御系では MathWorks 社の Simulink 等が使われている[5]．さらには，近年のモデルベース設計では，モデルの設計だけでなく，設計したモデルから C コードの自動生成がサポートされている．しかし，現状ではモデルから自動生成される C コードはシングルコア向けのコードで，マルチコアを十分に活用できない．前述のように，今後エンジン制御系でマルチコアプロセッサ利用への移行が必須のため，自動車分野のソフトウェア開発で主流となっている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．そこで，Simulink から生成される C コードの並列化に関する従来技術として，MathWork 社ツールでの同時実行設定のカスタマイズ[6]や user-defined Simulink blocks[7]などの並列化手法が挙げられる．しかし，これらはユーザーの指示により 1 つの Simulink モデル内のサブシステム間(機能レベル) で並列実行するものである．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，且つ各サブシステムの実行負荷が均等となっている場合のみ有効となる．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．

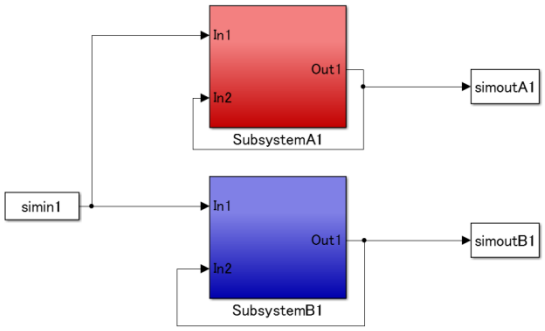


図1 サブシステムレベルで並列化が有効なモデル
Fig.1 A model which has parallelism between two subsystems

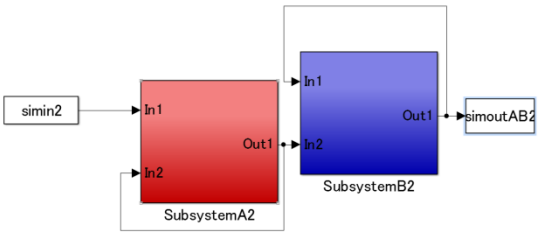


図2 サブシステムレベルで並列化による性能向上が困難なモデル
Fig.2 A model which has no parallelism between two subsystems

図 1 にサブシステムレベルで並列化が有効な Simulink モデルの例を，図 2 にサブシステムレベルで並列化による性能向上が困難な Simulink モデルの例を示す．たとえば，図1のように各サブシステムレベルで並列性が存在する場合，上記手法で並列化が可能である．しかし，並列性能向上率は各モデルの実行負荷バランスに依存してしまう．また，図 2 のように一方のサ

ブシステムからの入力があるような複数のサブシステムが逐次のSimulinkモデルでは全く性能向上が得られない．そのため，図 2 のモデルで並列化を行う場合，サブシステム内部で並列性を引き出す必要性がある．

本論文で並列化を行うエンジン制御モデルは各サブシステムレベルでの並列性が乏しく，サブシステム毎で実行負荷が異なるため，上記手法は適用困難である．仮に上記手法を適用する場合，モデルのアルゴリズムを再検討し，手作業による多くの書き換えが必要となる．

また，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通信が最小になる形で効果的にスケジューリングして，実行することが必要となる．しかし，このモデルベース設計されたエンジン制御のようなCコードにはループ構造がなく，条件分岐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．

このような状況を踏まえ，本論文では，モデルベース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，エンジン制御用Simulinkモデルから Embedded Coder[8]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及びその並列処理性能評価を行った．これにより本論文の手法で

- エンジン制御特有のループ構造がなく，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 モデルの変更がなく，Simulink Coder/Embedded Coder より Simulink モデルから自動生成されるCコードの自動並列化・高速化

に成功した．本論文で提案する手法では，モデルベース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，ソースコード全域にわたって並列性を抽出するため，サブシステム間レベルだけでなく，サブシステム内部の並列処理を行うことを可能とした．すなわち，図 1 や図 2 のように Simulink モデルの実装方法に依存なく，従来実現が困難であった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．

2 章ではOSCARコンパイラの概要について述べる．3章ではモデルベース設計されたエンジン制御Cコードの並列化手法について述べ，4 章では提案手法を用いた性能評価について述べる．

## 2. OSCAR コンパイラ

本章では，OSCAR コンパイラ[9]によるマルチグレイン並列処理技術について述べる．マルチグレイン並列処理[9][10]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)，繰り返しブロック(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[11]．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間の並列性を最早実行可能条件解析により引きだした結果をマクロタスクグラフ(MTG)として表現する[12][13]．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 及び MTG の例を図 3 に示す．図 3(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．図中の赤色のマクロタスクが BB を示し，BB 中の小円が条件分岐であり，黄色のマクロタスクが SB を示している． 図 3(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．

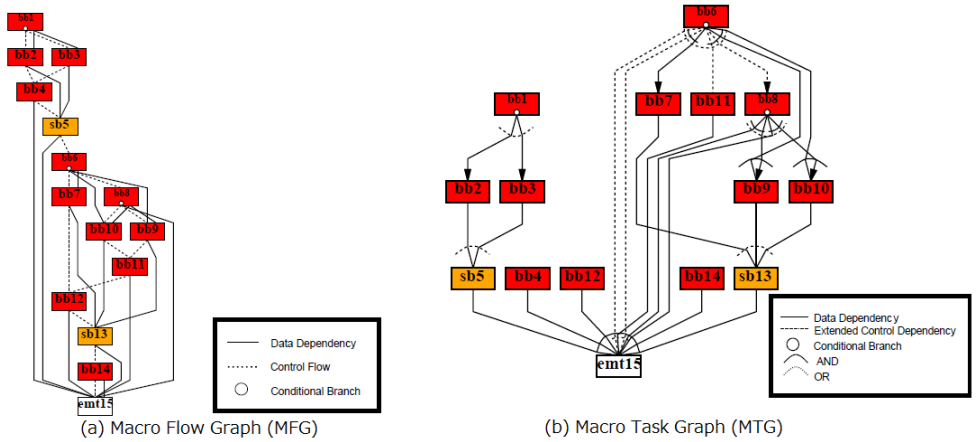


図3 マクロフローグラフ(MFG), マクロタスクグラフ(MTG)の例
Fig.3 Example of Macro-Flow Graph and Macro-Task Graph

### 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り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り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，実行時非決定性などを元に選択される．通常，マクロタスクグラフが条件分岐などの実行時不確定性を持つ場合は，実行時にマクロタスクを割り当てるダイナミックスケジューリングを用いる．

### 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[14]のサブセットをベースとした OSCAR API[15]を用いる．さらに，並列化コードは OSCAR API 標準解釈系により OSCAR API の各ディレクティブが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，ターゲットマルチコア用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．

## 3. モデルベース設計されたエンジン制御 C コードの並列化手法

本章ではまずモデルベース設計されたエンジン制御 C コードの特徴について述べる．次にモデルベース設計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. モデルベース設計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 4 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 5 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 MTG を図 6 に示す．

図 4 の Simulink モデル全体のアルゴリズムが Model_step 関数に相当する．図 4 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 5 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 4 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， 入力値を出力値にマップする関数に変換される．具体的には，図 4 の 1-D Lookup Table ブロックが図 5 の 13，30 行目の look1_binlxpw 関数に，図 4 の 2-D Lookup Table ブロックが図 5 の 15，19 行目の look2_binlxp 関数に変換される．

また，Embedded Coder により Simulink モデルから自動生成された C コードは使用メモリ量削減のため，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．例えば，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，並列処理の観点から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 6 (b)の MTG のように十分な並列性を抽出できていない．例えば，図 4 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 6 の黄色の sb8, sb9, sb11, sb19 が Lookup Table に該当するマップ関数であるが，図 6 (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．

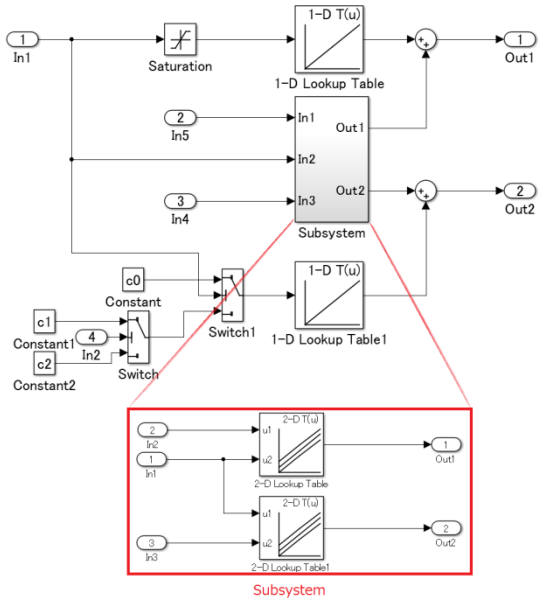


図4 エンジン制御モデルで特徴的なサンプルモデル
Fig.4 Sample model which Engine Control Model has

また，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モデルにはループである 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．以上のように条件分岐が多く，並列処理粒度が細かいため，モデルベース設計された C コードには，並列処理を困難とする要素が存在する．

### 3.2. 粗粒度タスク並列処理によるモデルベース設計されたエンジン制御 C コードの並列化手法

まず，Simulink モデルと同等の並列性を十分に引き出すために，OSCAR コンパイラにて図 5 の rtb_DLookupTable1_n のような再利用される一時変数のリネーミングを行った上で並列性解析を行う．その上で OSCAR コンパイラを使って，MFG 及び MTG を生成する．

```
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;
33  }
```

図5 Embedded Coder により変換されたサンプル C コード
Fig.5 Sample C code generated by Embedded Coder

また，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高々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，通常数十から数百クロックのオーバー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ーヘッドが高くなってしまう．そのため，本コードでは実行時にコストがかかるダイナミックスケジューリングを適用すると高速化が困難となる．そこで，本コードにスタティックスケジューリングを適用できるように，条件分岐をも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，すなわち擬似代入文ブロック

(Block)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，実行コストの見積もりの精度を向上させるために，逐次実行コストのプロファイリング結果を活用する．

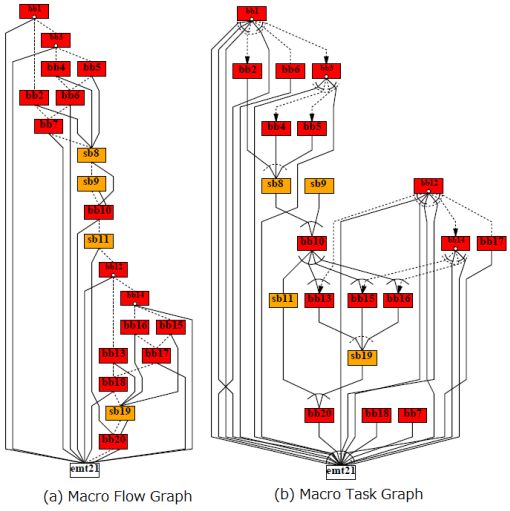


図6 サンプル C コードのマクロフローグラフ(MFG)とマクロタスクグラフ(MTG)

Fig.6 Macro Flow Graph (MFG) and Macro Task Graph (MTG) of sample C code

### 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ある程度粒度をもった MT を生成することが可能である．

step.1 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

step.2 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 を一つの擬似代入文として定義し，その擬似代入文を含むブロックを 1 つの MT(Block)として再定義する．

step.3 MFG を再構築する．

step.4 step1 へ

step.5 終了

図 7 に条件分岐隠蔽のためのタスク融合適用例を示す．図 7(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 7 (a)の小円を持つ bb14 が条件分岐を含む BB 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 つの疑似代入文ブロックとして再定義される．step3 で図 7 (b)の MFG が形成される．図 7(b)の灰色の 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 7(b)に対して，step1 により emt17 からスキャンしていき，bb12 が条件分岐をもつ 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，bb14，bb15 の連続した MT が 1 つの疑似代入文ブロックとして再定義される．step3 で図 7 (c)の MFG が形成される．同様にして，図 7 (c)の bb1 から bb7 の中で 2 度タスク融合され，図 7 (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 7 (d)のような MFG 上条件分岐がない形に変形できる．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 7 (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．また，図 7 (e)の Block1 が図 4 の Saturation ブロック， sb2 が図 4 の 1-D Lookup Table ブロック，sb3 が図 4 のサブシステム内部の 2-D Lookup Table ブロック，bb4 が図 4 の上段の加算ブロック， sb5 が図 4 のサブシステム内部の 2-D Lookup Table1 ブロック，Block6 が図 4 の Switch ブロックと Switch1 ブロック，sb7 が図 4 の 1-D Lookup Table1 ブロック， bb8 が図 4 の下段の加算ブロックを示す．図 4 では Saturation ブロックと二つの 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 4 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．

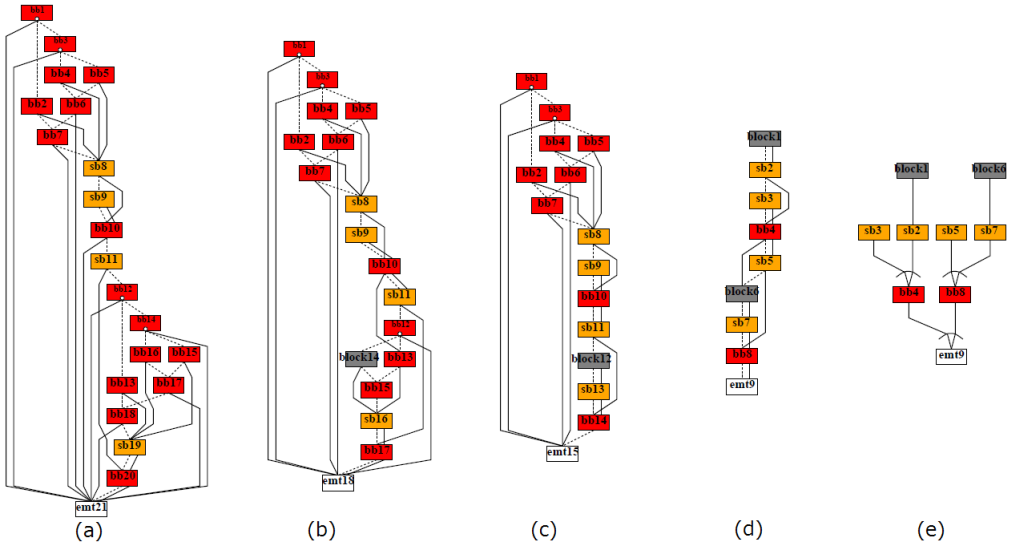


図7 タスク融合適用例

Fig.7 Example of Macro-Flow Graph and Macro-Task Graph before task fusion

## 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．

### 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8に示すエンジン制御基本モデルと図9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など各種センサー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8のモデルでは基本噴射量とISC(Idle State Controller)のようなサブシステムが並列であるが，ISC 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，燃料補正サブシステムは各サブシステムと並列ではない．そのため，図8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．

また，図 9 のエンジン燃料噴射制御モデルでは，排気再循環や減速燃料カットオフや Closed loop のようなサブシステム等が並列に見える．しかしながら，各サブシステムで行う処理内容及び規模が異なる．したがって，より効果的な並列処理性能を引き出すためには，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．

本論文では図 8 と図 9 の Simulink モデルから Embedded Coder により C コードを自動生成する．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 及び，車載用マルチコア V850E2R 上を用いる．

#### 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 10 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 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．チップ上の全てのコアは，SHwy バスによってオン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．

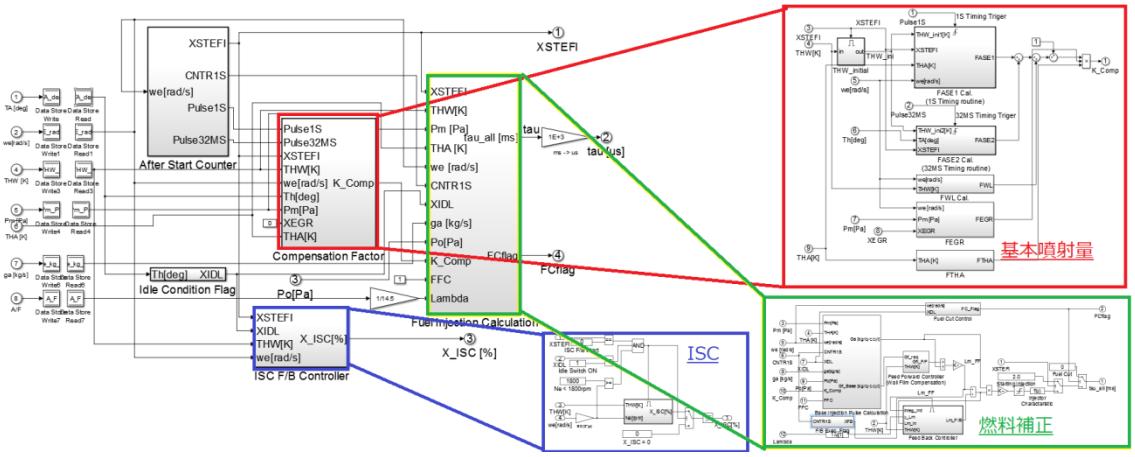


図8 エンジン制御基本モデルと燃料補正サブシステムモデル

Fig.8 Basic model of engine control and fuel injection calculation subsystem

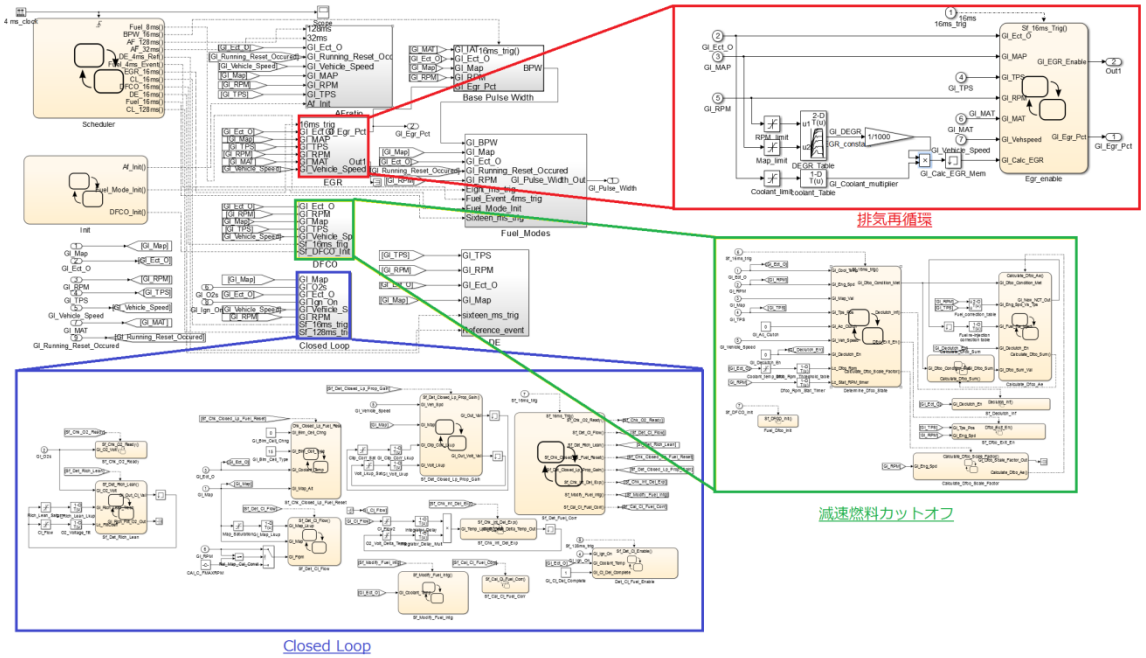


図9 エンジン燃料噴射制御モデル

Fig.9 Model of engine fuel control

RP2 上での評価ではメモリアクセスオーバーヘッドを削減するために，同期変数を分散共有メモリに配置すると共に，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，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 11 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[16]である．V850E2R マルチコアは 2 コアでそれぞれの CPU に他の CPU から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．

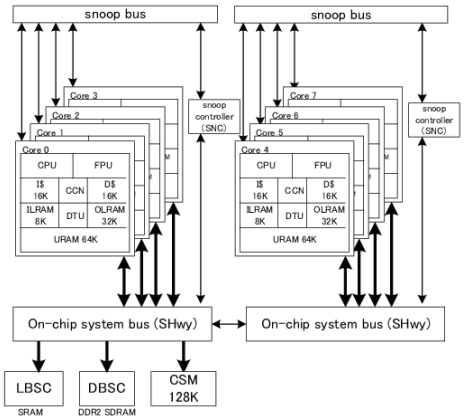


図10 情報家電用マルチコア RP2 の構造
Fig.10　Architecture of RP2

特に入力データに関しては，各 RAM に配置し，RAM のリモートアクセスの削減を行う．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する．

図 12 に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．

図 8 と図 9 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，低オーバーヘッドで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 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．

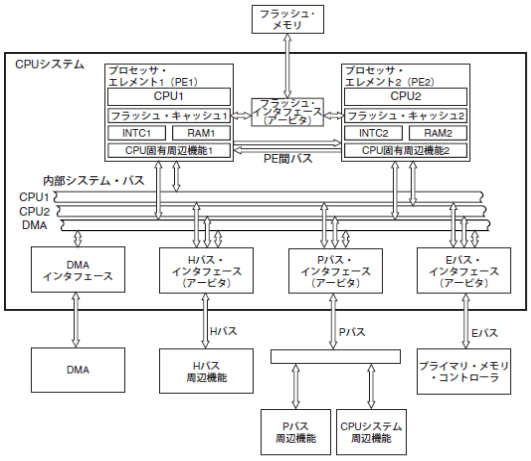


図11 V850E2R の構造
Fig.11　Architecture of V850E2R

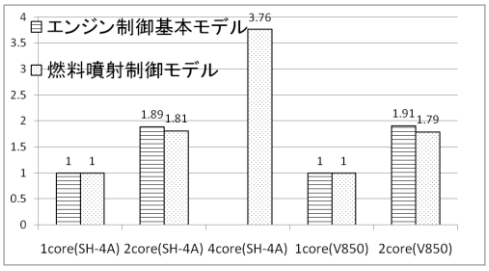


図12 各環境でのモデルベース設計された自動車制御ソフトウェアの並列処理性能評価結果
Fig.12　Evaluation of automotive control software developed by Model-Based Design on multi-core embedded processors

## 5. おわりに

本論文では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及び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，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及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．

## 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. AutomobileEngineering (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. 3(1):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 TRANSACTIONS ON INDUSTRIAL ELEC-TRONICS, VOL. 59, NO. 10, OCTOBER 2012 (2012).

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

[5] MathWorks: Simulink, http://www.mathworks.co.jp/products/simulink/.

[6] MathWorks: SimulinkCustomize, http://www.mathworks.co.jp/jp/help/simulink/multicoreprocessor-targets-1.html.

[7] Cha, M. and Kim, K.: Deriving High-Performance Real-Time Multicore Systems based on Simulink Applications, 2011 Ninth IEEE International Conference on Depend-able, Autonomic and Secure Computing. (2011).

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

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

[10] 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).

[11] Masayoshi Mase, Yuto Onozaki, Keiji Kimura and Hironori Kasahara,"Parallelizable C and Its Performance on Low Power High Performance Multicore Processors", 15th Workshop on Compilers for Parallel Computing 2010, Jul. 2010.

[12] 本多， 岩田， 笠原: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).

[13] 笠原， 合田， 吉田， 岡本， 本多: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).

[14] Open MP: Open MP，http://openmp.org/wp/.

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

[16] RENESAS: RENESAS, http://japan.renesas.com/products/mpumcu/v850/.