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

金羽木 洋平[1]　梅田 弾[1]　見神 広紀[1]　林 明宏[1]　沢田 光男[2]　木村 啓二[1]　笠原 博徳[1]

概要：より安全，快適，省エネな自動車の要求が高まっており，自動車制御系の計算負荷が増大している．これに伴い，制御用プロセッサコアに高い性能が求められるが，動作周波数の向上によるプロセッサコアの高性能化が困難となっており, マルチコアへの移行が求められている．しかし，エンジン制御におけるマルチコア利用においては手動によるプログラムの並列化が困難で，処理性能，並列化に伴うコスト, 期間等が問題となっている．本稿では，これらの問題を解決し，従来シングルコアのみで動作していた自動車エンジン制御ソフトウェアをマルチコア上で並列化する手法を提案する．具体的には，自動車エンジン制御 C プログラムに対し，より多くの並列性を抽出するため，関数のインライン展開および条件分岐の複製等，逐次プログラムのリストラクチャリングを行った後，OSCAR 自動並列化コンパイラにより自動並列化を行う．その結果，従来タスクの粒度が細かく，手動での並列化が困難であった自動車エンジン制御ソフトウェアを，組込用マルチコア RP-X 上で 2 コアを用いて並列実行したところ，1 コアに対して 1.71 倍の速度向上を得ることに成功し，自動車エンジン制御ソフトウェアのマルチコア上での並列処理が有効であることを確認した．

Parallelization of Automobile Engine Control Software on Multicore Processor

Youhei Kanehagi[1]　Dan Umeda[1]　Hiroki Mikami[1]　Akihiro Hayashi[1]　Mitsuo Sawada[2]
Keiji Kimura[1]　Hironori Kasahara[1]

Abstract: The calculation load in the automobile control system is increasing to achive more safety, comfort and energy-saving. Accordingly, control processor cores needs high performance. However, the improvement of clock frequency in processor cores is difficult, and it is important to use multicore processor. Using the multicore for the engine control, performance, development cost, development period, etc are problems because it is difficult to parallelize softwares. This paper proposes a parallelization method of the automobile engine control software on the multicore processor, which has only functioned on single-core processors. Concretely, it is applied restructuring the sequential program for extracting more parallelism, for example inlining functions and duplicating conditional branches, and the OSCAR compiler allows us perform automatic parallelization and generation of a parallel C program. Using proposed method, the automobile engine control software, which is difficult to parallelize manually because of very fine-grained program, is parallelized and give us 1.71x speedup using 2 cores on RP-X multicore. It is confirmed that parallelization of the automobile engine control software is effective.

1. はじめに

より安全，快適，省エネを実現する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，音楽・映像等の提示を行う情報系，制御系と情報系を統合し，制御を行う統合制御系，それぞれの高度化が重要となっている．

これらの制御系，情報系，および統合制御系の高度化の

[1] 早稲田大学
Waseda University
[2] トヨタ自動車株式会社
Toyota Motor Corporation

ⓒ 2013 Information Processing Society of Japan

ためには，プロセッサの高能力化が重要となる．例えば，安全，快適，省エネな自動車開発のために重要な，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，新制御機能の実現など計算負荷の増大を避けられない．この問題を解決するためには，エンジン制御を実現しているプロセッサの高能力化が必須となる．

しかし，従来のようにプロセッサの高能力化のためにプロセッサの動作周波数を向上させることは，消費電力が周波数の三乗に比例して増大してしまうため，自動車に適用することは困難である．このため 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，電力削減のために低周波数化・低電圧化したプロセッサコアを並列動作させることにより，処理の高速化と低電力化を同時に実現可能なマルチコアプロセッサへの移行が求められている．

このような要求に対し，自動車業界におけるマルチコアを利用した技術が提案されている．例えば，UCC アルゴリズムを利用したマルチコアアーキテクチャ [1][2] では，電子制御ユニットにおけるマルチコアが提案されている．これは 3 プロセッサコアを用い，電子制御ユニットを 3 個の機能に分割し，機能分散を実現している．機能分散を実現することでスループットを向上することが可能であるが，レイテンシの削減が困難である．また，機能毎の負荷バランスが均等でないとマルチコア資源を最大限に活用することができないという問題がある．

本論文で提案する手法では，自動車制御系の主要機能であるエンジン制御に対し，並列処理を適用し，レイテンシの削減すなわち高速化を行う．このようなマルチコア上で，エンジン制御等の計算を従来の 1 プロセッサコアより高速に行うためには，計算を分割し，計算負荷を複数のプロセッサにうまく割り当てて実行することが重要となる．しかし，このプログラムの並列化は，従来は人手で行うことが要求されていたが，並列処理の知識のないプログラム開発者がプログラムの並列化を行うことは大変困難で，長期間を要し，開発期間の増大それに伴うソフトウェア開発費の増大を招いてしまうという問題や，並列処理の知識のないプログラマが作成する並列プログラムの信頼性などの問題等，ソフトウェア的に解決しなければならない課題が多く生じている．

このような状況を踏まえ，本稿では，ループ処理がなく，条件分岐や代入文などの基本ブロックが連続している従来の並列化方式では並列化が困難なエンジン制御ソフトウェアに対するマルチコアプロセッサ上での並列化手法を提案する．具体的には，トヨタ自動車 (株) のエンジン制御用 C プログラムを，早稲田大学が開発した OSCAR 自動並列化コンパイラを用いて，マルチコア用の並列プログラムに自動的に変換するための方式を開発すると共にその性能評価を行った．

C プログラムの並列化においては，世界での長期間の研究にも関わらず，現存のコンパイラ全てが，ポインタの解析および再帰部分の並列化に限界があり，OSCAR コンパイラにおいても効果的に自動並列化機能を利用するためには，入力とする逐次 C プログラムを Parallelizable C[3] に修正する必要がある．このため，エンジン制御用 C プログラムを，OSCAR コンパイラが解析可能な Parallelizable C への書き換えを行った．具体的には，同 C プログラムのタスクに内在する並列性を最大に引き出すための逐次プログラムのリストラクチャリング (インライン展開，条件分岐の複製) を行い，まず第一段階として，早稲田大学が保有する SH4A を集積したマルチコア RP-X 上で，まず自動車エンジン制御で目標としている 2 プロセッサコアを用いて，並列処理性能評価を行った．その結果，2 プロセッサコアの利用により，1 プロセッサコアより最大で 1.71 倍の速度向上が得られ，自動車エンジン制御ソフトウェア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．

2 章では，自動車エンジン制御ソフトウェアの概要，3 章では，OSCAR コンパイラの概要，4 章では，自動車エンジン制御ソフトウェアの並列化手法，5 章では，性能評価，6 章では，まとめを述べる．

2. 自動車エンジン制御ソフトウェアの概要

本章では，本稿での自動車エンジン制御ソフトウェアの概要を示す．

本稿での自動車エンジン制御ソフトウェアは，トヨタ自動車 (株) により開発されたエンジン制御ソフトウェアである．本ソフトウェアは，C 言語で記述されており，自動車向けリアルタイム OS である OSEK/VDX 上 [4] で動作する．

プログラムの動作を以下に示す．

Step 1. OS の起動
C 言語の main 関数から OSEK/VDX が提供する API である StartOS 関数をコールすることで OS を起動する．

Step 2. エントリタスクの起動
エントリタスクから周期的に実行されるタスクの実行予約を行う．本エンジン制御ソフトウェアでは，タスクは Task2-Task39 の合計 38 個定義されており，周期タスクである Task22-Task39 の実行予約が行われる．なお，Task2-Task21 は周期タスクから間接的に呼び出されるタスクである．

Step 3. 周期タスクの起動
OS 内のタイマにより，周期タスクが周期的に実行される．

図 1 に実行プロファイル結果を示す．なお，本結果は実際にエンジン制御ソフトウェアを動作させている車載向けマイコンを用いて，測定した結果である．図 1 において，横軸はタスク番号，縦軸は各タスクの実行時間がプログ

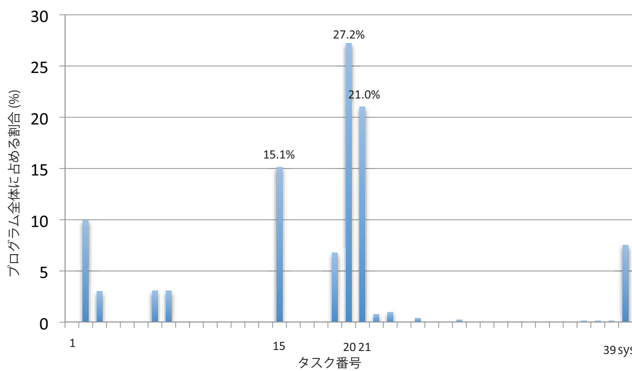

図 1　エンジン制御プログラムの実行プロファイル結果

Fig. 1　Execution Profile Result of Engine Control Program

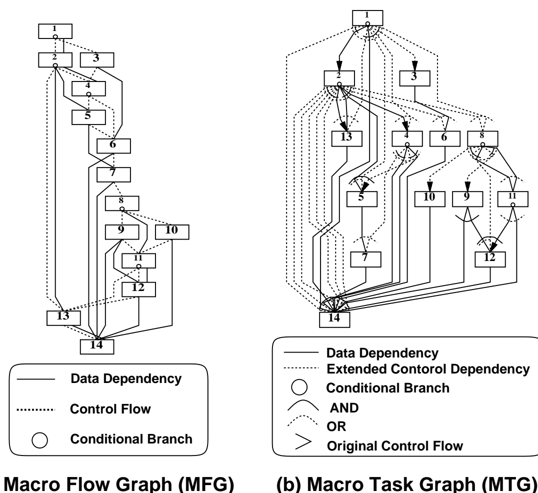

図 2　マクロフローグラフ (MFG)，マクロタスクグラフ (MTG) の例

Fig. 2　Example of Macro-Flow Graph and Macro-Task Graph

ラム全体実行時間に占める割合を示している．図 1 より，Task20，Task21，Task15 の順番で処理時間が大きいことがわかる．このうち，Task20，Task21 はアイドルタスクのため，本研究では実質的に処理量の一番大きい Task15 に着目し，並列化を行う．

3. OSCAR コンパイラの概要

本章では，OSCAR コンパイラ [5] 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 [6][7] 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プログラム全域にわたる並列処理を行なう手法である．

3.1 粗粒度タスク並列処理 (マクロデータフロー処理)

粗粒度並列処理では，ソースとなるプログラムを疑似代入文ブロック (BPA)，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)) に分割する．MT 生成後，コンパイラは BPA，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 [8][9]．その後コンパイラは 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 2 に示す．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．また，ノード内の小円は条件分岐を表す．MTG におけるノードも MFG 同様 MT を表し，ノード内の小円は MT 内の条件分岐を表す．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だけではなく，データ依存と制御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．MTG においてはエッジの矢印は省略されているが，下向きが想定されている．また，矢印を持つエッジはオリジナルのコントロールフローを表す．

3.2 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，ダイナミックスケジューリングと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．

3.2.1 ダイナミックスケジューリング

条件分岐などの実行時不確定性が存在する場合にはダイナミックスケジューリングによって実行時にマクロタスクを PG に割り当てる．ダイナミックスケジューリングルーチンはマクロタスクの終了や分岐方向の決定に応じて，マクロタスク実行管理テーブルを操作し，各マクロタスクの最早実行可能条件を検査する．マクロタスクが実行可能であればレディキューにマクロタスクが投入される．レディキュー内のマクロタスクはその優先順位に従ってソートされ，レディキューの先頭のマクロタスクがアイドル状態のプロセッサクラスタに割り当てられる．また，ダイナミックスケジューリングコード生成時には，一つの専用のプロセッサがスケジューリングを行う集中スケジューリング方式と，スケジューリング機能を各プロセッサに分散した分

ⓒ 2013 Information Processing Society of Japan

Vol.2013-ARC-203 No.2
2013/1/31

散スケジューリング方式を，使用するプロセッサ台数，システムの同期オーバーヘッドを考慮して使い分けることができる．

3.2.2　スタティックスケジューリング

一方，スタティックスケジューリングは，マクロタスクグラフがデータ依存エッジのみを持つ場合に使用され，自動並列化コンパイラがコンパイル時にマクロタスクの PG への割り当てを決める方式である．スタティックスケジューリングでは，実行時スケジューリングオーバーヘッドを無くし，データ転送と同期のオーバーヘッドを最小化することが可能である．

3.3　並列化プログラムの生成

並列化プログラムの生成は OSCAR API[10] を用いた並列化 C あるいは Fortran 77 というように，source-to-source で行うことも可能である．この場合には様々なプラットフォームにおいて実行可能な形にするため，OSCAR API 標準解釈系を用いて，API 部分をランタイムライブラリコールに変換した後，各プロセッサ用のコードを逐次コンパイラでコンパイルし，バイナリを生成する．

4.　自動車エンジン制御ソフトウェアの並列化手法

本章では，自動車エンジン制御ソフトウェアの並列化手法について述べる．

4.1　Task15 の並列性解析

Task15 にはエントリ関数があり，これを Task15main と呼ぶこととする．当該ソフトウェアは，Task15 計算部本体とテストドライバからなり，テストドライバは Scenario1，Scenario2，Scenario3 からなる 3 つの実行シナリオで Task15 を実行する．

図 3 に Task15main のマクロタスクグラフを示す．また，各タスクの実行シナリオ別のタスクコストを図 4 に示す．タスクコストの計測は，RP-X 上 648MHz にて行い，単位は clock である．図 3 より，この階層での並列度は，sb1，bb3，sb4 をそれぞれ実行可能であるため，3 であるが，図 4 より，sb4 の占める割合が，Scenario1 で 94.3%，Scenario2 で 92.6%と相対的に大きいため，これらのシナリオでは sb4 の内部を並列化することが重要である．Scenario3 に関しては，sb4 が 29%，sb7 が 68.9%となるため，sb4，sb7 の内部をそれぞれ並列化することが重要である．

また，ソフトウェアのプログラム構造に関しては，大きなループ処理がなく，条件分岐と代入文の連続であり，従来のループ並列化などの並列処理は適用不可能である．さらに，図 4 より，実行時間が非常に短いため，並列化した際の同期処理オーバーヘッドや，ダイナミックスケジューリングのスケジューリングオーバーヘッドが相対的に大きくなってしまうという問題点がある．

そこで，当該ソフトウェアに並列化を施すには，代入文や関数間の並列性を利用する粗粒度並列化と，スタティックスケジューリングを適用することが重要となる．

4.2　並列性向上のためのリストラクチャリング手法

本節では，並列性向上のために行った逐次プログラムへのリストラクチャリングについて述べる．

4.2.1　インライン展開

まず，図 3 における sb4 や sb7 が内包する並列性を有効活用するため，sb4 や sb7 の内部関数のマクロタスクグラフと RP-X のプロファイル情報を基に，並列性があり，かつ相対的にタスクコストの大きい関数を選択し，その関数を Task15main の階層までインライン展開していく．これにより，Task15main の階層における並列性が向上する．

4.2.2　条件分岐の複製

次に，sb7 に関しては，条件分岐内に存在するため，sb7 にインライン展開を施してもそれらは 1 つの条件分岐内に収まってしまい，1 つのタスクとしてプロセッサに割り振

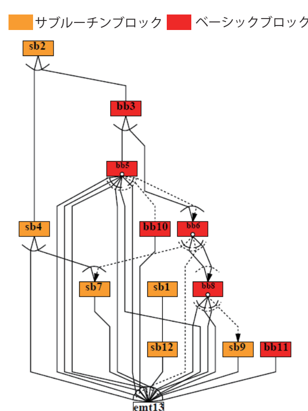

図 3　Task15main の MTG
Fig. 3　MTG of Main Function in Task15

Scenario1				
sb1	sb2	sb4	sb12	全体
978	3415	114739	519	121680

Scenario2					
sb1	sb2	sb4	sb9	sb12	全体
42	489	20031	393	38	21641

Scenario3					
sb1	sb2	sb4	sb7	sb12	全体
40	228	37225	88476	455	128364

図 4　Task15 内の各タスクのクロック数
Fig. 4　Clock Cycles of Tasks in Task15

ⓒ 2013 Information Processing Society of Japan

られてしまう．そこで，条件分岐内の並列性を抽出するために，手動で条件分岐の複製を行う．例えば，図 5 のような条件分岐があり，条件分岐内の 3 個の関数が並列化可能だとし，条件式 (condition) が条件分岐内で変更されないとする．図 5 の状態では，この条件分岐を 1 つのプロセッサに割り当てることになり，func1-func3 の並列性を活かすことができない．そこで，図 6 のようにプログラムを書き換える．この処理を行うことにより，各条件分岐が 1 つのタスク，合計 3 つのタスクに複製され，別々のプロセッサに割り当てることが可能となる．これにより，条件分岐内の並列性を抽出することが可能となり，Task15main の階層における並列性が向上する．

```
If (condition) {
   func1();
   func2();
   func3();
}
```

図 5 条件分岐内の並列性

Fig. 5 Parallelism in Conditional Branch

```
If (condition) {
   func1();
}
If (condition) {
   func2();
}
If (condition) {
   func3();
}
```

図 6 条件分岐の複製

Fig. 6 Duplication of Conditional Branch

インライン展開や条件分岐の複製を行い，並列性の向上を図った後の Task15main の MFG を図 7 に示す．

4.2.3 タスク融合

最後に，条件分岐に対し，実行時にタスクをプロセッサに割り当てるダイナミックスケジューリングを適用すると，数百から数千クロック程度のタスクサイズに対して，数十から数百クロックを要するダイナミックスケジューリングのオーバーヘッドが生じてしまうため，適用が困難である．しかし，条件分岐に対し，そのままでは OSCAR コンパイラでコンパイル時にタスクをプロセッサに割り当てるスタティックスケジューリングを適用することができない．そこで，本稿では条件分岐とその分岐先を 1 つの粗粒度タスクとして融合するタスク融合を OSCAR コンパイラにて行った．これにより，コントロールフローを全てデータ依存の形に集約することができ，低オーバーヘッドなスタティックスケジューリングが適用可能となる．また本稿では行っていないが，OSCAR コンパイラでコンパイル時にタスクをプロセッサに割り当てるスタティックスケジューリングを適用することにより，各プロセッサでよく使用される変数を OSCAR コンパイラを用いて解析することができ，その情報を用いてメモリ管理を行うことでより高速化を図ることが可能となる．

4.3 並列性向上のためのリストラクチャリング手法適用後の Task15 の並列性解析

図 7 に対し，タスク融合を行った後の MTG を図 8 に示す．タスク融合を施したブロックは loop と表示されるものである．図 8 からわかるように，条件分岐や代入文を並列性を損なうことのない範囲でタスク融合を行うことにより，1 つ 1 つのマクロタスクが数百から数千クロック程度の処理コストを持った粒度とすることができた．また，並列性に関し，データ依存のみの 2 並列程度の並列性抽出に成功した．これにより，低オーバーヘッドなスタティックスケジューリング粗粒度並列化が可能となる．

5. 性能評価

本章では，ルネサスエレクトロニクス/日立/早稲田大学で開発した組込用マルチコア RP-X 上での自動車エンジン制御ソフトウェアの並列処理性能の評価を行う．本研究ではまず第一段階として，並列性能の評価に組込用マルチコア RP-X を用いる．

5.1 組込用マルチコア RP-X

RP-X[11] は日立製作所，ルネサステクノロジ，東京工業大学，早稲田大学にて共同開発された 45nm Low Power テクノロジ，15 コアのヘテロジニアスマルチコアで，汎用コアとして動作周波数を 648MHz，324MHz，162MHz，81MHz と変更して動作する SH-4A コアを 8 基，アクセラレータコアとして 324MHz で動作する FE-GA[12] を 4 基，その他ハードウェア IP を搭載している (図 9)．

各汎用プロセッサ内メモリは命令キャッシュ (32KB)，データキャッシュ (32KB)，ローカルメモリ (ILM, DLM:16KB)，分散共有メモリ (URAM:64KB)，データ転送ユニットを持つ．また，アクセラレータコアはコントローラなしアクセラレータであり，オンチップバス (SHwy#1) に接続されている．

本稿では，現在エンジン制御系で 2 コアのマルチコアが検討されているため，汎用コアとして 2 基の SH-4A コアを計算資源として用いた．また，汎用コアの動作周波数を 648MHz から 324MHz，162MHz，81MHz と変化させた時のバスの動作周波数は 324MHz に固定し，性能評価を行った．これはバスの動作周波数を固定し，汎用コアの動作周波数を下げることで，メモリアクセスレイテンシを相対的に低くするためであり，メモリアクセスレイテンシの小さい車載向けマイコンの環境に近付けるためである．

5.2 組込用マルチコア RP-X 上での並列処理性能評価条件

図 10 に RP-X 上 648MHz 実行時の Task15 の実行プロファイル結果を示す．なお，単位は clock である．RP-X 上で性能評価を行うにあたっては，各 Scenario 毎に OSCAR

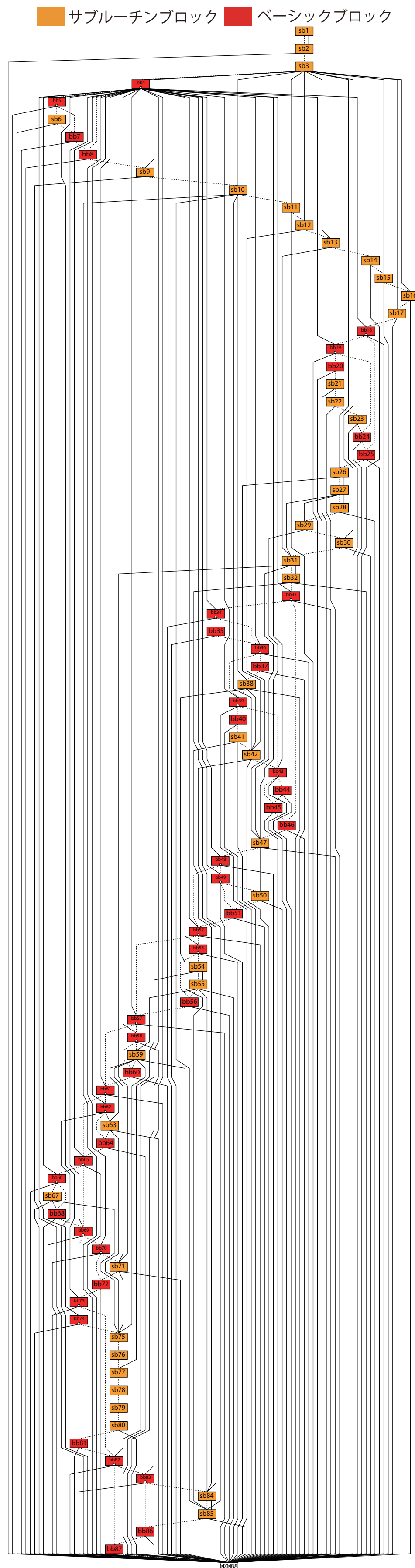

図 7　Task15 のインライン展開・条件分岐の複製後の MFG
Fig. 7　MFG of Task15 After Function Inlining and Duplication Conditional Branch

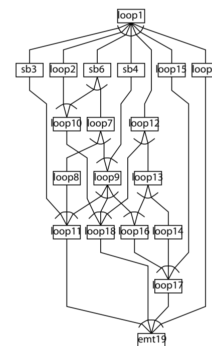

図 8　Task15 のインライン展開・条件分岐の複製・タスク融合後の MTG
Fig. 8　MFG of Task15 After Function Inlining and Duplication Conditional Branch and Task Fusion

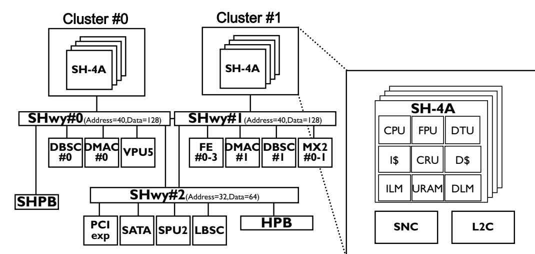

図 9　組込用ヘテロジニアスマルチコア RP-X
Fig. 9　Heterogeneous Multicore RP-X

コンパイラに本プロファイル情報を与え，スタティックスケジューリングを行うことで，負荷分散を図る．これは図 10 より，各タスクコストが各 Scenario で全く異なることがわかるが，各 Scenario において高速化するためには，負荷分散が重要なためである．

また通常，グローバル変数はオフチップ共有メモリに配置する．この場合，キャッシュヒットの際には 1 サイクルかかるが，キャッシュミスの際には 55 サイクルかかってしまう．エンジン制御プログラムのような us オーダーのプログラムにおいて，このペナルティは非常に大きい．そこで，メモリアクセスレイテンシの小さいローカルメモリに配置することが重要となる．しかし，全てのグローバル変数をローカルメモリに配置してしまうと，メモリ容量を超えてしまうため，メモリ容量内に収めるために，初期値無しのグローバル変数 (約 7.5kbyte) をローカルメモリに配置する．また，プロセッサコア間の同期を担う同期変数に関しても，オフチップ共有メモリに配置してしまうと，ペナルティが大きいため，メモリアクセスレイテンシの小さい分散共有メモリに配置する．これにより，高速化が可能となる．

メモリ配置による性能差を比較するため，グローバル変数全てを共有メモリに配置した場合と，グローバル変数の一部をローカルメモリに，プロセッサコア間の同期を担う同期変数を分散共有メモリに配置した場合の性能評価を行う．

ⓒ 2013 Information Processing Society of Japan

	Scenario1	Scenario2	Scenario3		Scenario1	Scenario2	Scenario3
loop1	6038	987	586	loop10	22476	1485	168
loop2	9665	2518	1558	loop11	38268	4672	18534
sb3	4204	197	1090	loop12	285	357	2826
sb4	1720	727	403	loop13	370	129	2459
loop5	7371	7961	4583	loop14	403	127	1685
sb6	5606	3096	11696	loop15	116	129	3739
loop7	12957	2518	2291	loop16	351	129	19811
loop8	4234	323	1393	loop17	426	129	39603
loop9	10326	3742	4641	loop18	380	324	15219

図 10 RP-X 上での実行プロファイル結果
Fig. 10 Execution Profile Result on RP-X

表 1 Scenario1 の実行時間
Table 1 Execution Time of Scenario1

	81MHz	162MHz	324MHz	648MHz
1CPU	356.2us	253.9us	198.5us	174.7us
2CPU	227.0us	163.8us	129.8us	118.3us
2CPU-memory opt	222.4us	148.1us	117.1us	107.7us

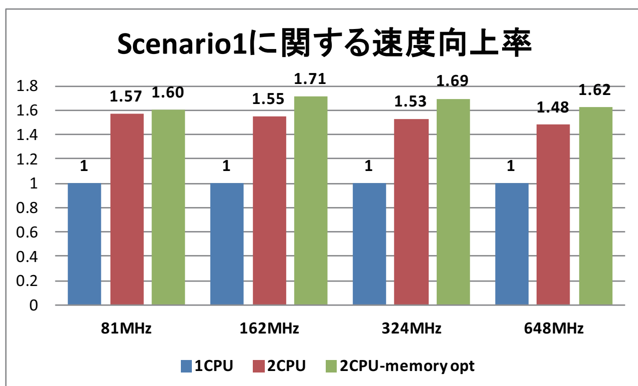

図 11 RP-X 上での Scenario1 の速度向上率
Fig. 11 Speedup of Scenario1 on RP-X

5.3 組込用マルチコア RP-X 上での並列処理性能評価

図 11 に Scenario1 を 1CPU，2CPU，メモリ配置を工夫した 2CPU で実行した場合の速度向上率を示す．図 11 において，横軸は動作周波数，縦軸は 1CPU 実行時に対する速度向上率である．具体的な実行時間は表 1 に示すとおりである．81MHz 構成で 1.57 倍，162MHz で 1.55 倍，324MHz で 1.53 倍，648MHz で 1.48 倍の速度向上が得られた．メモリ配置を工夫した場合，81MHz 構成で 1.60 倍，162MHz で 1.71 倍，324MHz で 1.69 倍，648MHz で 1.62 倍の速度向上が得られた．メモリ配置の工夫を行うことで，2%から 11%の性能改善が得られた．

図 12 に Scenario2 を 1CPU，2CPU，メモリ配置を工夫した 2CPU で実行した場合の速度向上率を示す．図 12 において，横軸は動作周波数，縦軸は 1CPU 実行時に対する速度向上率である．具体的な実行時間は表 2 に示すとおりである．81MHz 構成で 1.38 倍，162MHz で 1.46 倍，324MHz で 1.40 倍，648MHz で 1.17 倍の速度向上が得られた．メモリ配置を工夫した場合，81MHz 構成で 1.54 倍，162MHz で 1.58 倍，324MHz で 1.45 倍，648MHz で 1.25 倍の速度向上が得られた．メモリ配置の工夫を行うことで，3%から 11%の性能改善が得られた．

表 2 Scenario2 の実行時間
Table 2 Execution Time of Scenario2

	81MHz	162MHz	324MHz	648MHz
1CPU	122.0us	74.4us	49.9us	37.2us
2CPU	88.6us	51.0us	35.6us	31.7us
2CPU-memory opt	79.5us	47.1us	34.5us	29.9us

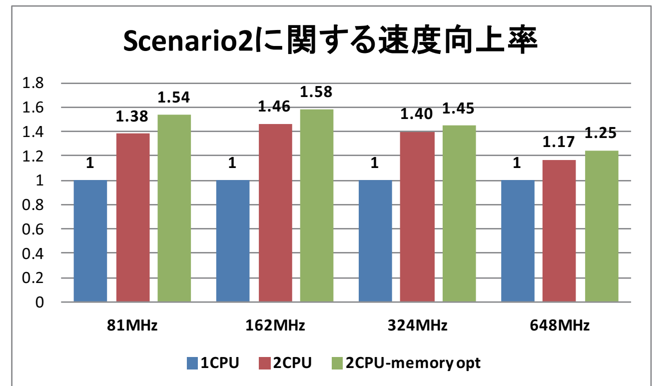

図 12 RP-X 上での Scenario2 の速度向上率
Fig. 12 Speedup of Scenario2 on RP-X

表 3 Scenario3 の実行時間
Table 3 Execution Time of Scenario3

	81MHz	162MHz	324MHz	648MHz
1CPU	546.3us	356.2us	255.5us	207.7us
2CPU	360.8us	243.2us	181.1us	146.7us
2CPU-memory opt	356.5us	237.7us	175.1us	145.4us

図 13 に Scenario3 を 1CPU，2CPU，メモリ配置を工夫した 2CPU で実行した場合の速度向上率を示す．図 13 において，横軸は動作周波数，縦軸は 1CPU 実行時に対する速度向上率である．具体的な実行時間は表 3 に示す通りである．81MHz 構成で 1.51 倍，162MHz で 1.46 倍，324MHz で 1.41 倍，648MHz で 1.42 倍の速度向上が得られた．メモリ配置を工夫した場合，81MHz 構成で 1.53 倍，162MHz で 1.50 倍，324MHz で 1.46 倍，648MHz で 1.43 倍の速度向上が得られた．メモリ配置の工夫を行うことで，1%から 3%の性能改善が得られた．

6. まとめ

本稿では，自動車エンジン制御ソフトウェアのマルチコア上での並列化手法を提案した．今回は研究の第一段階として，並列性を抽出するための手動による条件分岐の複製

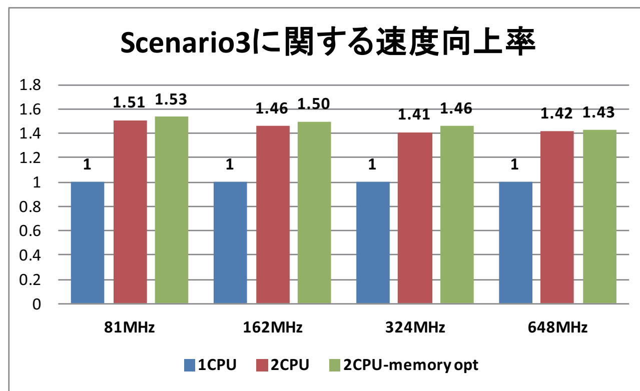

図 13 RP-X 上での Scenario3 の速度向上率
Fig. 13 Speedup of Scenario3 on RP-X

および相対的にコストの大きい関数のインライン展開，スタティックスケジューリング適用のための OSCAR コンパイラによるタスク融合を用いて，逐次プログラムのリストラクチャリングを行った後，OSCAR コンパイラを用いて自動並列化を行い，組込用マルチコアプロセッサ RP-X 上で並列処理性能の評価を行った．その結果，マルチコア上で従来並列化に成功した例がない極めて並列化が困難であった自動車エンジン制御ソフトウェアにおいて，2 プロセッサコアを用いた場合，1 プロセッサコアを使用した場合と比較して，Scenario2 における 162MHz の場合，1.71 倍の性能向上が得られた．

本稿で OSCAR コンパイラによる並列化を利用し，高速化を実現できたため，自動車エンジン制御ソフトウェアの自動並列化及び高速化が可能であることが確認できた．自動車エンジン制御ソフトウェアのような，条件分岐や代入文の多いプログラムにおいても，並列処理による高速化が可能であることを示すことができた．自動車エンジン制御ソフトウェアのマルチコアによる高速化により，自動車の燃費の向上や新制御機能等による，エンジン制御の更なる高度化の可能性を示すことができた．

今後の展望としては，手動で行った条件分岐の複製を OSCAR コンパイラへ実装し，全て自動化し，エンジン制御計算の並列化を可能にすると共に，メモリ配置の最適化を行うことが挙げられる．

参考文献

[1] Seo, K., Yoon, J., Kim, J., Chung, T., Yi, K. and Chang, N.: Coordinated implementation and processing of a unified chassis control algorithm with multi-central processing unit, *JAUTO1346*, Vol. 224 (2009).

[2] Seo, K., Chung, T., Heo, H., Yi, K. and Chang, N.: 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, pp. 53–62 (2010).

[3] 間瀬, 木村, 笠原: マルチコアにおける Parallelizable C プログラムの自動並列化, 情報処理学会研究報告,Vol.2009-ARC-184, No.15,pp.1-10 (2009).

[4] OSEK/VDX-Portal: OSEK VDX Portal, http://portal.osek-vdx.org/.

[5] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc of The 13th International Workshop on Languages and Compilers for Parallel Computing(LCPC2000)* (2000).

[6] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. of 13th International Workshop on Languages and Compilers for Parallel Computing (LCPC'00)* (2000).

[7] 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).

[8] 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).

[9] 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).

[10] 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 Workshop on Languages and Compilers for Parallel Computing(LCPC2009)* (2009).

[11] Yuyama, Y., Ito, M., Kiyoshige, Y., Nitta, Y., Matsui, S., Nishii, O., Hasegawa, A., Ishikawa, M., Yamada, T., Miyakoshi, J., Terada, K., Nojiri, T., Satoh, M., Mizuno, H., Uchiyama, K., Wada, Y., Kimura, K., Kasahara, H. and Maejima, H.: A 45nm 37.3GOPS/W heterogeneous multi-core SoC, *IEEE International Solid-State Circuits Conference, ISSCC* (2010).

[12] 津野田, 高田, 秋田, 田中, 佐藤, 伊藤: ディジタルメディア向け再構成型プロセッサ FE-GA の概要, 信学技報 RECONF2005-65 (2005).

ⓒ 2013 Information Processing Society of Japan