Vol.2012-ARC-201 No.22
2012/8/2

エンジン基本制御ソフトウェアモデルのマルチコア上での並列処理

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

概要：自動車の安全性・快適性・環境負荷の低減を目指し，自動車制御系は年々高度化している． これに伴い， 制御プロセッサには高い性能が求められるが， シングルコアの動作周波数， 及び命令レベル並列性の向上が困難となり， 1 コアによる処理性能が限界に達したため， マルチコアへの移行が求められている． しかし， マルチコアではプログラムの並列化の困難なため， 並列化プログラムの開発コスト・開発期間・信頼性等が問題となっている． 本稿では従来シングルコアのみで動作していた基本エンジン制御ソフトウェアモデルのマルチコア上での並列化手法を提案する． 具体的には基本エンジン制御 C プログラムをポインタ利用等に制限を加えた Parallelizable C によって記述されたプログラムに変換し， 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， 従来タスク粒度が細かく手動では並列化ができなかった基本エンジン制御 C プログラムを情報家電用 RP2 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 1.89 倍， V850 2 コア上で 1 コアに対して 2.06 倍の性能向上することに成功し， エンジン制御ソフトウェアモデルのマルチコア上での並列処理が可能であることを確認した．

Parallelization of Basic Engine Controll Software Model on Multicore Processor

Dan Umeda[1]　Youhei Kanehagi[1]　Hiroki Mikami[1]　Akihiro Hayashi[1]　Mituhiro Tani[2]
Yuji Mori[2]　Keiji Kimura[1]　Hironori Kasahara[1]

Abstract: The automobile control system is advancing from year to year to achieve safety, comfort and fuel efficiency. Accordingly, control system needs high performance. However, the improvement of clock frequency and instruction-level parallelism are difficult, and the performance of a single-core processor has reached the limits. This paper proposes a parallelization method of a basic engine control software model for a multicore processor, which has only functioned on single-core processors. In the multicore, development cost, development period, and software reliability are problems because it is difficult to parallelize a software. By developing a Parallelizable C program with some limitations for pointer usage, the OSCAR compiler allows us perform automatic parallelization and generation of a parallel C program. Using the proposed method, the basic engine control program, which is difficult to parallelize because of very fine grain, is parallelized and gives us 1.89 times speedup using 2 cores on RP2 multicore and 2.06 times speedup using 2 cores of V850 multicore. It is confirmed that parallelization of a basic engine control C program on multi-core processor is possible.

1. はじめに

より安全， 快適， かつ省エネな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， 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，情報系，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のために重要なエンジン制御系を高度化するために

[1] 早稲田大学
WASEDA UNIVERSITY
[2] 株式会社デンソー
DENSO CORPORATION

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． このためには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プロセッサの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，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，速度向上と信頼性のために低周波数化・低電圧化した（すなわち低速化した）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の検討が行われている．

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 コアより高速に行うためには，計算を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，このソフトウェアの並列化は複雑で， 通常のプログラム開発者が行うことは困難で長期間を要してしまうため，開発期間， 及び開発コストの増大を招いてしまうという問題が生じる． また， 未熟なプログラマが作成する並列プログラムの信頼性の問題等， ソフトウェア的に解決しなければならない課題が多く指摘されている．

このような状況を踏まえ， 本稿では， マルチコアプロセッサを用いた基本エンジン制御 C プログラムの自動並列化について報告する． 具体的には， 株式会社デンソー より開発されたエンジン制御用 Simulink[1] ブロックモデルを Real-Time Workshop[2] により変換した C プログラムをご提供戴き，同 C プログラムを早稲田大学が開発した OSCAR 並列化コンパイラで, エンジン制御に使われている V850 プロセッサベースの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の開発及びその性能評価を行った．

提案する OSCAR 自動並列化コンパイラによる自動並列化ではサブシステム間のみの並列化にとどまらず，サブシステム内の細かい並列解析を行うことが可能である． さらには負荷バランスが均衡するように自動的にスケジューリングを行うため、より高い性能を自動で引き出すことが可能である．

本稿では基本エンジン制御 C プログラムを，並列性の抽出しやすく， 性能向上が最も得られやすい Parallelizable C[3][4] への手動書き換えを行い，OSCAR コンパイラがそれを入力として生成する並列プログラムの自動並列化，及び 8 コアの SH4A を集積したホモジニアスマルチコア RP2， 及び V850 を 2 コア集積したマルチコア上での評価を行った．

2. 基本エンジン制御ソフトウェアモデルの概要

本稿での基本エンジン制御ソフトウェアモデルは株式会社デンソーにより Simulink 上でモデルベース開発されたエンジン制御ブロックモデルである．本モデルは基本噴射量計算や燃料補正計算を行うサブシステムより構成され，スロット開度やクランクの回転角，吸気温の排気中の酵素温度など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． このモデルを実際の組み込みシステム上動作させるため， Real-Time Workshop により C プログラムを自動生成する． Real-Time Workshop で出力される C プログラムはすべて逐次プログラムであるため，別途マルチコアに適した並列化プログラムに変換する必要がある．

2.1 本エンジン制御 C プログラムの基本的な構成

Real-Time Workshop が出力した C プログラムは以下のように初期化関数， 繰り返し実行される主要エンジン制御計算関数から構成される．

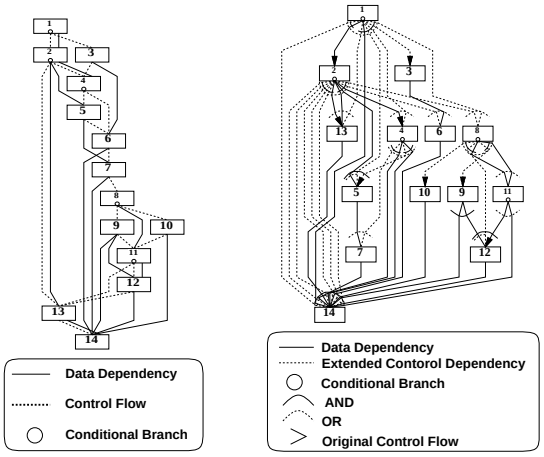
```
int main(void) {
      int tid;
      MdlInitializeSizes();  /*初期化関数*/
      MdlStart();
      for(回転数) {
      MdlOutputs(tid); /*ブロック線図の情報がすべて記載される関数*/
     }
}
```

実際のエンジン制御にあたる演算は MdlOutputs 関数である． この関数を繰り返し実行することで，シミュレーションを行う． 本プログラムでの MdlOutputs 関数では 1 回転あたり高々マイクロオーダーであり， プログラムの粒度は細かい． この関数は対象の Simulink ブロック線図モデルのモデル内のすべてのブロックに該当する処理を， 計算するものである． 本稿ではエンジン制御に計算部分に直結している MdlOutputs 関数に着目して並列化を行う．

また， 本プログラムの並列化に際して， 注意すべき点は， このプログラムは従来の並列化コンパイラあるいは並列化手法が対象としてきた並列化可能なループが存在しないということである．すなわち， マルチコア上での並列化においてループ並列化が適用できないプログラムである．しかしながら， 本来の Simulink ブロック線図のレベルでは並列性が認められ，この並列性を後述する OSCAR コンパイラによる粗粒度レベルタスク並列化により抽出する．

3. OSCAR コンパイラの概要

OSCAR コンパイラではループイタレーションレベルで



(a) Macro Flow Graph (MFG) (b) Macro Task Graph (MTG)

図 1 マクロフローグラフ (MFG)，マクロタスクグラフ (MTG) の例

Fig. 1 Example of Macro-Flow Graph and Macro-Task Graph

の並列処理 (中粒度並列処理) を行うのみでなく，ループ・手続き間の粗粒度タスク並列処理，ステートメント間の近細粒度並列処理を組み合わせたマルチグレイン並列処理 [5] が実現されている．

OSCAR コンパイラ [6] は C 言語のポインタ制限や関数ポインタや再帰処理に制限を加えた Parallelizable C や Fortran 77 言語で記述された逐次ソースプログラムを入力とし，(1) 粗粒度タスクの生成及びコストの推定 (2) プロセッサへの粗粒度タスクのスタティックスケジューリング (3) タスク間同期コードを含めた並列化コードの生成を行う．

OSCAR コンパイラでは基本ブロック (BB) やループ (RB) やサブルーチンブロック (SB) 等へのブロックに分割を行う．その上で粗粒度タスクすなわちマクロタスク (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) を生成する．その際，コンパイラはソース中のタスクの入力変数の定義，出力変数の使用を解析することでデータ依存解析を行う．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出した結果をマクロタスクグラフ (MTG) として表現する [7][8]．MFG および MTG の例を図 1 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 内の条件分岐を表している．また，実線のエッジはデータ依存を表し，点線の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． 並列化プログラムの生成は OSCAR API を用いた並列化 C あるいは Fotran 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毎に実行可能な形にするため， OSCAR API 標準解釈系 [9] 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，各ロセッサ用コードを逐次コンパイラでバイナリを生成する．

4. エンジン制御ソフトウェアモデルの並列化手法

OSCAR コンパイラにより解析された MdlOutputs 関数に関する MFG を図 2 に示す． MdlOutputs 関数は並列性を細かく引き出す必要性があるため， インライン展開を施してある．また， インライン展開後の本プログラムに MFG は大変細かいため， 図 2 の A に該当する MFG の拡大図を図 3 に示す．

この MFG において bb は基本ブロックを示し， この図からわかるように，本プログラムには処理コストの大きな並列化可能なループが存在せず，ノード内の小円により表される条件分岐を持つ基本ブロックが多数が存在する． また， 条件分岐の分岐先の演算もおおよそが高々数十クロックサイクルである． そのため， これらの条件分岐や代入文を関数呼び出しや回転数の小さなループに融合することにより， できる限り粒度の粗い， 粗粒度タスクを並列性の損なわれない範囲で生成し，極力同期処理を挟まないように粗粒度タスク並列処理を行えるようにプログラムのリストラクチャリングを行う． また， 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， 数十クロック程度のタスクサイズに対して，数十〜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． そのため， 本手法ではスタティックスケジューリングによりコンパイル時にタスクをプロセッサにスケジュールを行うことが必要となる． しかし， 本プログラムでは条件分岐があるため， そのままではタスクのスタティックスケジューリングが適用できないという問題がある． 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手動で行った．それにより， コントロールフローをすべてデータ依存の形に集約することで， スタティックスケジューリングが適用可能となる． 従来 OSCAR コンパイラではこのように粒度が小さく条件分岐が連続しているようなプログラムに対する検討を行なっていなかったため，現状では手動でタスク融合を行い， その上で OSCAR コンパイラで並列化を試みた．

タスク融合やリネーミング後の MTG を図 4 に示す． 図 4 からわかるように条件分岐や代入文を並列性を損うことが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〜数千クロック程度の処理コストを持っ

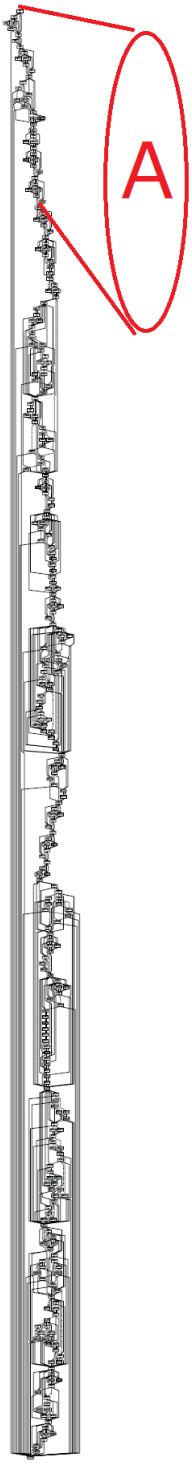


図 2 MdlOutputs 関数内の MFG
Fig. 2 MFG of MdlOutputs Function

た粒度とすることができた． なお， このマクロタスクにおける loop と表示されたブロックは実際のループではなく， チューニングの過程でタスク融合されたブロックが loop と表示されている． また， 並列性に関してもデータ依存のみの二並列程度の並列性抽出が成功した． これにより， 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．

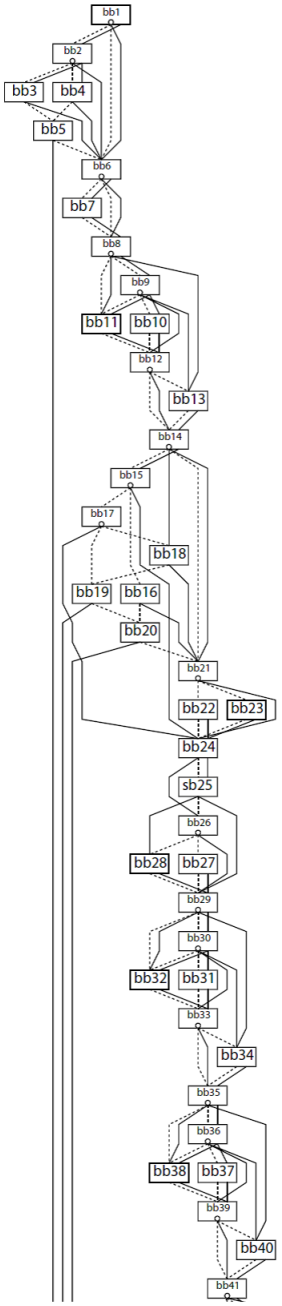


図 3 MdlOutputs 関数内の A に該当する MFG の拡大図
Fig. 3 Enlargement of MFG Part of MdlOutputs

5. 組み込み用マルチコアプロセッサ RP2 及び V850 マルチコア上での性能評価

本章でにおいては， ルネサスエレクトロニクス/日立/早稲田大学で開発した情報家電用マルチコア RP2 及び，V850 マルチコア上で基本エンジン制御プログラムの並列化の評価を行う．

5.1 評価環境

情報家電用 RP2 は図 5 に示すアーキテクチャ構造を取り，SH4A(SH-X3) コアを 8 コア搭載したホモジニアスマルチコアである．RP2 では各コアの動作周波数を 600MHz，

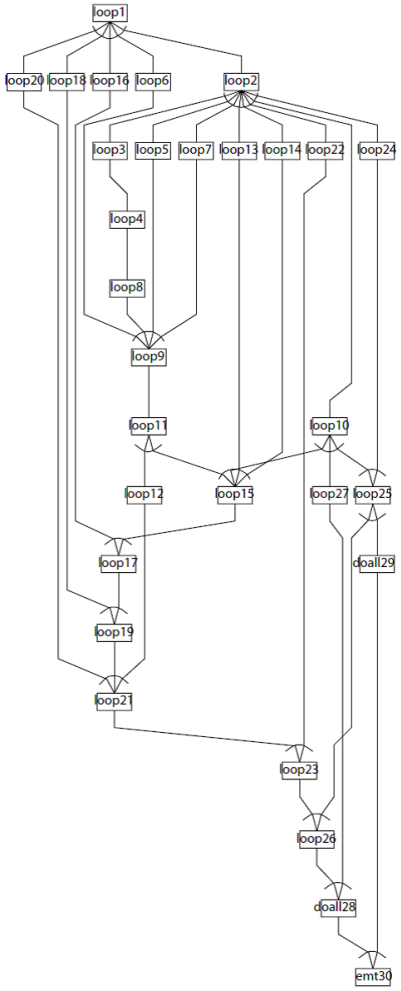


図 4 タスク融合後の MdloOutputs 関数の MTG
Fig. 4 MTG of MdlOutputs Function after Task Fusion

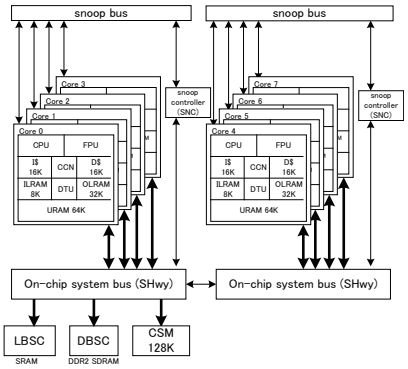


図 5 情報家電用マルチコア RP2 の構造
Fig. 5 Architecture of RP2

300MHz，150MHz，75MHz に独立して変更することが可能である．また， 各コアには，中央演算処理装置 (CPU)，ローカルプログラムメモリ (LPM)，ローカルデータメモリ (LDM)，分散共有メモリ (DSM)，そして CPU と非同期に動作しデータ転送を行うことができるデータ転送ユニット (DTU) 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 (OnChipCSM) やオフチップ集中共有メモリ (OffChipCSM) に接続されている．表 1 の 600MHz 部分を比較してわかるように LDM は read が 1 クロック， write が 1 クロック， リモート DSM は read が 22 クロック， write が 12 クロックのようにメモリアクセスレイテンシがそれぞれ異なり，明示的なメモリ管理を行うことで効果的な性能を引き出すことが可能となっている．RP2 での評価では極力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し， 度々使われる入力データは， それぞれのローカルデータメモリに配置し， これらに対してノンキャッシャブルにアクセスを行う．また， バスの動作周波数を 300MHz と固定しているため， 表 1 に示すようにプロセッサの動作周波数を 300MHz， 150MHz， 75MHz と低くするとメモリは相対的に低レイテンシとなる．例えば， DSM のリモートアクセスが 600MHz のとき read が 22 clock cycle，write が 12 clock cycle に対し， 75MHz のとき read が 6 clock cycle， write が 4 clock cycle である． 75MHz の際は DSM のリモートアクセス， オンチップ CSM 及び， オフチップ CSM のメモリアクセスはすべて 10 clock cyle 以下の低アクセスレイテンシとなる．

次に現在エンジン制御で使用されている V850 ベースのマルチコアは図 6 に示すような構造 [10] である．V850 マルチコアは 80MHz 2 コアでそれぞれのコアに双方向アクセス可能な RAM を持ち， 低レイテンシなデータアクセスが可能である． 特に度々使われる入力データに関しては， 両方の RAM に配置し， RAM のリモートアクセスの削減を行った． 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．

表 1 各周波数毎における RP2 上のそれぞれのメモリにかかるレイテンシ (clock cycle)

Table 1 Latency of Each Memory in RP2(clock cycle)

	600MHz		300MHz		150MHz		75MHz	
	R	W	R	W	R	W	R	W
LDM	1	1	1	1	1	1	1	1
DSM(ローカル)	2	2	2	2	2	2	2	2
DSM(リモート)	22	12	12	7	8	5	6	4
オンチップ CSM	22	12	12	7	8	5	5	4
オフチップ CSM	65	55	33	28	18	15	10	8

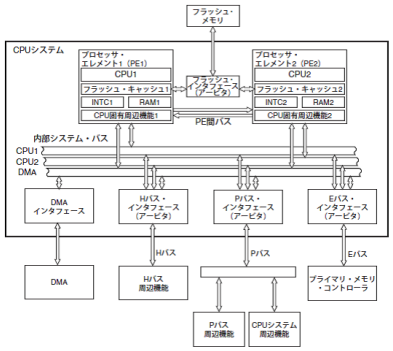


図 6 V850 の構造

Fig. 6 Architecture of V850

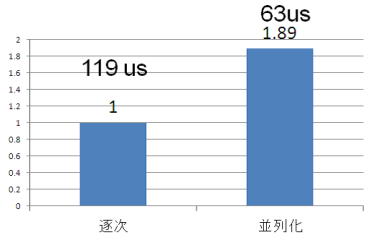


図 7 情報家電用マルチコア上の速度向上率

Fig. 7 Speed Up Ratio on Multicore RP2

フラッシュメモリは共有で， 両 PE に命令供給するが，キャッシュメモリはは独立である．

5.2 情報家電用マルチコア RP2 上での並列処理性能

今回は V850 の動作周波数に近い環境の RP2 上で評価を行うため，75MHz を基本基本周波数として評価を行った．また， 本稿の評価はエンジン制御に関する主要な演算処理を行う MdlOutputs に対しての評価結果である．なお， 本稿では， 現在のエンジン制御系では 2 コアのマルチコアが検討されているため， 2 コア上までの評価を行う．

その結果， 逐次に対する 1 回転あたりの MdlOutputs 関数の速度向上率を図 7 に示す． 逐次に比べ，並列化により実行時間が 119us から 63us まで短縮し， 1.89 倍の速度向上が得られた． また， RP2 上で動作周波数を 600MHz から 300MHz,150MHz,75MHz と変化させたときの (バスの動作周波数は 300MHz に固定) 1 回転あたりの

表 2 各動作周波数毎における MdlOutputs 関数 1 回転あたりの実行時間 (us)

Table 2 Time every Frequency(us)

	75MHz	150MHz	300MHz	600MHz
逐次	119	63	36	22
並列化	63	34	19	13

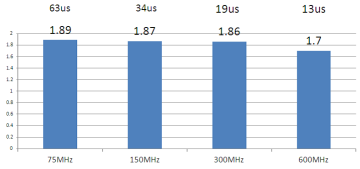


図 8 動作周波数毎の速度向上率

Fig. 8 Speed Up Ratio every Frequency

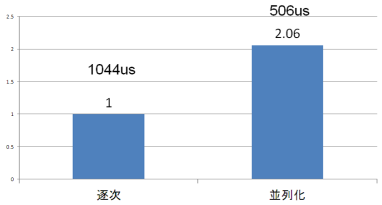


図 9 V850 におけるサイズ優先最適化及び速度優先最適化なしの場合の速度向上率

Fig. 9 Speed Up Ratio on V850 with No Native Compile Optimiztion

MdlOutputs 関数の実行時間を表 2 に、逐次に対する 1 回転あたりの MdlOutputs 関数の速度向上率を図 8 に示す．表 275MHz のとき逐次で 119us， 2 コア並列化で 63us に対して， 600MHz では逐次で 22us， 並列化で 13us まで実行時間減少する．実行時間は動作周波数増加に伴いスケールして減少する一方で， 図 8 に示す速度向上率に関して，75MHz は 1.89 倍であるが， 600MHz は 1.7 倍であるように動作周波数の増加に伴い， 低下していることがわかる．これは， 前述のようにプロセッサの動作周波数が低いほど相対的なメモリアクセスレイテンシが小さくなるため，最も動作周波数の低い 75MHz のときに最大の速度向上率 (1.89 倍) が得られたものである．

5.3 V850 上での並列処理性能

V850 で評価した場合の逐次に対する 1 回転あたりの MdlOutputs 関数の速度向上率を図 9 示す．実行時間に関しては逐次が 1044us， 2 コア並列化が 506us である． 速度向上率に関しては逐次に対して並列化で 2.06 倍の速度向上が得られた．これはプログラムがフラッシュメモリに保持されているため， 2 コアでは各コアが命令キャッシュを搭載するため，命令供給率が倍増し，2 倍以上のスーパーリニアな結果が得られたと考えられる．

実際の車載で用いられるサイズ優先最適化の場合の速度

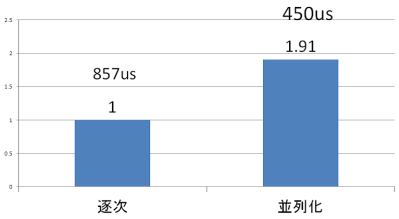


図 10 V850 におけるサイズ優先最適化の場合の速度向上率
Fig. 10 Speed Up Ratio on V850 with Size Optimization

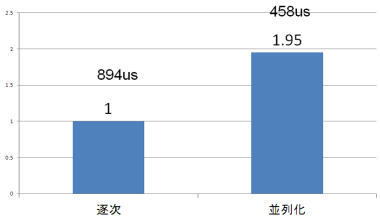


図 11 V850 における速度優先最適化の場合の速度向上率
Fig. 11 Speed Up Ratio on V850 with Speed Optimization

向上率を図 10 示す． 1 回転あたりの MdlOutputs 関数の実行時間に関しては逐次が 857us， 並列化が 450us である． 速度向上率に関しては逐次に対して並列化で 1.91 倍の速度向上が得られた．

サイズ優先最適化の場合では， 最適化なしの場合と比べて，1 回転あたりの MdlOutputs 関数の実行時間は大きく減少している一方で，速度向上率 (1.91 倍) は最適化なしの場合の場合 (2.06 倍) より若干低い結果となっている．また， 速度優先最適化の場合の速度向上率を図 11 に示す．図 10 と図 11 を比較してわかるようにサイズ優先最適化の場合が速度優先最適化の場合より実行時間が短いという結果となっている． これはフラッシュメモリアクセス回数が減少したためと考えられる．

6. まとめ

本稿ではエンジン制御の C プログラムの並列化を提案した． 今回は研究の第一段階として， タスク粒度の細かい条件分岐に対して， 手動でタスク融合し， スタティックスケジューリングが適用できるようにして， マクロタスクグラフを OSCAR 自動並列化コンパイラで並列化を行い，組み込みマルチコアプロセッサ上で並列処理性能の評価を行った． その結果， 手動でマルチコア上での並列化が困難であったエンジン制御プログラムにおいて， 現在自動車への導入が検討されている 2 コアまでを対象として評価を行った． SH4A ベース RP2 マルチコア 2 コア上で 1 コアの 1.89 倍， V850 ベースマルチコア 2 コア上で 1 コア 2.06 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で高速化が実現したため，モデルベース開発された自動車エンジン制御プログラム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． エンジン制御マルチコアによる高速化により， 自動車の燃費が向上または新たな機能の等による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．

今後の展望としては，本稿で手動で行ったタスク融合等を自動化できるように OSCAR コンパイラの改良を試みる． また， エンジン制御のみならず様々な Simulink 上でモデルベース開発された C プログラムを自動並列化するなどが挙げられる．

参考文献

[1] MathWorks: Simulink, http://www.mathworks.co.jp/products/simulink/.
[2] MathWorks: Real-Time Workshop, http://www.mathworks.co.jp/products/simulink-coder/.
[3] Mase, M., Onozaki, Y., Kimuraa, K. and Kasahara, H.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *Proc. of 15th Workshop on Compilers for Parallel Computing* (2010).
[4] 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).
[5] Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., Ishizaka, K. and Kasahara, H.: Multigrain Parallel Processing on Compiler Cooperative Chip Multiprocessor, *Proc. of 9th Workshop on Interaction between Compilers and Computer Architectures (INTERACT-9)* (2005).
[6] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc of The 13th International Workship on Languages and Compilers for Parallel Computing(LCPC2000)* (2000).
[7] 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).
[8] 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).
[9] 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 Workship on Languages and Compilers for Parallel Computing(LCPC2009)* (2009).
[10] RENESAS: RENESAS, http://documentation.renesas.com/.