Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

自動車リアルタイム制御計算の 複数クラスタ構成マルチコア上での並列化

宮田仁[†1]　島岡護[†1]　見神広紀[†1]　西博史[†2]　鈴木均[†2]　木村啓二[†1]　笠原博徳[†1]

概要：1 チップに集積するプロセッサコア数が増加するにつれ，少数のコアから構成されるクラスタを複数 1 チップ上に搭載し，またメモリ構成も各コアのローカルメモリ，及びクラスタのクラスタローカルメモリと階層的に配置するアーキテクチャも見られるようになってきた．リアルタイム制御用の車載マルチコアマイクロコントローラは通常デッドラインを厳守するため，キャッシュが搭載されておらず，ローカルメモリが使用されており，ローカルメモリを有効に活用することが更なる高速化を目指すうえで重要である．さらにこのようなアーキテクチャでは，クラスタを考慮に入れたタスクスケジューリングアルゴリズムを用いることで，コア間及びクラスタ間のデータ転送を最小化する必要がある．本稿では，クラスタを考慮したスケジューリングアルゴリズムと参照情報と実機情報を利用した自動最適メモリ配置アルゴリズム手法を用いる並列化手法を提案する．本手法の性能評価を 4 コア×2 クラスタの 8 コア構成の車載用マルチコアコントローラ上で行ったところ，エンジン制御アプリケーションで 8 コア時に 7.77 倍の速度向上，モーター制御アプリケーションで 4 コア時に 2.22 倍の速度向上，2 モーター制御アプリケーションで 8 コア時に 2.96 倍の速度向上が得られた．

キーワード：マルチコア，並列処理，自動並列化コンパイラ，スケジューリング，ローカルメモリ

1. はじめに

自動車制御アプリケーションの特徴として，デッドラインを厳密に守らなければならないハードリアルタイム制約が挙げられる．また自動車の電子制御高度化に伴い車載アプリケーションの複雑化が進んでいることが知られている．この 20 年で車載システムのソースコードは約 1 千倍に増大しており[1]，車載アプリケーションは 1 億行にも到達している[2]．デッドライン制約を満たすためには高速処理は必須であるが，プロセッサの最大動作周波数は既に限界に達していると言われており[3]，また，最大動作周波数で動作させてしまうと消費電力が大きいすなわち発熱が大きいという問題もある．そのため，車載シングルコアマイクロコントローラでは対応できなくなることが考えられ，マルチコア化が検討されている．このようなマルチコアプロセッサ上で高速処理を実現するためには，アプリケーションの並列化が必須となる．

近年，1 チップに集積するプロセッサコア数が増加するにつれ，少数のコアから構成されるクラスタを複数搭載し共有メモリで接続したマルチコアがリアルタイム制御用マルチコアで見られるようになってきている．たとえば本稿で評価対象とする 8 コアの車載用マルチコアマイクロコントローラは，4 コア×2 クラスタの 8 コア構成を想定している．またリアルタイム制御用車載マルチコアマイクロコントローラは通常デッドラインを厳守するためキャッシュが搭載されておらず，その代わりにローカルメモリが使用されており，ローカルメモリを利用した並列処理を行うことが可能である．しかし手動で並列化とメモリ配置を行うためには経験者でも膨大な時間を要する．

自動並列化では複数のプロセッサに処理すべきタスクを割り当て，実行時間を最小にしようとするスケジューリングアルゴリズムを使用する．マルチプロセッサスケジューリング問題はほとんどの場合で強 NP 問題であることが知られている[4]．ハードリアルタイム制約を満たす実行時ダイナミックスケジューリング手法として RM(Rate-Monotonic)スケジューリングや EDF(Earliest-Deadline-First)スケジューリングが知られており，RM スケジューリングでの実用的なタスク配置についての研究がなされている[5]．また CSP 理論を基に Matlab Simulink のモデルから並列性を解析し，タスクをマルチコアへ割り当てるスタティック手法も研究されている[6]．

筆者らが従来から開発している OSCAR 自動並列化コンパイラ[7]では，このようなスケジューリング問題に対して，ダイナミックスケジューリングオーバーヘッド及びデータ転送オーバーヘッド削減のため，スタティックヒューリスティックアルゴリズムである CP/DT(Critical Path / Data Transfer)法[8]を用いてスケジューリングを行う．CP/DT 法は，タスクのプロセッサ割り当て時に，まずレディ(実行可能)タスク集合を CP 長の大きい順にソートしたプライオリティリストを作成し，プライオリティリストの先頭(最も CP 長が大きい)タスクを選択，それらのタスクを，データのプレロード，ポストストアを考慮して，局所的なデータ転送が最小になる組み合わせでプロセッサに割り当て，タスクのプロセッサへの割り当て決定時に，CP 長も実行時に必要なデータ転送時間も等しいタスクとプロセッサの組が複数ある場合には，直接後続タスク数が多いものを優先するというアルゴリズムである．

ここで，OSCAR コンパイラにおける最終的なメモリ配

†1 早稲田大学 理工学術院 情報処理学科
Waseda University.
†2 ルネサスエレクトロニクス株式会社
Renesas Electronics Corporation

Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

置の決定は，タスクのスケジューリング後に行う．すなわち，スケジューリングの時点ではクラスタ構成を考慮していない従来のスケジューリングアルゴリズムでは，ローカルメモリの有効活用ができないため，クラスタ構成を考慮に加えたスケジューリングアルゴリズムが必要である．

本稿では，前述のようなクラスタ構成のマルチコアに対して，コア間及びクラスタ間のデータ転送を考慮しつつ負荷分散を行う CP/DT/CL 法を提案する．さらに参照情報(変数サイズ・アクセス回数)と実機情報(メモリ構成・メモリ容量・メモリアクセスレイテンシ)を利用した自動最適メモリ配置アルゴリズム手法を提案する．これらの提案手法を，OSCAR 自動並列化コンパイラと新たに実装した自動メモリ配置機能，新スケジューリングアルゴリズムを利用して自動車制御アプリケーションの並列化を実施し，その有用性を評価する．

以下本稿では，第 2 節にて本手法の対象となるアーキテクチャについて説明する．第 3 節にて提案手法であるスケジューリングアルゴリズムとメモリ配置アルゴリズムについて説明する．第 4 節にて OSCAR 自動並列化コンパイラついて説明する．第 5 節にて評価環境となるアーキテクチャと評価アプリケーションについて述べる．第 6 節にて提案手法実装後の OSCAR 自動並列化コンパイラを用いたアプリケーションの評価結果について述べる．

2. 対象マルチコアアーキテクチャ

今回提案する手法の対象となるアーキテクチャについて説明する．対象とするのは複数クラスタ構成マルチコアアーキテクチャであり，少数のマルチコア設計をクラスタと扱ったものである．また，コア毎にローカルメモリ(LRAM)を持ち，クラスタ毎にクラスタローカルメモリ(CRAM)を持つ．このアーキテクチャの例を図で表したものが以下の図 1 である．

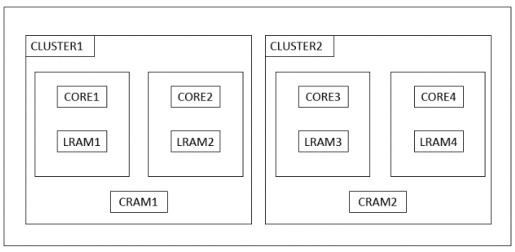

図 1　複数クラスタ構成マルチコアアーキテクチャの例

また，ローカルメモリはグローバルアドレススペースにもマップされているため，各コアは自身が持つ LRAM 以外のローカルメモリにもアクセスが可能であるが，アクセスするローカルメモリの場所によってロードとストアのアクセスレイテンシが異なるものとする．

3. 提案手法

本節にて提案するスケジューリング手法とメモリ配置手法について説明する．並列化コンパイル時は，まずクラスタを考慮したスケジューリングによりデータ転送コストを最小化し，その後，メモリ配置手法によりメモリアクセスコストを最小化する．

3.1 スケジューリングアルゴリズム

3.1.1 CP/DT/CL 法

次に提案手法である CP/DT/CL(Critical Path / Data Transfer considering Cluster Configuration)法について説明する．これは CP/DT 法で考慮していないクラスタという概念を加えてスケジューリングを行うことで，クラスタの特性を有効に活用することを目的として，CP/DT 法を拡張したスケジューリングアルゴリズムである．以下の手順でタスク割り当てを行う．

1. タスクのプロセッサ割り当て時に，まずレディ(実行可能)タスク集合を CP 長の大きい順にソートしたプライオリティリストを作成する
2. プライオリティリストの先頭(最も CP 長が大きい)タスクを基準タスクとして選択する
3. 基準タスクの先行タスクがどちらのクラスタに割り当てられているかを確認し，どちらのクラスタに割り当てるべきかを決定する
4. プライオリティリストから，基準タスクと同じクラスタに割り当てるべきタスクを探索する
   (a) 基準タスクの後続タスクを調べる
   (b) それらのタスクの先行タスクの内，プライオリティリスト上にあるタスクと一致するものを同じクラスタに割り当てるべきタスクとする
5. 同じクラスタに割り当てるべきタスクの CP 長に他クラスタ割り当て時のデータ転送コストを加えた CPwithDTpath を計算する
6. プライオリティリストの内，基準タスク以外を CPwithDTpath でソート
7. 基準タスクを，データのプレロード，ポストストアを考慮して，局所的なデータ転送が最小になる組み合わせでプロセッサに割り当てる
8. タスクのプロセッサへの割り当て決定時に，CP 長も実行時に必要なデータ転送時間も等しいタスクとプロセッサの組が複数ある場合には，直接後続タスク数が多いものを優先する

3.1.2 タスクスケジューリング実行例

以下に CP/DT/CL 法でタスクスケジューリングを行う際の処理フローの例を図 2 と共に示す．

Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

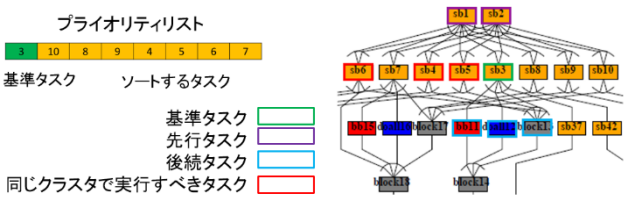

図 2　CPDT/CL 法実行例

ここではプライオリティリストの先頭であるタスク 3 を基準タスクとして処理を進める．まずタスク 3 の先行タスクであるタスク 1,2 がどちらに割り当てられているかを確認する．そして，タスク 3 の後続タスクである 11,12,13 の先行タスクを調べ，プライオリティリストのタスクと一致するものを調べ上げる．その結果，タスク 4,5,6 が当てはまる．タスク 4,5,6 に対して CPwithDTpath を計算し，プライオリティリストをソートする．その結果プライオリティリストは以下の図 3 のように変化する．

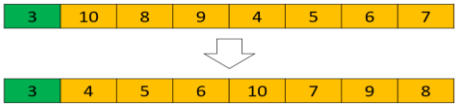

図 3　プライオリティリストの変化

その後，タスク 3 をコアに割り当て，次にタスク 4 を基準タスクとして，処理を続けていく．

このように，プライオリティリストの先頭タスクを基準として，そのタスクと関係性が高いタスクを探索しながらタスクスケジューリングを行うことで，クラスタを有効活用した割り当てを目指す．先ほど提案した自動最適メモリ配置アルゴリズム手法は，タスクスケジューリング後に参照情報の取得，割り当てを行うため，スケジューリング時にはクラスタを考慮していなかった．CP/DT/CL 法を用いることでスケジューリング時にクラスタを考慮し，この問題点を解決した．

3.2　メモリ配置アルゴリズム

3.2.1 処理アルゴリズム

自動最適メモリ配置アルゴリズム手法の処理アルゴリズムについて図を用いて説明する．このアルゴリズムを図で示したものが以下の図 4 である．

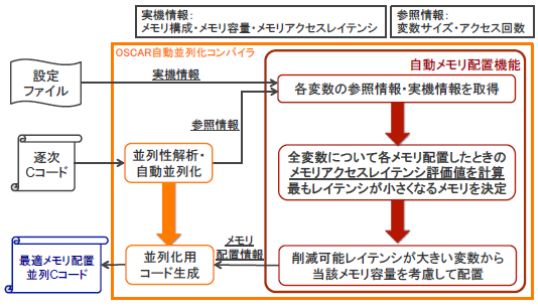

図 4　自動最適メモリ配置アルゴリズム

1. アプリケーションを解析することで参照情報を作成
2. 参照情報と実機情報から必要なデータを取得
3. 各変数について，それぞれの LRAM，CLAM に配置した時のメモリアクセスレイテンシ評価値を計算
4. メモリアクセスレイテンシ評価値が最小となる LRAM・CRAM を決定
5. それぞれの変数について LRAM に配置したときに削減できるクロック数が多い順に並び替える
6. 当該メモリ容量を考慮しながら配置

3.2.2 入力情報

本手法で利用する入力情報ついて述べる．入力情報として，参照情報と実機情報を用いる

(a) 参照情報

OSCAR 自動並列化コンパイラが生成した変数の参照情報を利用する．参照情報には変数名，サイズ，各コアでの定義回数，参照回数が記述されている．

(b) 実機情報

第 2 節で述べた評価環境となる実機の情報を利用する．実機情報にはコア構成，クラスタ構成，各メモリアクセスレイテンシ，各メモリ容量が記述されている．

3.2.3 メモリアクセスレイテンシ評価値計算方法

最適な配置を決定するためのメモリアクセスレイテンシ評価値の計算方法について述べる．2 コア実行時の例を以下に示す．その際の計算フローを図で表したものが以下の図 5 である．

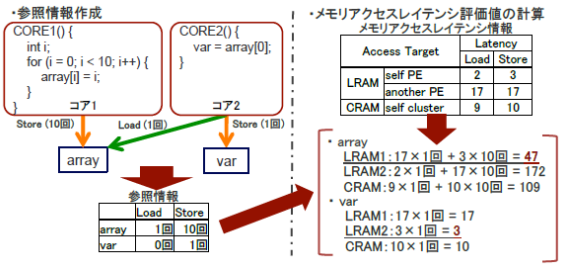

図 5　メモリアクセスレイテンシ計算フロー

まずアプリケーションから変数の参照情報が生成される．この例ではコア 1 が array に Store アクセス 10 回，コア 2 が array に Load アクセス 1 回，var に Store アクセス 1 回していることが解析結果として得られる．実機情報としてメモリアクセスレイテンシ情報が与えられ，これらの情報を用いて，各変数について評価値の計算を行う．

まず array を LRAM1 に配置した場合にはコア 2 から 17 クロックの Load リモートアクセスが 1 回，コア 1 から 3 クロックの Store セルフアクセスが 10 回で合計 47 クロックのメモリアクセスレイテンシが必要になる．同様

Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

に LRAM2 に配置した場合と CRAM に配置した場合の総メモリアクセスレイテンシ計算を行う．そして最後にメモリアクセスレイテンシが最小となるメモリである LRAM1 が選ばれ，array の最適メモリ配置場所は LRAM1 であることが決定される．そして var についても同様の計算を行い，LRAM2 に配置する場合が最もメモリアクセスレイテンシが小さくなることが決定される．このようにメモリアクセスレイテンシ評価値はアクセス回数にメモリアクセスレイテンシを積和演算することで得られ，最もメモリアクセスレイテンシが小さいメモリを配置場所として決められる．また共有変数であっても，各コアから Load アクセスしかされないような変数の場合には LRAM のセルフ領域に配置するように設定する．

例として挙げた図 2 では 1 クラスタの 2 コアを利用した場合であったが，2 クラスタ 8 コアを利用する場合には LRAM1 から LRAM8 までの 8 つと CRAM0 と CRAM1 の 2 つについて計算を行い，最もメモリアクセスレイテンシが小さくなるメモリを決定する．またこのとき各変数はメモリアクセスレイテンシが最小になるメモリの他に，LRAM 候補と CRAM 候補の 2 つのメモリ候補メモリを保持するものとする．

3.2.4 配置優先度決定方法

次に変数の配置優先度について述べる．変数をメモリ配置するに当たり，限られたメモリ容量を最大限活用するためには，メモリ配置効果の高い変数から割当てていくことが重要であると考えられる．そこで変数配置優先度として，以下のように定めた．

1. 「CRAM 評価値- LRAM 評価値」を LRAM 配置で削減できるクロック数として求める
2. 「LRAM 配置で削減できるクロック数」が大きい順に並び替える

このような優先度決定方法にて，すべての変数に順位をつける．そして優先度の高い変数からメモリ割当てが行われる．

3.2.5 メモリ割り当て方法

メモリ配置は前節で述べたように，優先度に基づいて割当てられる．このとき実機情報として与えられていた，各メモリの容量を用いる．メモリ割り当ては以下のように行われる．

1. 変数のサイズと配置場所を取得
2. 当該メモリの残り容量取得
3. もし「当該メモリの残り容量＞ 変数サイズ」ならば割当て
   (a) 当該メモリの残り容量を「当該メモリの残り容量- 変数サイズ」に変更する
4. もし「当該メモリの残り容量＜ 変数サイズ」ならばもう一方のメモリに配置
   (a) もう一方のメモリにも配置できない場合には配置をしないように設定

また LRAM のセルフ領域に配置される変数については，まずセルフ領域の残りメモリ容量を確認し，配置できれば割当て，配置できなければ上記処理フローにてメモリ割り当てされる．

以上により適切なメモリへの配置が完了する．

4. OSCAR 自動並列化コンパイラ

4.1 OSCAR 自動並列化コンパイラの概要

OSCAR 自動並列化コンパイラとは，遂次的に記述された C 言語のプログラムから並列性を自動で抽出し，並列化された C 言語のプログラムを自動生成するものである．並列化手法としてはマルチグレイン並列処理を特徴としている．マルチグレイン並列処理とは，ループやサブルーチンなどの粗粒度タスク間の並列性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性を利用する中粒度並列処理，基本ブロック内のステートメントレベルの並列性を利用した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の並列化を行う手法である．

4.2 粗粒度タスク並列化

ここでは OSCAR 自動並列化コンパイラを用いた粗粒度タスク並列化について述べる．OSCAR 自動並列化コンパイラではプログラムを基本ブロック(BB)，繰り返しブロック(RB)，サブルーチンブロック(SB)の 3 種類のマクロタスク(MT)に分割し，RB，SB 内部に粗粒度タスク並列性が存在する場合には，内部も同様に分割することで，階層的な MT を生成する．MT 間のデータ依存関係とコントロールフローを解析して，マクロフローグラフ(MFG)を生成，MFG に再早実行可能条件解析を適用し，MT 間の並列性を抽出してマクロタスクグラフ(MTG)を生成する．最後に MTG 上の MT を複数のプロセッサに割り当てて実行することにより並列処理を実現する[9][10].

第 3 節で提案した CP/DT/CL スケジューリングは，上記マクロタスクをプロセッサに割り当てる際に使用する．さらにスケジューリング後にメモリ配置手法を適用する．

5. 評価マルチコア及びアプリケーション

5.1 評価マルチコアアーキテクチャ

今回用いた環境は車載用の RH850 をベースとした CPU コアを搭載した評価環境であり，4 コア×2 クラスタの 8 コア構成のマルチコアマイクロコントローラである．LRAM が各コアに，CRAM が各クラスタに搭載されている．また LRAM にはセルフ領域と呼ばれる領域が存在し，変数の配

Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

置をセルフ領域に指定すると，その変数は全コアの LRAM に配置される．各コアからそれぞれのローカルメモリにアクセスするレイテンシが異なるという特徴があり，そのレイテンシを以下の表にまとめる．

表 1　各ローカルメモリのアクセスレイテンシ

	レイテンシ	
アクセス先	ロード	ストア
自 LRAM	2	3
自クラスタ内他 LRAM	17	17
他クラスタ内他 LRAM	22	22
自 CLAM	9	10
他 CRAM	20	21

この表からも分かる通り，自クラスタ内のデータアクセスは自 LRAM，自 CRAM，自クラスタ内他 LRAM の順で高速である．よってなるべく自身の LRAM にデータを配置するように工夫する必要がある．

5.2 評価アプリケーション

5.2.1 エンジン制御アプリケーション[11]

エンジン制御アプリケーションの特徴として，粒子群最適化(pso)処理の実行コストが全体の 99.3%を占めていることが挙げられる．この pso 処理は 8 つ並んでおり，前半と後半で 2 回行う．pso 処理にそれぞれに依存はないため，並列実行可能タスクと判断されている．さらに，この 8 つの pso 処理は 4×2 の構造となっており，クラスタを活用できるように処理を割り当てることで，ローカルメモリを有効活用することができる．並列度は 7.99 と高く，ms オーダーであり，モーター制御と比べて実行時間が大きいこと，処理で扱う変数の数が多いことが特徴である．

5.2.2 モーター制御アプリケーション

三角関数を用いた算術計算が全体のコストの 78.1%を占めること，並列度(総コスト/クリティカルパス)が 3.45 と低いこと，実行時間がμsオーダーでとても小さいこと，処理で扱う変数の数が比較的少ないことが特徴である．三角関数はルックアップテーブル化をしており，また並列性が小さいため，Simulink モデルから並列性を抽出している．

5.2.3 2 モーター制御アプリケーション

モーター制御アプリケーションを 2 つ並べることで疑似的に並列度を高めたベンチマークアプリケーションである．μsオーダーで実行時間が非常に小さいが，並列度は 6.70 となっている．また，モーター制御を 2 つ並べた設計上，処理は 2 つの塊に分かれており，それぞれ独立して実行することが可能であるため，それぞれをクラスタに分けて実行することでローカルメモリの有効活用が可能である．

6. 性能評価

6.1 性能評価結果

3 つの評価アプリケーションの評価結果を図 6 に示す．

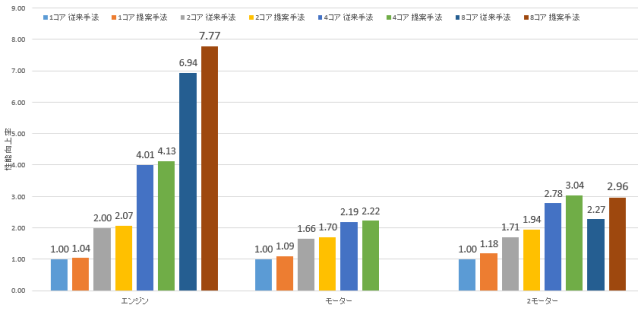

図 6　エンジン制御アプリケーション評価結果

エンジン制御アプリケーションにおいては，従来手法では 2 コアで 2.00.倍，4 コアで 4.01 倍，8 コア時では 6.94 倍の性能向上が得られた．提案手法を用いた場合には 1 コアで 1.04 倍，2 コアで 2.07 倍，4 コアで 4.13 倍の性能向上が得られ，8 コア時では 7.77 倍となった．LRAM を活用したことでの速度の向上が確認でき，またクラスタを越えたアクセスが発生する 8 コアにおいても従来手法から 12%の性能改善が見られた．クラスタを活用するように処理を割り当てたことで，ローカルメモリを有効に活用することができたことも確認できる．

モーター制御アプリケーションにおいては，従来手法では 2 コアで 1.66 倍，4 コアで 2.19 倍の性能向上が得られた．提案手法を用いた場合には 1 コアで 1.09 倍，2 コアで 1.70 倍，4 コアで 2.22 倍の性能向上が得られ，ローカルメモリを活用したことでの速度の向上が確認できた．

2 モーター制御アプリケーションにおいては，従来手法では 2 コアで 1.71 倍，4 コアで 2.78 倍の速度向上が得られたが，クラスタを越えたデータのアクセスが発生する 8 コア時では 2.27 倍と速度が落ち込んでいることが分かる．提案手法を用いた場合には 1 コアで 1.18 倍，2 コアで 1.94 倍，4 コアで 3.04 倍の性能向上が得られ，8 コア時では 2.96 倍となった．ローカルメモリを活用したことでの速度の向上が確認でき，またクラスタを越えたアクセスが発生する 8 コアにおいては従来手法から 30%の性能改善が見られた．

クラスタを越えたアクセスが発生するエンジン制御アプリケーションと 2 モーター制御の 8 コア時のガントチャートをそれぞれ図 7，図 8 に示す．図 7 では，実行時間の大半を占めている pso 処理以外のタスクは実行時間が小さいため，ガントチャート上では見えなくなっている．ガントチャートから 4×2 の pso 処理がそれぞれのクラスタに分けられてスケジューリングされていることが確認できる．図 8 のガントチャートから 2 つ並べたモーター制御の処理がほぼ 2 つのクラスタにそれぞれ割り当てられていることが確認できる．

Vol.2017-ARC-225 No.30
Vol.2017-SLDM-179 No.30
Vol.2017-EMB-44 No.30
2017/3/10

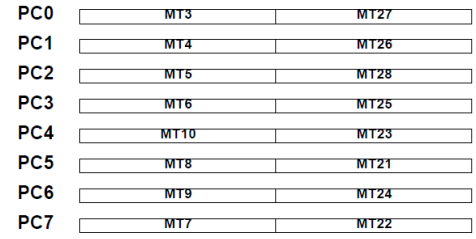

図 7 エンジン制御アプリケーションガントチャート

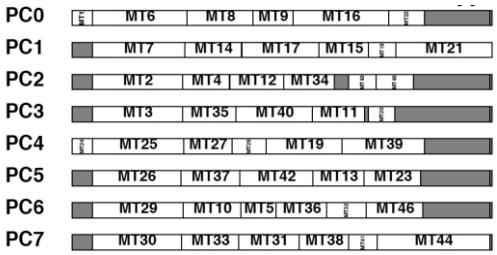

図 8 2 モーター制御アプリケーションガントチャート

6.2 メモリ配置状況

図 9 にエンジン制御アプリケーションのメモリ配置状況を図 10 にモーター制御及び 2 モーター制御のメモリ配置状況を示す.

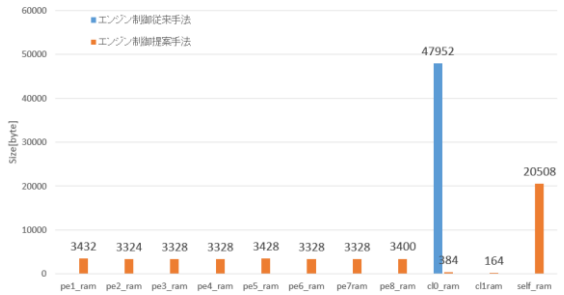

図 9　エンジン制御メモリ配置状況

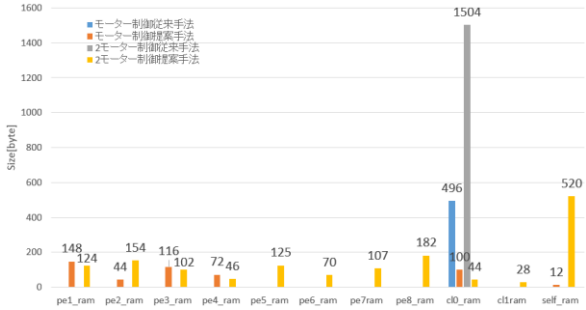

図 10　モーター及び 2 モーター制御メモリ配置状況

図 10 より，エンジン制御 8 コア時にほぼ均等のデータが各 LRAM に配置されていることが確認できる．8 コア時の性能向上は，アクセス速度の速い自 LRAM を有効に活用できた結果である．図 11 より，モーター制御，2 モーター制御においても十分なデータがローカルメモリに配置されていることが確認でき，8 コア時の性能改善はローカルメモリの有効活用のためである．

7. まとめ

本稿では，複数クラスタ構成マルチコア上での自動車リアルタイム制御計算の並列化手法として，クラスタを考慮に入れたタスクスケジューリングアルゴリズムの CP/DT/CL 法と，ローカルメモリに変数を自動で配置する自動最適メモリ配置アルゴリズム手法を提案した．また，提案手法をエンジン制御アプリケーション，モーター制御アプリケーション，2 モーター制御アプリケーションに適用した場合の性能評価を 4 コア×2 クラスタ構成の車載マルチコアマイクロコントローラ上で行い，性能評価を行い，提案手法の有用性を検証した．

その結果，エンジン制御アプリケーションでは 8 コア時に 7.77 倍の速度向上が得られ，従来手法から 12%の性能改善が見られた．モーター制御アプリケーションでは 4 コア時に 2.22 倍の速度向上が得られ，2 モーター制御アプリケーションでは 4 コア時に 3.04 倍，クラスタを越えたアクセスが発生する 8 コア時においても 2.96 倍の性能向上が得られ，8 コア時では従来手法から 30%の性能改善が見られた．

以上のように本手法の有用性を確認できた．

参考文献

[1] 堀川健一，目崎元太，渡辺章代，加櫓武，松本達治，“自動車ソフトウェアの標準仕様"autosar"の評価”,SEI テクニカルレビュー, Vol. 175, pp. 92-97, July, 2009.
[2] Codebases. Millions of lines of code. www.informationisbeautiful.net/visualizations/million-lines-of-code/.
[3] Ralph Mader, Armin Graf, and Gerd Winkler, “ Autosar based multicore soft-ware implementation for powertrain applications”, SAE International Journal of Passenger Cars - Electronic and Electrical Systems, Vol. 8, No. 2, pp. 264-269,2015.
[4] Kasahara H. and Narita S., “Practical Multiprocessor Scheduling Algorithm for Efficient Parallel Processing”, IEEE Trans. Comput., C-33, 11,pp.1023-1029, Nov. 1983
[5] 鈴木紀章,森義和,岩熊憲二,枝廣正人, “ハードリアルタイム処理向けマルチコアタスク配置の評価関数設計”,情報処理学会研究報告,Vol.2011-EMB-20 No.16,Mar,2011.
[6] 大川禎,枝廣正人,久村孝寛, “CSP 理論にもとづいた制御モデルのマルチコア実装向けタスク割当て”.情報処理学会研究報告, Vol.2013-EMB-28 No.23,Mar,2013.
[7] 笠原博徳，尾形航，木村啓二，小幡元樹，飛田高雄，稲石大祐,“マルチグレイン並列化コンパイラとそのアーキテクチャ支援”，電子情報通信学会技術研究報告, ICD,集積回路, Vol. 98, No. 22, pp. 71-76, apr 1998.
[8] 藤原和典,白鳥健介,鈴木真,笠原博徳, “データプレロードおよびポストストアを考慮したマルチプロセッサスケジューリングアルゴリズム”,電子情報通信学会論文誌,D-1 Vol. J75-D-I No.8 pp495-503,Aug,1992.
[9] 小幡元樹,笠原博徳,木村啓二, “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”,情報処理学会,2002.
[10] 笠原博徳, “並列処理技術”,コロナ社 ,1991.
[11] 山浦和隆,新井邦晴,鈴木均,安達浩次,城倉梨香. “ハイブリッド・エンジン制御におけるリアルタイム・モデル予測制御の並列化とマルチコアへの実装”,自動車技術会,学術講演会前刷集,No147-13,p.1 - ,Oct,2013