Vol.2017-ARC-225 No.11
Vol.2017-SLDM-179 No.11
Vol.2017-EMB-44 No.11
2017/3/9

車載マルチコアシステムの性能見積もり手法

小川 真彩高[1,a)]　本田 晋也[1]　高田 広章[1]

概要：近年車載システムは高機能化してきており，エンジン制御に用いられるパワトレアプリにもマルチコア化が求められている．マルチコアシステムは処理のコア配置や共有データのメモリ配置によって性能が大きく変化するが，これらの配置パターンは膨大な数に及ぶため，実機での性能評価の前に性能見積もりを行い，処理が時間制約を満たす可能性が低いパターンを排除することが必要となる．本研究では車載向けのマルチコアアーキテクチャを対象とし，パワトレアプリを想定した評価ソフトの処理をいくつかの集合に分割し，その集合単位でコアに配置する組合せパターンに対して排他制御やメモリアクセス遅延の影響を考慮した性能見積もりを行う手法を提案する．

キーワード：車載システム，マルチコア，性能見積もり

Performance Estimation Method for Multicore Automotive Systems

Masataka Ogawa[1,a)]　Shinya Honda[1]　Hiroaki Takada[1]

Abstract: In recent years, vehicle systems have become more functionality, and powertrain used for engine control applications are also required to multi-core. In the multi-core system, the performance depends on the core arrangement of processing and memory arrangement of shared data. However, since these arrangement patterns extend to enormous numbers, it is necessary to estimate the performance and exclude patterns with low possibility that processes satisfy the time constraint before before performance evaluation in a micro controller. In this study, as a target of multi-core architecture for automotive applications, we propose the performance estimation method that allocate processing of evaluation software assumed to be the power-train application to several sets and then arrange these sets in the core and then estimate the performance considering the influence of exclusive control and memory access delay.

Keywords: Vehicle control System, Multicore, Performance estimation

1. はじめに

近年燃費向上や排ガス規制，ハイブリッドエンジンへの対応などから，車載制御ソフトウェアに対する要求が高くなってきており，今後も増々その傾向が強くなると予想されている．[1] 自動車に搭載されるエンジン制御ユニット（ECU）のうち，エンジンやトランスミッションといったパワートレインを制御するパワートレイン・アプリケーション (パワトレアプリ) の効率化・高機能化は，それらの要求を満たすために重要である．現在主流のシングルコアでは動作周波数が限界に達していると言われているため，今後のパワトレアプリの高機能化に対応するにはマルチコアでの実行が不可欠である．

パワトレアプリは時間制約に従って動作する複数の処理で構成されており，その中でも重要な処理が時間制約を満たさない場合，人命に関わる重大な事故に繋がる可能性があるため，パワトレアプリのリアルタイム性を高いレベルで保証する必要がある．シングルコアシステムについてはすでにリアルタイム性を保証する手法が提案されているが [3]，マルチコアシステムではバス衝突や共有データのアクセス排他などの影響で予想が困難であり，シングルコア

[1] 名古屋大学大学院情報科学研究科, 名古屋市
Graduate School of Information Science, Nagoya University, Nagoya-shi, 464-8603 Japan
[a)] masa-bach@ertl.jp

Vol.2017-ARC-225 No.11
Vol.2017-SLDM-179 No.11
Vol.2017-EMB-44 No.11
2017/3/9

システムよりもリアルタイム性の保証が難しい．また，マルチコアシステムは処理のコア配置や共有データのメモリ配置によって性能が大きく変化するが，処理の配置パターンの組合せは膨大な数に及ぶため，手作業によってそれらの配置パターンに対して実機による検証を行うのは困難である．そのため，考えられる配置パターンすべてに対して性能見積もりを行い，リアルタイム性を保証できないパターンを排除した上で適切な処理や共有データの割当ての探索をサポートするツールが必要となる．

本研究ではパワトレアプリを想定した評価ソフトを車載向けのマルチコアアーキテクチャに実装した場合の，排他制御やメモリアクセス遅延の影響を考慮した性能見積もりを行う手法を提案する．

まず，組合せ数を削減するために処理をデータ依存性や制御依存性を考慮して幾つかの集合に分割する．次にその集合単位で処理をコアに配置するすべてのパターンに対して共有データのメモリ割当てを行い，その後排他制御やメモリアクセス遅延の影響を考慮した性能見積もりを行い，処理が時間制約を満たさないパターンを排除する．最後に，残った配置パターンに関して得られた見積もり値を出力する．

2. パワートレインアプリケーションの評価ソフト

本章では，本研究で対象とするパワートレインアプリケーション (パワトレアプリ) を想定した評価ソフト（以下，評価ソフト）ソフトウェア構成について述べる．評価ソフトは複数の処理と共有データから構成されている．

2.1 処理

評価ソフトでは，時間もしくはエンジンの回転角に同期して処理が実行される．評価ソフトは多数のモジュールで構成されており，そのモジュールの中の機能毎に実行契機や周期，オフセットが異なる．そのため，機能の追加，周期変更を容易に実現できるように図 1 のような構成を持っている．評価ソフトはエンジン，トランスミッションなど，同一の機能を実現するための処理の集合 (セット) で構成される．セット内には機能を構成する処理の集合セットマネージャを持つ．セットマネージャからは同一の周期や優先度を持つ処理の集合であるサブセットが呼び出される．サブセットは処理の単位となる公開関数を呼び出す．

2.2 共有データ

評価ソフトでは複数の処理間で共有データの引き渡しを行う．共有データの読み込み，書き込みは公開関数によって行われる．また，公開関数が同一の共有データへ読み込みまたは書き込みを行う回数は 1 回の公開関数呼び出しにつき高々 1 回までである．図 1 に公開関数間での共有データのアクセス例を示す．共有データは一貫性や安定性を保つため排他制御をする必要がある．

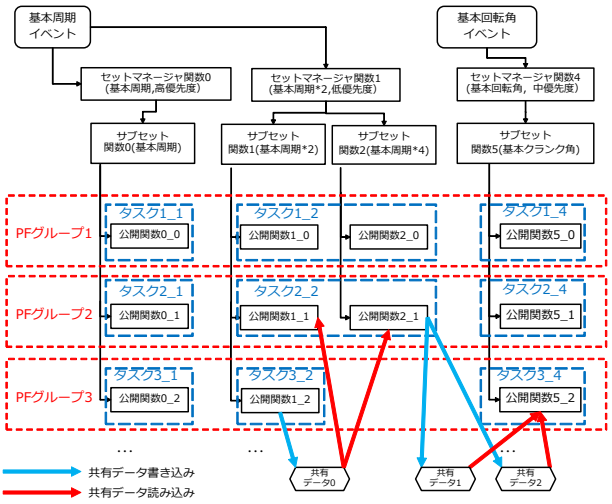

図 1 評価ソフトの構成

3. パワトレアプリにおけるシングルコアシステムのスケジューリング解析手法

本章では，既存の車載向けシングルコア向けのリアルタイムスケジューリング理論について述べる．タスクが常にデッドライン以内に収まっていることをスケジュール可能と呼び，スケジュール可能であるかどうかを解析することをスケジューリング解析と呼ぶ．

パワトレアプリでは起動タイミングによって処理内容が変化するため，タスクの最大実行時間と平均実行時間には大きな差が出る可能性がある．よって，タスクの最大実行時間をそのままスケジューリング解析に利用すると悲観的な解析結果となることが予想される．このような問題に対処するため，タスクが複数の実行時間を持つことを前提としたマルチフレームタスクモデル [2] を導入する．

マルチフレームタスクモデルとは，周期ごとの最大実行時間があるパターンに従って変化するようなタスクを扱うためのモデルである．P_i をタスク i のフレーム長，C_i^j をタスク i の j 番目のフレームの最大実行時間，N_i をタスク i のフレーム数とすると，タスク i のマルチフレームタスク (MF タスク) は $((C_i^0, ..., C_i^{N_i-1}), P_i)$ と表現される．例えば MF タスク $((1,2,3),4)$ の場合，時刻 0 で最大実行時間 1, 時刻 4 で最大実行時間 2, 時刻 8 で最大実行時間 3, 時刻 12 で最大実行時間 1 のタスクとなる．

MF タスクセットのスケジュール可能性判定を効率よく行う方法として Maximum Interference Function（MIF）[3] を用いたものが提案されている．MIF $M_i(t)$ は長さ t の間にタスク i より優先度の高いタスクがタスク i の実行を邪魔する最大時間である．図 2 に MF タスク $((1,2,3),4)$ がフレーム 0,1,2 それぞれから開始したときの CPU 利用時間を示す．各フレームから開始したときの CPU 利用時

Vol.2017-ARC-225 No.11
Vol.2017-SLDM-179 No.11
Vol.2017-EMB-44 No.11
2017/3/9

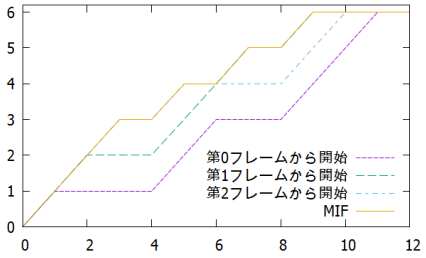

図 2　MF タスク $((1,2,3),4)$ の MIF

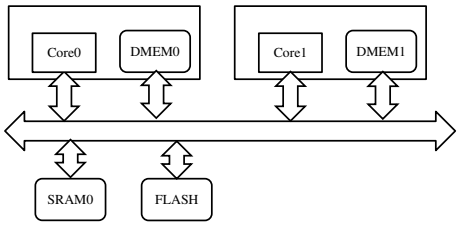

図 3　対象とするハードウェアアーキテクチャ

間の最大値をとる関数が MIF の値となる．

MF タスクのスケジュール可能性は次の式で判定することができる，

$$\exists t, 0 < t \leq D_i^k, \sum_{\forall j \in hp(i)} M_j(t) + C_j^k \leq t \qquad (1)$$

ここで，$hp(i)$ はタスク i よりも高優先度なタスクの集合，D_i^k をタスク i の k 番目のフレームのデッドライン，C_j^k をタスク i の k 番目のフレームとする．

4. 評価ソフトのマルチコア化

本章では評価ソフトのマルチコア化におけるハードウェア環境やソフトウェアの変更点について述べ，マルチコア化の課題について説明する．

4.1 ハードウェアアーキテクチャ

本研究で対象とする車載マルチコアアーキテクチャを図 3 に示す．各コアはバスを介せずにアクセスできる DMEM を持つ．他のコアの DMEM にアクセスする場合はバスを介して行う．コア間で共有して利用するようなデータについては SRAM に配置される．

4.2 ソフトウェアの実現方法

本研究で対象とするソフトウェアアーキテクチャは 2 章で示したとおりである．手作業による公開関数のコア割当て手順は次の様になる．

- 各公開関数を配置するコアを決定する
- 共有データの配置メモリを決定する
- 共有データごとに使用する排他制御機構を決定する
- スケジューリング解析する

スケジューリング解析に関してタスクの応答時間とデッドラインが重要な概念である．あるタスクについて，そのタスクのデッドラインと応答時間の差を余裕時間と呼ぶ．さらにシステム全体でタスクの余裕時間が最も短いタスクをタイムクリティカルタスク (TC タスク) と呼び，TC タスクの余裕時間を最悪余裕時間と呼ぶ，最悪余裕時間はシステム全体での余裕時間の最小値であり，これが 0 以下である場合タスクがデッドラインミスする可能性が発生する．

4.3 マルチコア化における課題

マルチコアシステムでは共有データのメモリ割当ての結果によって排他オーバーヘッドやメモリアクセス時間が大きく変化する．よってシングルコアでのスケジューリング解析とは異なり，これらについてタスクの実行時間とは別に考慮する必要が出てくる．排他オーバーヘッドは排他 API そのものにかかる時間と排他による競合状態による遅延時間の合計である．本来であればこの双方について考慮すべてであるが，競合状態による遅延時間はタスクの実行状況によって大きく変化するため，見積もりを行うことは難しい．従って，本研究では，排他オーバーヘッドは排他 API そのものの時間のみを考慮している．また，公開関数は現状 1,000 個程度であると言われており，それらの割当てパターンは組合せ爆発によって膨大な数になってしまい，その中で選択したパターンが適切であることを示すことは困難であるため，公開関数の配置パターンを削減する仕組みが必要である．

4.4 公開関数グループ (PF グループ) 単位の配置

公開関数の割当てパターン数を減らすために，公開関数の集合を制御依存性やデータ依存性を考慮して適切に 7,8 程度のグループに分割し，そのグループ単位でコア割当てを行う．その公開関数の集合を公開関数グループ (PF グループ) と呼ぶ．図 1 は公開関数の集合を PF グループに割り当てたときの例を示している．同 PF グループに含まれ，同セットマネージャから呼び出される公開関数を一つのタスクで実行する．PF グループをコアに配置するパターン数は第二種スターリング数で求められる．現状，想定されている車載マルチコアマイコンのコア数が 4 程度であり，コア数 4，PF グループ数 8 とした場合の PF グループの配置パターンは 1701 通りである．このパターン数に対して手作業で解析を行うことは困難であるため，性能見積もりツールによる解析が必要である．

本研究では同 PF グループかつ同セットマネージャに属する公開関数を同じタスクに割当てる．従って各タスクはサブセットによって周期の異なる公開関数を含む MF タスクとなる．

5. 車載マルチコアシステムの性能見積もり

ここでは，PF グループをコアへ配置する全パターンに対して性能見積もりを行うことで PF グループの配置の決定を支援するのを目的とする．本研究では，性能見積もりによって各配置パターンにおける最悪余裕時間と CPU 利

Vol.2017-ARC-225 No.11
Vol.2017-SLDM-179 No.11
Vol.2017-EMB-44 No.11
2017/3/9

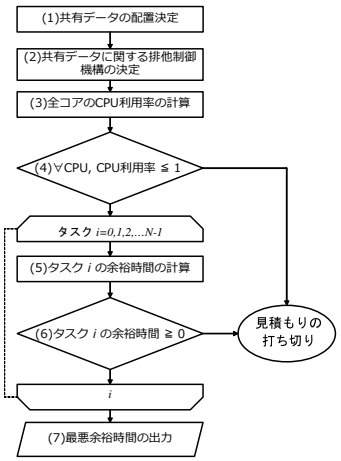

図 4　各割当てパターンに対する性能見積もりのフロー

用率を計算する．すべての配置パターンに対して手作業による見積もりを行うのは困難であるため，提案手法を実現するような性能見積もりツールを実現する．

5.1　見積もりのフロー

本ツールではすべての公開関数グループのコア配置パターンに対して図 4 で示した (1)〜(7) を実施する．

まず (1) 共有データのメモリ割当てを決定し，(2) 各共有データに関して使用する排他制御機構を決定する．次に，(3) 各コアの CPU 利用率を計算し，(4)CPU 利用率が 1 より大きいコアがある場合はスケジュール不可能となるのでその割当てパターンを排除する．その後，(5) 各タスクについて余裕時間を計算し，(6) 1 つでも余裕時間が 0 より小さくなっている場合はその配置パターンを排除する．最後に (7) 各タスクの余裕時間の中から最悪余裕時間を選択し，出力する．

5.2　共有データの配置決定

公開関数のコア割当てによって共有データへの各コアのアクセス数および頻度は変化するため，公開関数のコア割当てごとに適切な共有データのメモリ割当てをする必要がある．共有データ i について以下のアルゴリズムを適用する．

(1) アクセスするコアが単一である場合，そのコアの DMEM に i を配置する．

(2) それ以外の場合，以下により，コア j から共有データ i への読み込み頻度 $rf_{i,j}$ 書き込み頻度 $wf_{i,j}$ を計算する．ただし $rF_{i,j}$ はコア j に配置され共有データ i を読み込む公開関数の集合，$wF_{i,j}$ はコア j に配置され共有データ i に書き込む公開関数の集合，Pe_f を公開関数 f の周期とする．

$$rf_{i,j} = \sum_{f \in rF_{i,j}} \left(\frac{1}{Pe_f}\right), wf_{i,j} = \sum_{f \in wF_{i,j}} \left(\frac{1}{Pe_f}\right) \quad (2)$$

(3) 以下により，共有データ i をメモリ k に配置したときのアクセス時間 $T_{i,k}$ を計算する．ここで $rl_{j,k}$ をコア j からメモリ k を読み込んだときの遅延時間，$wl_{j,k}$ をコア j からメモリ k に書き込んだときの遅延時間とする．

$$T_{i,k} = \sum_j (rf_{i,j} * rl_{j,k} + wf_{i,j} * wl_{j,k}) \quad (3)$$

(4) $T_{i,m} = \min_k T_{i,k}$ となるようなメモリ m に共有データ i を配置する．

5.3　共有データに関する排他制御機構の決定

共有データを複数の公開関数がアクセスする場合，データの一貫性・安定性を保つために排他制御を行う必要がある．排他制御はアクセスする公開関数の関係性に応じて可能な限り実行オーバヘッドが小さいものを使用する．共有データをアクセスする公開関数の関係性は (1) アクセスする公開関数が同一のタスクのみに呼び出され，同一コアに割り付けられている場合，(2) アクセスする公開関数が他のタスクに呼び出され，同一コアに割り付けられている場合，(3) アクセスする公開関数が他のコアに割り付けられている場合の 3 通り存在する．(1) の場合は，その共有データにアクセスする処理の実行中に他のその共有データにアクセスする処理が実行されないので排他制御は不要である．(2) の場合は，その共有データにアクセスする処理の実行中に割込みによって他のその共有データにアクセスする処理を実行するタスクが起動する可能性があるので割り込み禁止を行う．(3) の場合は，スピンロックを用いてコアがその共有データにアクセスしている間は他のコアからアクセスできないようにする．

5.4　各コアの CPU 利用率の計算

コア j の CPU 利用率 u_j の計算式を以下に示す．ここで F_j をコア j に配置されている公開関数の集合とし，C_f を公開関数 f の最大実行時間とする．

$$u_j = \sum_{f \in F_j} \left(\frac{C_f}{Pe_f}\right) \quad (4)$$

ここで以下の条件式を満たさない場合については，スケジュール不可能となる．

$$\forall j, u_j < 1 \quad (5)$$

5.5　タスク i の余裕時間の計算

タスクの余裕時間を計算するために，そのタスクの最大応答時間を計算する必要がある．本研究ではタスクの実行時間を公開関数の実行時間，データアクセス時間，排他オーバーヘッド時間の合計値とする．そのため，T_i をタスク i の周期とすると，タスク i の最悪応答時間は以下の 4 つの値の合計値となる．

(1)　時刻 0 から時刻 T_i までのそのタスク以上の優先度を持つタスクに含まれる公開関数がアクセスするデータのアクセス時間の合計 (S_i^{ma})

(2) 時刻 0 から時刻 T_i までのそのタスク以上の優先度を持つタスクに含まれる公開関数がアクセスするデータに関する排他オーバーヘッドの合計 (S_i^{ex})

(3) 時刻 0 から時刻 T_i までのそのタスクより高い優先度を持つタスクの MIF の総和 (S_i^{MIF})

(4) そのタスク自身の最悪実行時間 (C_i)

ここで，$T_i - (S_i^{ma} + S_i^{ex} + S_i^{MIF} + C_i)$ がタスク i の余裕時間となる．

5.5.1 メモリアクセス時間の計算 (S_i^{ma})

$rF_{d,i}$ をタスク i と同じコアに含まれる公開関数のうち，共有データ d を読み込み，その公開関数を含むタスクがタスク i 以上の優先度をもつものの集合とする．そして $wF_{d,i}$ をタスク i と同じコアに含まれる公開関数のうち，共有データ d に書き込み，その公開関数を含むタスクがタスク i 以上の優先度をもつものの集合とする．ここで，$rA_{d,i}$ を共有データ d を読み込む公開関数の実行回数，$wA_{d,i}$ を共有データ d に書き込む公開関数の実行回数とすると，

$$rA_{d,i} = \sum_{f \in rF_{d,i}} \frac{T_i}{Pe_f}, wA_{d,i} = \sum_{f \in rF_{d,i}} \frac{T_i}{Pe_f} \quad (6)$$

となる．$rl_{d,i}$ をタスク i が割当てられているコアから共有データ d が配置されているメモリを読み込んだときの遅延時間，$wl_{d,i}$ をタスク i が割当てられているコアから共有データ d が配置されているメモリに書き込んだときの遅延時間とすると，S_i^{ma} は以下によって計算される．

$$S_i^{ma} = \sum_d (rA_{d,i} * rl_{d,i} + wA_{d,i} * wl_{d,i}) \quad (7)$$

5.5.2 排他オーバーヘッドの計算 (S_i^{ex})

共有データ d に対する排他オーバーヘッドを Ex_d とすると，S_i^{ex} は以下によって計算される．

$$S_i^{ex} = \sum_d \{(rA_{d,i} + wA_{d,i}) * Ex_d\} \quad (8)$$

5.5.3 MIF の総和 (S_i^{MIF})

タスク i を MF タスクとすると，k 番目のフレームの最大実行時間 C_i^k は以下の式で与えられる．

$$C_i^k = \sum_{f \in F_i} \{C_f * a_f(k)\} \quad (9)$$

ただし F_i をタスク i から呼び出される公開関数の集合，C_f を公開関数 f の最大実行時間とする．また，$a_f(k)$ は公開関数 f がフレーム k で実行されるなら 1，そうでないなら 0 を与える関数である．タスク i の MIF $M_i(t)$ は 3 章で示した方法によって MF タスクから導かれる．本手法では t をタスク i のデッドラインとして MIF を計算する．これにより S_i^{MIF} は以下の式で計算される．

$$S_i^{MIF} = \sum_{\forall j \in hp(i)} M_j(D_i) \quad (10)$$

ここで，$hp(i)$ はタスク i よりも高優先度なタスクの集合，D_i をタスク i のデッドラインとする．

5.5.4 最悪実行時間 (C_i)

最後に，タスクの最悪実行時間は以下の式で決定される．

$$C_i = \max_k(C_i^k) \quad (11)$$

5.6 最悪余裕時間の出力

$\min_i\{T_i - (S_i^{ma} + S_i^{ex} + S_i^{MIF} + C_i)\}$ がその割当てパターンにおける TC タスクの余裕時間である．

6. 評価

本章では，本研究の車載マルチコアシステムの見積もり手法に関して，実機実行との比較評価を行う．性能評価に用いるメモリへの読み込み，書き込み時間および排他オーバーヘッドは実機環境で測定したものを使用する．本章では以下の 3 つの評価を行う．

（評価 1） 見積もり時間の評価

（評価 2） 最悪余裕時間の比較評価

（評価 3） 各種オーバーヘッドの考慮・非考慮による比較評価

評価 1 では見積もりツールによる性能見積もりによって実際に要した時間を計測する．この評価により，現実的な時間で車載マルチコアシステムの性能見積もりが可能であることを示す．評価 2 では，実機実測での最悪余裕時間を計測し，見積もり値との比較を行う．実機実測で着目するタスクは，見積もり時に TC タスクとしたタスクとする．評価 3 では，実機実測による最悪余裕時間と，各種オーバーヘッドの考慮・非考慮による最悪余裕時間の見積もり結果を比較する．具体的には，排他オーバーヘッド，メモリアクセス時間を考慮・非考慮した時の最悪余裕時間を計測する．このような評価を行うのは実機と近い見積もり値を出すためにはどの程度実機と条件を近くする必要があるかを確認するためである．一般的には実機と条件が近いほど見積もり値は実機と近くなるが，その分見積もりにかかる時間が増加する．マルチコアシステムでは多数のコア割当てパターンに対して見積もりを行う必要が出てくるため，実機の実行時間と近似している範囲で短い時間で見積もりを行うことが求められる．

6.1 評価環境

本研究の評価環境としては，実機環境として車載デュアルコアマイコンを用いた．実機への実装には我々が開発している車載ソフトのマルチコア実装支援ツールである PMPF[4] を用いた．見積もりについても図 3 のデュアルコア環境を想定している．

6.2 評価モデル

本評価で使用するソフトウェアモデルはパワトレアプリを想定した評価ソフトウェアモデルであり，30 個程度の周期タスクで構成される．タスクは 7 つの PF グループに分けられ，PF グループ単位でコアに配置される．7 つ PF グループをデュアルコアに配置するパターン数は 63 通りである．

Vol.2017-ARC-225 No.11
Vol.2017-SLDM-179 No.11
Vol.2017-EMB-44 No.11
2017/3/9

6.3 （評価 1）見積もり時間の評価

6.2 節のモデルに対し，2,4 コアに PF グループを配置する場合について，性能見積もりに要した時間を計測する．2,4 コアを想定して，すべてのパターンがスケジュール可能であった場合での見積もり時間も計測する．結果を表 1 に示す．2,4 コアのどちらの場合でも現実的な時間で性能見積もりが可能であることがわかった．コア数の増加による見積もり時間への影響は大きいものの，コア数と比べて見つかったスケジュール可能パターン数が性能見積もり時間に与える影響は大きなものではなかった．

表 1 評価 1 の結果

コア数	全配置パターン数	スケジュール可能パターン数	見積もり時間 [s]
2	63	6	24
		63	46
4	350	263	202
		350	239

6.4 （評価 2）最悪余裕時間の比較評価

6.2 節のモデルについて，5 章で示した手法を用いた性能見積もりおよび実機による最悪余裕時間の比較評価を行った．実機実行による評価は性能見積もり時に出力された TC タスクについて 10s 間実行を行い，応答時間を測定し，その中で最大であったときの余裕時間を用いた．

結果を表 2 に示す．性能見積もりによってスケジュール可能となったパターンは 6 通りであり，全体の約 10 分 1 であった．また，それぞれの割当てパターンについて，実機と見積もり値の比較を見てみると，すべてのパターンにおいて見積もり値が実機の値を下回っており，見積もりによってスケジュール可能であると判定された割当てパターンが安全であることがわかる．しかし実機と見積もり値で最悪余裕時間に大きな差があり，本来スケジュール可能であったパターンについても除外されている可能性がある．

表 2 評価 2 の見積もり結果

配置パターン	実機での最悪余裕時間 [μ s]	見積もりでの最悪余裕時間 [μ s]
1	308	95
2	156	94
3	267	56
4	148	56
5	118	11
6	137	2

6.5 （評価 3）各種オーバーヘッドの考慮・非考慮による比較評価

6.2 節のモデルについて，排他オーバーヘッドおよびメモリアクセス時間の考慮・非考慮時の性能見積もりを行い，それぞれ最悪余裕時間が最も短いパターンについて実機実行による結果との比較評価を行った．結果を表 3 に示す．

排他オーバーヘッドを非考慮にした場合でも，排他オーバーヘッドを考慮している場合と大きな変化は見られなかった．しかし，メモリアクセス時間を考慮しない場合では性能見積もりではスケジュール可能と判定されたパターンであっても実機実行ではタスクのデッドラインミスが発生し，実機実行による TC タスクの余裕時間が見積もりより短くなる可能性があった．以上より，排他オーバーヘッドを不考慮としても見積もりに大きな影響はないが，メモリアクセス時間は影響が大きく，メモリアクセス時間は性能見積もりに不可欠な要素であることがわかった．

表 3 評価 3 の結果

排他オーバーヘッド	メモリアクセス時間	10s 間のデッドラインミス数	デッドラインミス確率 [%]	実機の最大応答時間 [μ s]	見積もりの最大応答時間
○	○	0	0	137	2
×	○	0	0	145	2
○	×	198	2.47	-	4

7. おわりに

本研究ではマルチコアで実装されたパワトレアプリの評価ソフトに対する性能見積もりの手法について述べた．PF グループを用いることでツールによる全数探索が可能となるまで公開関数のコア配置パターンを削減し，全配置パターンについて最悪余裕時間を計算することで公開関数のコア配置の決定を支援するツールを実現した．今後の課題としては，排他競合時間による影響についても見積もりを行うことができるようにすることや実機結果と見積もり値の差が縮まるように本手法を改良することが上げられる．

謝辞 本研究を進めるにあたりご協力いただいたトヨタ自動車株式会社に深く御礼申し上げます．

参考文献

[1] L. Michel, T. Flaemig, D. Claraz, R. Mader, ”Shared SW development in multi-core automotive context”, in ERTS2-2016, Toulouse, Jan.2016.

[2] Mok, Aloysius K., and Deji Chen. ”A multiframe model for real-time tasks.” IEEE transactions on Software Engineering 23.10 (1997): 635-645.

[3] 飯山真一, 高田広章, 菅沼英明,「エンジン制御システムのためのリアルタイム性検証手法」, 情報処理学会論文誌, vol.43, no.6, pp.1715-1724，Jun. 2002.

[4] 小川真彩高，本田晋也，高田広章， 車載制御向けマルチコアプログラミングフレームワーク, 研究報告組込みシステム（EMB） , Vol.2015-EMB-39,No.9, pp. 1 - 6 , お茶の水女子大学, Nov 2015.