Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

タスクマイグレーション手法のエンジン制御ソフトウェアへの適用

相庭裕史[†1]　本田晋也[†2]　高田広章[†1]

近年，組込みシステムの分野においてもマルチコアシステムの利用が進んでいる．マルチコアシステムは，大きく対称型と機能分散型に分けられるが，組込みシステムにおいては，リアルタイム性を確保しやすい機能分散型が用いられるのが一般的である．しかしながら，対称型は汎用的なハードウェアが使用できることや，対称型の RTOS が提供するタスクマイグレーション機能が使用できるといった利点がある．そこで，本研究では，エンジン制御ソフトウェアに対して，対称型マルチコアシステムを適用する．具体的には，ソフトリアルタイムタスクのデッドラインミス軽減を目的にタスクマイグレーションの利用を検討した．また，タスクマイグレーションにより，コアに静的に割り当てているハードリアルタイムタスクに与える影響について評価を行った．その結果，ハードリアルタイムタスクの実行時間に影響を及ぼす要因として，タスクマイグレーションを実現するオーバヘッドより，ソフトリアルタイムタスクがマイグレーションすることによるキャッシュミス増加の影響が大きいことが分かった．

Application of Task Migration to Engine Management Software

Hiroshi Aiba,[†1] Shinya Honda[†2] and Hiroaki Takada[†1]

In recent years, multi-core processor systems have been used even in embedded systems. Multi-core processor systems are broadly classified as Function Distributed Multiprocessor (FDMP) systems and Symmetric Multiprocessor (SMP) systems. Generally, FDMP is used in embedded systems because it is easier to satisfy real-time property than SMP. SMP, however, has the advantage that versatile hardware is available and a task migration can be used. Therefore, we apply SMP to an engine management software. Specifically, we use the task migration to reduce deadline misses of soft real-time tasks and evaluate effect on hard real-time tasks. Experimental results show that the increase in cache misses due to the task migration has more impact on the execution time of hard real-time tasks than the overhead of the task migration.

1. はじめに

近年，組込みシステムの分野においても，マルチコアシステムの重要性が急速に増している．この背景には，消費電力の増大を抑えつつ，処理性能の向上をはかるためには，クロック周波数を上げるよりも，コア数を増やした方が有利であるという状況がある．

マルチコアシステムは，対称型と機能分散型に分類することができる．対称型マルチコア（Symmetric Multiprocessor，SMP）システムとは，各コアから計算機のすべての資源にアクセスすることができ，（機能的には）どの処理（タスク）をどのコアでも実行できるものをいう．タスクは，負荷分散のために，OS により動的にコアに割り当てられる．それに対して機能分散型マルチコア（Function Distributed Multiprocessor，FDMP）[⋆1]システムとは，各コアからアクセスできる資源に違いがあり，どの処理をどのコアで行うかが，あらかじめ決まっているようなシステムをいう．

組込みシステムは，ある応用に専用化された計算機システムであり，どのような処理を行う必要があるかは，あらかじめ決まっているのが通常である．そのため，機能分散型マルチコアシステムを採用した方が，スケーラビリティの確保，リアルタイム性の確保の観点から，有利であると言える．しかしながら，コストの観点では対称型マルチコアシステムに対して不利な場合が多い．これは，対称型マルチコア用のハードウェア（以下，対称型ハードウェア）は汎用性が高いのに対して，機能分散型マルチコアシステム用のハードウェア（以下，機能分散型ハードウェア）は，システム毎に専用に用意する必要があることや，タスクを静的にコアに割り当てるため，動的に割り当てた場合と比較してコアの利用効率が低く，同じ性能を得ようとすると，高いコア性能が必要となるためである．

そのため，車載制御システムのようなコストを重視するシステムにおいては，対称型ハードウェアやタスクマイグレーションの利用によるコスト削減が期待されている．そこで，これまでに我々は，TOPPERS/FMP カーネル（以下，FMP カーネル）を開発した[9]．FMP カーネルは，OS によるタスクの自動的なコアへの割り当ては行わず，ユーザーからの要求

†1 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University
†2 名古屋大学大学院情報科学研究科付属組込みシステム研究センター
Center for Embedded Computing Systems,Nagoya University
⋆1 非対称型マルチコア（Asymmetric Multiprocessor，AMP）と呼ばれる場合もある．

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

時にのみタスクマイグレーションを行うことで，リアルタイム性とタスクマイグレーションの両立を目指している．しかしながら，これまで適用事例がなく，タスクマイグレーション手法や，ハードリアルタイムタスクに与える影響が不明である．また，FMP カーネルのタスクマイグレーション機能が十分であるかの評価もなされていない．

そこで本研究では，エンジン制御ソフトウェアを例に，FMP カーネルと対称型ハードウェアのハードリアルタイムシステムへの適用性を評価する．具体的には，ハードリアルタイムシステムにおいてタスクマイグレーションが有効な状況と，その状況において有効なマイグレーション手法について検討する．次に，検討したマイグレーション手法をエンジン制御ソフトウェアに適用して効果と影響を評価する．

本論文の構成は，まず 2 章で FMP カーネルの特徴について述べ，3 章で評価対象としたエンジン制御ソフトウェアについて述べる．4 章では，エンジン制御ソフトウェアに有効なマイグレーション手法について検討し，5 章で検討した手法の評価を行う．6 章で本研究のまとめを行う．

2. FMP カーネル

FMP カーネルは，前述の対称型 OS の問題を解決し，リアルタイム性とロードバランスの実現の両立を目指したマルチコア向け RTOS である．

FMP カーネルでは，OS が動的にロードバランスを行わず，ユーザーからの要求時（API による要求）にのみタスクマイグレーションを行うことで，リアルタイム性を高めている．また，OS の内部構造を機能分散型 OS と同等とすることで，コア数に対する性能のスケーラビリティを確保している．リアルタイム性を確保しやすいという機能分散型 OS の利点を活かすために，FMP カーネルでは，**図 1** に示すように，コアごとにレディキューを持っている．各コアのレディキューは個別に管理するので，各コアが自コアに閉じた処理を実行している限りは，他のコアの影響を受けることはない．スケジューリングは，プリエンプティブな優先度ベースをコア毎に行うため，シングルコアと同等であり，リアルタイム性を保証しやすい．FMP カーネルを用いた研究成果として，文献 8) では，FMP カーネルを用いたロードバランス機構の提案がなされている．文献 8) では，ソフトリアルタイムシステムを対象としているが，本研究では，ハードリアルタイムシステムを対象とする．

FMP カーネルでは，以下の 2 種類の API によるタスクマイグレーションをサポートする．

- mig_tsk(ID tskid,ID prcid)
  tskid で指定したタスクを prcid で指定したコアにマイグレーションする．この API は，

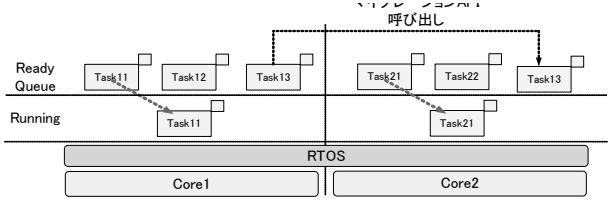

図 1 TOPPERS/FMP カーネル
Fig. 1 TOPPERS/FMP Kernel

API を発行したコアに割り当てられたタスクに対してのみ発行可能である．また，非タスクコンテキストからは本 API を発行することはできない．これらは，システムコールの最悪実行時間を抑えるための制約である．

- mact_tsk(ID tskid,ID prcid)
  tskid で指定したタスクを prcid で指定したコアで起動する．

3. エンジン制御ソフトウエア

自動車のエンジン制御は，エンジン回転数やアクセル開度などのセンサから得られる入力信号を処理し，燃料の噴射量，噴射時間や点火タイミングなどを決定し，各アクチュエータを駆動することで，実現される．これらの処理は，エンジン制御 ECU（Electronic Control Unit）と呼ばれるコンピュータを用いて実現される．

本章では，エンジン制御システムのソフトウェア構成及び，評価で用いたエンジン制御 ECU ベンチマークプログラムについて説明する．

3.1 ソフトウェア構成

エンジン制御ソフトウェアを構成するタスクは，回転角同期タスクと時間同期タスクに分類される．回転角同期タスクは，クランクの回転角度に同期して起動されるタスクの集合である．これらの処理は，静的優先度割付された優先度ベーススケジューリングを行う RTOS により，プリエンプティブなマルチタスク環境下で実行される．

回転角同期タスクはエンジン回転数に応じて，起動周期が変わるため，実行時に負荷が変動する．回転角同期タスクには燃料の噴射タイミングを決定する処理などが含まれる．時間同期タスクは，タスク毎の固定周期で起動されるタスクの集合である．時間同期タスクには，温度センサの信号から，各種補正パラメータを計算する処理などが含まれる．

タスクの多くは，デッドラインミスの許されないハードリアルタイムタスクである．しか

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

しながら，補正計算などの処理の中には，前回の計算結果で代用できるものもあるため，ある程度のデッドラインミスが許容されるソフトリアルタイムタスクも含まれる．自動車のコスト制約は厳しく，容易にハードウェアの性能を上げることはできないため，システム高負荷時にソフトリアルタイムタスクのデッドラインミスは避けられない．ソフトリアルタイムタスクのデッドラインミスは，制御精度の低下につながるものの，安全性には問題はない．

3.2 エンジン制御 ECU ベンチマークプログラム

エンジン制御ソフトウェアを実行するには，実際のエンジン，もしくは Hardware in the Loop Simulation（HILS）といった，高価な環境が必要となる．そこで，これらの環境がなくても，エンジン制御 ECU の評価が可能なベンチマークプログラムが用意されている．

ベンチマークプログラムは，一定のエンジン回転数で一定時間走行した場合のエンジン制御ソフトウェアの振る舞いを模擬している．外部割込みを再現するために，回転角同期割込スケジューラ，時間同期割込スケジューラと呼ばれる 2 つの周期タスクが存在する．これらのタスクは，最高優先度で動作し，起動すると，外部割込みを模擬した関数（擬似割込み関数）を呼び出す．エンジン制御ソフトウェアを構成するタスクは基本的に，擬似割込み関数から起動され，処理を行うと終了する．そのため，既に起動状態のタスクに対して起動を行った場合がデッドラインミスとなる．

ベンチマークプログラム内のシミュレーション時間は，時間同期割込スケジューラの起動周期より決定され，実時間とは必ずしも一致しない．本研究では回転角同期割込スケジューラと時間同期割込スケジューラの起動周期が等しいときを，エンジン回転数 5000rpm と定め，時間同期割込スケジューラの起動周期を一定にしたまま，回転角同期割込スケジューラの起動周期を変更することで，異なるエンジン回転数を再現した．

4. エンジン制御ソフトウェア向けのマイグレーション手法

本章では，エンジン制御ソフトウェアに有効なタスクマイグレーション手法について検討する．なお，評価環境のコア数は 4 コアである．評価環境の詳細については，5 章で述べる．

4.1 方　　針

タスクマイグレーションのオーバヘッドはタスクの予測性を低下させる．そこで，文献 1) では，予測性を高めるためのマイグレーション手法が提案されている．また，文献 2) では，マイグレーションのコストを計算し，全タスクが時間制約を満たせる場合のみ，マイグレーションを行うことで，リアルタイム性を確保している．文献 1)，2) は，全てハードリアルタイムタスクによって構成されているシステムを対象としている．

しかしながら，エンジン制御ソフトウェアなど，実際のハードリアルタイムシステムで用いられるソフトウェアの多くは，ソフトリアルタイムタスクを含んでいる場合が多い．ハードリアルタイムタスクとソフトリアルタイムタスクが混在するシステムの場合，ハードリアルタイムタスクはコアに固定し，ソフトリアルタイムタスクのみをマイグレーション対象とすることで，ハードリアルタイムタスクへの高い予測性の要求に対応しつつ，負荷変動への対応が可能である．文献 6)，7) ではソフトリアルタイムタスクのスループット向上を目的とし，優先度の高いタスクはコアに固定し，優先度の低いタスクは動的に各コアに割当てる機構を提案している．本研究でも同様に，ハードリアルタイムタスクはコアに固定し，マイグレーションは行わないこととした．3.1 で述べたとおり，自動車のコスト制約は厳しいため，マルチコアプロセッサを用いた場合でもソフトリアルタイムタスクのデッドラインミスが避けられないことが予想される．そこで本研究では，静的割当てを行った結果，高負荷時にソフトリアルタイムタスクにデッドラインミスが発生している状況を想定し，それを軽減するため，ソフトリアルタイムタスクに対してマイグレーションを行う．実行時に負荷の低いコアにソフトリアルタイムタスクをマイグレーションすることで，応答時間が短縮されデッドラインミスの軽減が期待できる．

4.2 問 題 設 定

本章では，対象ソフトウェア上での問題設定に述べる．具体的には，4.2.1 で，マイグレーション対象とするソフトリアルタイムタスクについて述べる．4.2.2 では，静的割当ての結果，マイグレーション対象とするソフトリアルタイムタスクのデッドラインミスが，避けられない状況の設定について述べる．

本章で設定した状況において，ソフトリアルタイムタスクのデッドラインミス率を下げるためのマイグレーション手法を検討する．

4.2.1 対象タスク

マイグレーション対象とするソフトリアルタイムタスクは，簡単化のために 1 つとした．このタスクを最低優先度時間同期タスクと呼ぶ．このタスクは，エンジン制御 ECU ベンチマークプログラムに存在している最低優先度で動作する時間同期のソフトリアルタイムタスクであり，他のタスクと比較して処理時間が長い．そのため，静的割当ての場合，このタスクがデッドラインミスを起こさないようにプロセッサを選択すると，全てのタスクがデッドラインを満たせる．言い換えると，このタスクの存在により，要求されるプロセッサ能力が上がってしまう．そのため，マイグレーション対象のタスクとして適切と考えた．なお，

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

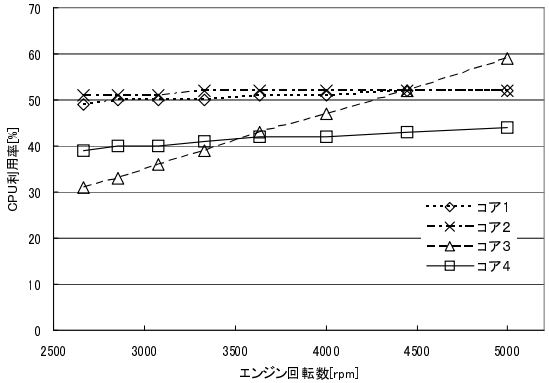

図 2 最低優先度時間同期タスクを除いた CPU 利用率
Fig. 2 CPU utilization wituout lowest priority time synchronization task

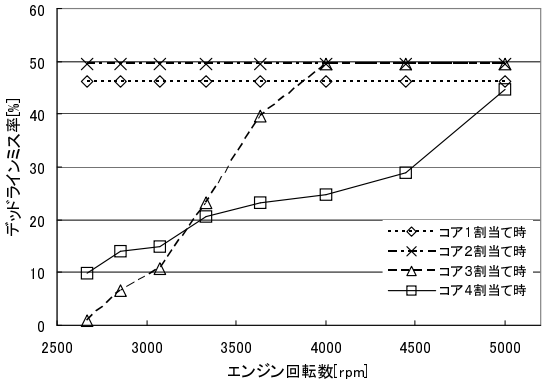

図 3 静的割当て時の最低優先度時間同期タスクのデッドラインミス率
Fig. 3 Deadline miss ratio of lowest priority time synchronization task in static allocation

これ以降，最低優先度時間同期タスク以外のタスクを便宜上[⋆1]，ハードリアルタイムタスクと呼ぶ．

4.2.2 静的割当てと時間設定

エンジン制御 ECU ベンチマークプログラムでは，実時間とシミュレーション時間との関係を時間同期割込スケジューラの起動周期を変更することにより調整可能である．そのため，まず，ハードリアルタイムタスクのみを対象とするとデッドラインミスが発生せず，最低優先度時間同期タスクも対象とするとデッドラインミスが発生するような，時間同期割込スケジューラの起動周期と静的なタスクのコア割り当てを決定する．

これまでに，マルチコアに対するタスク割当アルゴリズムに関する研究がなされている[3)–5)]．タスク割当てアルゴリズムの多くはタスクへの優先度を自由に設定できることが前提である．しかしながら，今回対象としたエンジン制御 ECU ベンチマークプログラムは，既に優先度が設定されており，変更するこが出来ないため，アルゴリズムの適用は行わず，手作業で様々なタスク割当てを試して決定した．

図 2 は静的割当て後のハードリアルタイムタスクのコア毎の CPU 利用率とエンジン回転数の関係を示している．この状態では，ハードリアルタイムタスクにはデッドラインミスは発生していない．コア 3 には，処理量の多い回転角同期タスクが割り当てられているため，回転数の上昇と共に，CPU 利用率が上昇している．

この静的割当てにおいて，最低優先度時間同期タスクをそれぞれのコアに割り当てた場合のデッドラインミス率を，**図 3** に示す．前述のように，コア 3 には，回転角同期タスクが割り当てられているため，回転数が上昇すると共に，デッドラインミス率も上昇する．

4.3 動的割当て手法

本手法は，**図 4** のように，実行可能状態のハードリアルタイムタスクがなくなった場合に，最低優先度時間同期タスクの状態をチェックし，どのコアでも実行されていなければ，自コアにマイグレーションして実行する．

一方，最低優先度時間同期タスクが実行されているコアで，ハードリアルタイムタスクが起動した場合には，最低優先度時間同期タスクはプリエンプトされる．そして，いずれかのコアがアイドル状態になると，そのコアにマイグレーションされて実行される．

本手法では，カーネルオブジェクトとして，データキュー 1 個と，各コアにアイドルタスクを用意する．最低優先度時間同期タスクを起動する処理は書き換え，タスク ID を登録するようにする．アイドルタスクはこのデータキューを調べることで，最低優先度時間同期タスクの状態をチェックする．

ハードリアルタイムタスクは起動時に最低優先度時間同期タスクをプリエンプトしたかど

⋆1 実際には最低優先度時間同期タスク以外のソフトリアルタイムタスクも含む

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

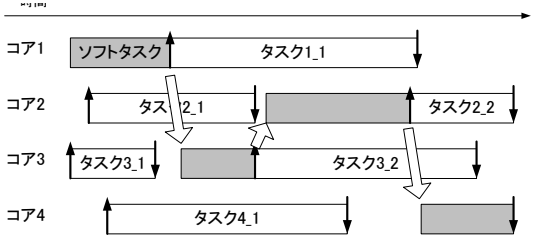

図 4 動的割当て手法概要
Fig. 4 Detail of dynamic assign method

うか判断し，プリエンプトした場合には，タスク ID をデータキューに戻す．本手法がハードリアルタイムタスクに直接的に影響を与えるのは，この処理のオーバヘッドである．5 章では，この評価も行う．

4.4 モード切替え手法

動的割当て手法は，汎用的な手法であるが，ハードリアルタイムタスク起動時に最低優先度時間同期タスクのプリエンプションを毎回チェックする必要があることや，タスクマイグレーションが頻発することにより，マイグレーションのオーバヘッドやキャッシュの効率が悪化することが予想される．そこで，アプリケーションの特性を考慮したマイグレーション手法として，モード切替え手法を検討した．

マルチコアシステムでは，図 2 で示したように，エンジン回転数によってコア毎の負荷に偏りが生じる場合がある．本手法では，エンジン回転数に応じて，最低優先度時間同期タスクのデッドラインミス率が低くなるコアを予め求めておき，実行時にエンジン回転数を判断して，最低優先度時間同期タスクをどのコアで実行するか決定する．

最低優先度時間同期タスクをそれぞれのコアに静的に割り当てた上で，エンジン回転数を変化させた場合のデッドラインミス率を計測する．そして，各回転数で最もデッドラインミス率が低くなるコアを求め，その回転数で最低優先度時間同期タスクを割り当てるコアとする．

図 3 から，約 3250rpm 以上の場合は最低優先度時間同期タスクをコア 4 に割り当て，それ以下の場合はコア 3 に割り当てることとした．

タスクの移動は，起動時に FMP カーネルのタスクマイグレーション API の一つである，mact_tsk を用いて最適なコアで起動させることで行う．タスクマイグレーションは，起動するコアを切替えることで行うことから，既にソフトリアルタイムタスクが起動されている状態で，マイグレーションすべきエンジン回転数に変わっても，そのタイミングではマイグレーションは行われない．しかし本手法は，あるエンジン回転数で一定時間実行した場合に，合計のデッドラインミス回数を少なくすることを目的としているため，マイグレーションタイミングがわずかに遅れても，その影響は小さいと考えられる．

表 1 NaviEngine の仕様
Table 1 Specification of NaviEngine

CPU コア	ARM11 MPCore(ARM11 × 4 個)
命令キャッシュ	コア毎に 32Kbyte
データキャッシュ	コア毎に 32Kbyte
最大動作周波数	399MHz
内部バス	AXI バス（バス幅 64bit，133MHz）

5. 評価実験

4 章で述べた 2 種類のマイグレーション手法をエンジン制御 ECU 用ベンチマークプログラムに適用し，デッドラインミス率軽減の効果と，ハードリアルタイムタスクへの影響の評価を行った．

5.1 評価環境

評価環境として，NEC エレクトロニクス社製 LSI「NaviEngine」を搭載した評価ボードを用いた．ボードの仕様を**表 1** に示す．プロセッサは，ARM11 コアを 4 個搭載した「ARM11 MPCore」が，メモリは，DDR2 SDRAM（メモリ容量：256Mbyte，サイクルタイム:DDR2-667，バス幅 32bit）が搭載されている．ARM11 MPCore は典型的な SMP 型ハードウェアである．それぞれのコアのデータキャッシュは，ハードウェアによってコヒーレントが保たれる．

プログラムの実行時間等の測定は，ARM11 MPCore が持つパフォーマンスモニタ機能を用いて行った．

5.2 評価項目

エンジン制御 ECU 用ベンチマークプログラムを，最低優先度時間同期タスク以外を静的にコアに割り当てた上で，最低優先度時間同期タスクを，コア 4 に固定した場合，モード切替え手法を適用した場合，動的割当て手法を適用した場合について，各種項目を測定した．比較対象として最低優先度時間同期タスクをコア 4 に固定した場合を選択した理由は，図 3 より，コア 4 に固定した場合が最もデッドラインミス率が低いためである．

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

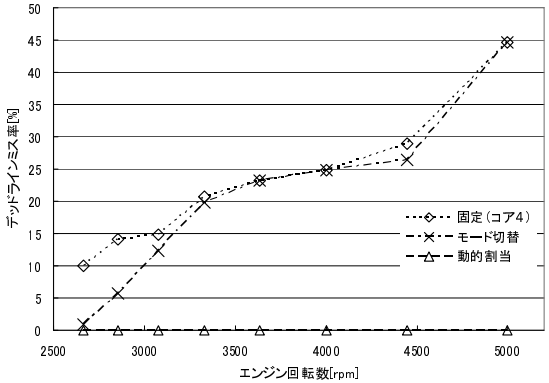

図 5 最低優先度時間同期タスクのデッドラインミス率
Fig. 5 Deadline miss ratio of lowest priority time synchronization Task

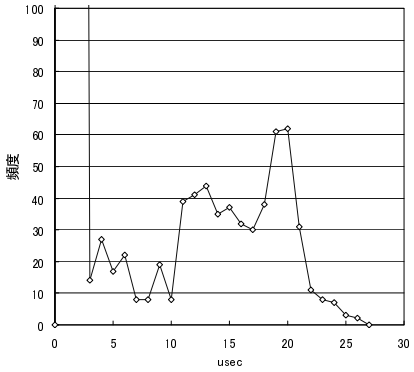

図 6 動的割当て手法の実行オーバヘッド
Fig. 6 Execution overhead in dynamic assign method

また，それぞれの条件で，プログラムの実時間は 400msec とし，エンジンの回転数は，5000rpm，4444rpm，4000rpm，3636rpm，3333rpm，3076rpm，2857rpm，2667rpm として測定した．

評価項目は次の通りである．デッドラインミス率とは，デッドラインミス回数を起動命令発行回数で割った値とする．

(1) 最低優先度時間同期タスクのデッドラインミス率
(2) 動的割当て手法の実行オーバヘッド
(3) ハードリアルタイムタスクのコア毎の総実行時間
(4) ハードリアルタイムタスクのコア毎のキャッシュミス回数

5.3 評価結果

最低優先度時間同期タスクのデッドラインミス率を**図 5** に示す．モード切替え手法では，エンジン回転数が 3250rpm 以下になった場合に，ソフトリアルタイムタスクをコア 4 から，3250rpm 以下で最も負荷が低くなるコア 3 にマイグレーションする．そのため，高回転域では，コア 4 に固定した場合と差は無いが，低回転域では，コア 4 に固定した場合よりもデッドラインミス率が下がる．

動的割当て手法適用時は想定する最大回転数である 5000rpm 時においてもデッドラインミス率は 0% となった．ただし，5000rpm 時には，コア 3 に静的に割当てた他のタスクに約 6% デッドラインミスが発生した．この原因として，動的割当てのためにハードリアルタイムタスクで発生する実行オーバヘッドが考えられる．

動的割当て手法の実行オーバヘッドの 5000rpm 時における時間分布を**図 6** に示す．なお，2usec 未満の場合の頻度は 8,460 回であり，オーバヘッドの最大値は 25.3usec であった．ハードリアルタイムタスクの起動は全部で 9,064 回あり，そのとき，最低優先度時間同期タスクをプリエンプトしていたのは 512 回であった．この結果から，ハードリアルタイムタスク起動時の最低優先度時間同期タスクのプリエンプションは多くの場合に発生しないことが分かる．この場合は，プリエンプションのチェックのために，2usec 程度の実行オーバヘッド発生するのみである．一方，最低優先度時間同期タスクのプリエンプションが発生した場合には，発生しなかった場合の 10 倍以上の実行オーバヘッドが発生するということが分かる．

図 7，**図 9** は，実時間で 400msec プログラムを実行した場合の，ハードリアルタイムタスクの総実行時間をコア毎に集計したものである．コア固定時と比較して，モード切替え手法と動的割当て手法のそれぞれで実行時間に変化がみられる．

その原因としては，動的割当て手法では，前述した起動時の実行オーバヘッドが考えられる．5000rpm 時の各コアでの実行オーバヘッドと，タスクの合計実行時間に対する割合を**表 2** に示す．タスクの全実行時間に対して，この処理に要した時間の割合は，数 % である

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

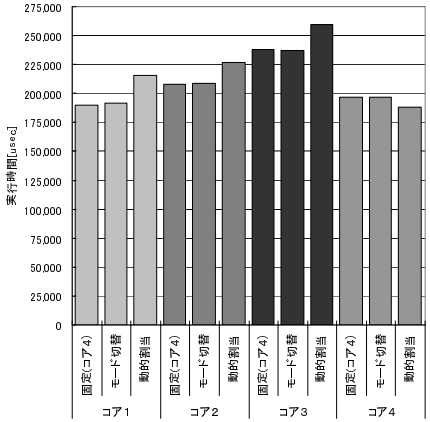

図 7 実行時間（5000rpm）
Fig. 7 Execution time （5000rpm）

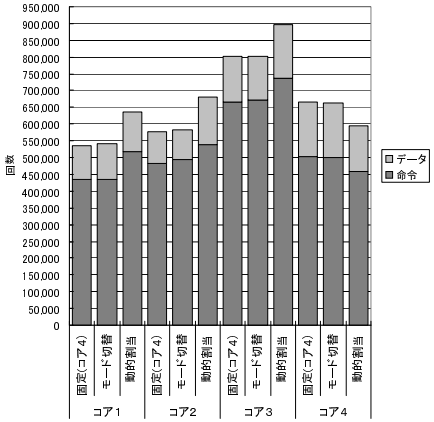

図 8 キャッシュミス回数（5000rpm）
Fig. 8 cache miss counts（5000rpm）

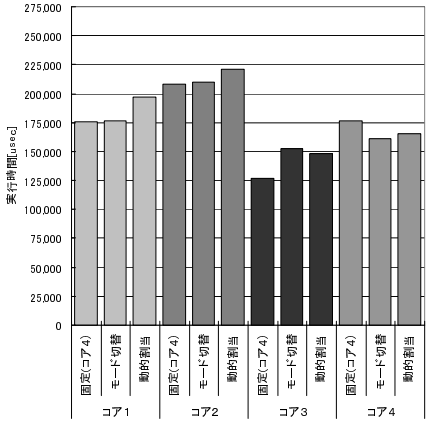

図 9 実行時間（2667rpm）
Fig. 9 Execution time（2667rpm）

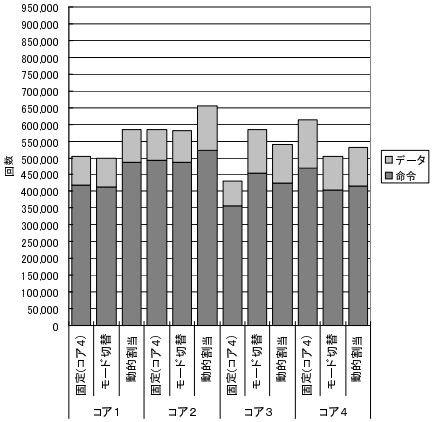

図 10 キャッシュミス回数（2667rpm）
Fig. 10 cache miss counts（2667rpm）

表 2 動的割当て手法の実行オーバヘッドの割合（5000rpm）
Table 2 Execution overhead ratio in dynamic assign method（5000rpm）

	コア 1	コア 2	コア 3	コア 4
タスク合計実行時間（usec）	215,474	227,047	259,716	188,439
実行オーバヘッド（usec）	5,955	3,004	4,076	1,192
実行オーバヘッドの割合（%）	2.764	1.323	1.569	0.633

ため，この処理時間だけでは，実行時間の増加を説明することはできない．

図 8，**図 10** は最低優先度時間同期タスク以外のタスク実行中に発生したキャッシュミス回数をコア毎に集計したものである．キャッシュミス回数と図 7，図 9 で示した実行時間との間には相関があることがわかる．評価環境では，キャッシュミス 1 回当たり，およそ 200nsec のペナルティが発生する．そこで，実行時間から，キャッシュペナルティ分を減算したものを，**図 11**，**図 12** に示す．図 7，図 9 と比較して，コア固定時とそれぞれの手法適用時の実行時間の差は小さくなっている．したがって，タスク実行時間増加の主たる要因はキャッシュミスによるものと言える．

キャッシュミス回数変化の原因は，最低優先度時間同期タスクが，マイグレーション先のコアのキャッシュを汚すためであると考えられる．モード切替え手法では 2667rpm 時には，コア 3 にソフトリアルタイムタスクが割当てられるため，コア 4 に固定した場合と比較して，コア 3 のキャッシュミス回数が増加している．逆にコア 4 は最低優先度時間同期タスクが無くなったため，キャッシュが汚されなくなり，キャッシュミス回数が減少したと考えられる．

5.4 考　　察

タスクマイグレーションを行うとキャッシュミス回数が変化し，コアに固定したタスクの実行時間が変動することが分かった．

動的割当て手法では，最低優先度時間同期タスクのマイグレーションのタイミングを設計時に予想することは困難であるため，キャッシュの振る舞いを予測することは難しい．また，ハードリアルタイムタスクが最低優先度時間同期タスクをプリエンプトした場合にはプリエンプトしなかった場合の 10 倍ほどのオーバヘッドが発生する．しかしながら，動的割当て手法はコアの利用効率が高く，最低優先度時間同期タスクのデッドラインミスを大きく軽減することが可能である．動的割当て手法は，ハードリアルタイムタスクの負荷が低いシステムでの利用や，ハードリアルタイムタスクの負荷が高いコアにはマイグレーションしないなどの改良を加えて実用化することが考えられる．

モード切替え手法についても，マイグレーションに伴うキャッシュ汚染の問題はあるが，

Vol.2010-SLDM-144 No.6
Vol.2010-EMB-16 No.6
Vol.2010-MBL-53 No.6
Vol.2010-UBI-25 No.6
2010/3/26

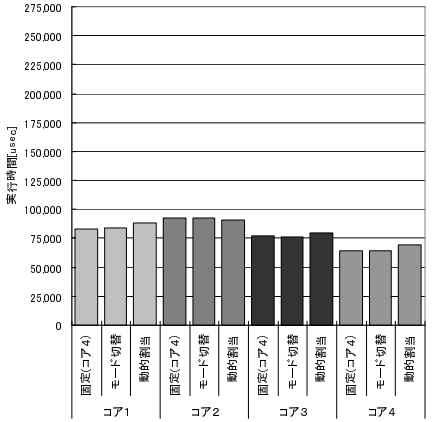

図 11 キャッシュペナルティを除去した実行時間（5000rpm 時）
Fig. 11 Execution time without cache penalties (5000rpm)

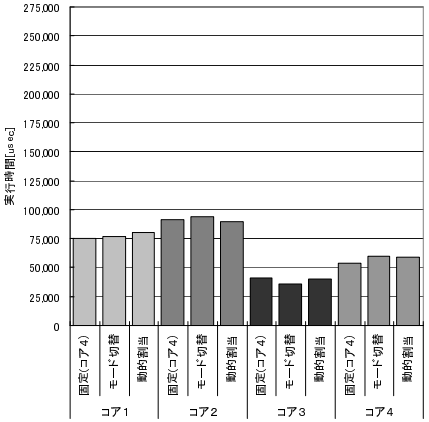

図 12 キャッシュペナルティを除去した実行時間（2667rpm 時）
Fig. 12 Execution time without cache penalties (2667rpm)

モード切替え手法では，設計時にマイグレーション先のコアを予め決定しておくことから，キャッシュの振る舞いを予測することは，動的割当て手法よりも容易であると考えられる．

6. おわりに

本研究では，エンジン制御ソフトウェアを対象に，ハードリアルタイムシステムにおいて有効なタスクマイグレーション手法を検討した．エンジン制御ソフトウェアは主にハードリアルタイムタスクによって構成されているが，ソフトリアルタイムタスクも含まれていることに着目し，ソフトリアルタイムタスクのデッドラインミス軽減にタスクマイグレーションを用いた．そして実験により，検討した手法の効果の確認と，ハードリアルタイムタスクへの影響を評価した．タスクマイグレーションの影響として，移動したタスクが移動先のコアのキャッシュを汚し，そのコアに割り当てられているタスクの実行時間に大きな影響を与えることが明らかになった．

今後の課題として，ソフトリアルタイムタスクが複数ある場合や，ソフトリアルタイムタスクに優先度が設定されている場合のタスクマイグレーション手法を検討することが挙げられる．

謝辞 本研究を進めるにあたりご協力頂いたトヨタ自動車株式会社，NEC エレクトロニクス株式会社に深く御礼申し上げます．

参 考 文 献

1) A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan：Pushassisted migration of real-time tasks in multi-core processors，In ACM SIGPLAN Conference on Language, Compiler, and Tool Support for Embedded Systems, pp 80-89, June 2009.
2) Kedar M.Katre，Harini Ramaprasad，Abhik Sarkar，and Frank Mueller：Policies for Migration of Real-Time Tasks in Embedded Multi-Core Systems, RTSS2009，pp17-20，Dec 2009，Washington,D.C.,USA
3) Yingfeng Oh，Sang H. Son：Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems, Real-Time Systems Volume 9，Number 3 1995 Springer pp.207-239
4) Davari, S. and S.K. Dhall：An On Line Algorithm for Real-Time Tasks Allocation, IEEE Real-Time Systems Symposium，pp 194-200 1986
5) Davari, S. and S.K. Dhall：On a Periodic Real-Time Task Allocation Problem, Proc. of 19th Annual International Conference on System Sciences，pp 133-141 1986
6) 深江輝昭，本田晋也，冨山宏之，高田広章：機能分散マルチプロセッサ向け RTOS へのマイグレーション可能タスクの導入, 電子情報通信学会技術研究報告（IEICE Technical Report），Vol.106,No.601, pp. 7-12, 広島, Mar 2007.
7) 蛸井基明, 宮内新, 石川知雄, 高田広章：マルチプロセッサ環境におけるマイグレート可能タスクの導入, 電子情報通信学会技術研究報告. CPSY, コンピュータシステム, Vol. 101, No.672, pp. 13-20, 2002.
8) 石田利永子, 本田晋也, 高田広章, 福井昭也, 小川敏行,（ルネサステクノロジ）：マルチプロセッサ対応 RTOS におけるロードバランス機構の実現, 情報処理学会 組込みシステム研究会 第 14 回研究発表会, Jul 2009.
9) TOPPERS プロジェクト, http://www.toppers.jp/