社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（２）
2007／3／15

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

深江 輝昭† 本田 晋也† 冨山 宏之† 高田 広章†

†名古屋大学大学院情報科学研究科 〒464-8601 愛知県名古屋市千種区不老町
E-mail: †{fukae,honda,tomiyama,hiro}@ertl.jp

あらまし 近年，組込みシステムでもマルチプロセッサを用いる機会が増えている．マルチプロセッサ向けRTOSには様々なタイプがあるが，組込みシステムにおいては，リアルタイム性保証や検証性の観点から，各プロセッサへタスクを静的に配置する機能分散マルチプロセッサ向けRTOSを用いるのが一般的である．しかしタスクを静的に配置することは時間制約が緩いタスクの処理を待たせる可能性が高い．そこで本稿では，機能分散マルチプロセッサ向けRTOSで時間制約が緩いタスクがマイグレーションできる様拡張し，スループットの向上を実現した．またこの結果，従来通りのタスク処理は大よそ変わらないが，マイグレーション可能タスク処理が関係するとオーバヘッドが増加することを確認した．

キーワード リアルタイムOS，マルチプロセッサ，タスクマイグレーション

Task Migration in a Real-Time Operating System for Functionally Distributed Multiprocessors

Teruaki FUKAE†, Shinya HONDA†, Hiroyuki TOMIYAMA†, and Hiroaki TAKADA†

† Graduate School of Information Science Nagoya University Graduate School of Information Science, Nagoya University Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan
E-mail: †{fukae,honda,tomiyama,hiro}@ertl.jp

Abstract Recently, multiprocessor-based designs have become popular in embedded systems. Although there are some types of RTOS for multiprocessors, a RTOS for functionally distributed multiprocessor that allocates tasks statically according to their time constraints is more appropriate for embedded systems from the viewpoint of guaranteeing the real-time and easy of verification. However, it has the high possibility of making the tasks that have loose time constraints wait long time. To cope with this problem, in this work we make it possible that migrating the tasks with loose time constraints, which achieves higher throughput. From the measurement results, we found that the overhead for scheduling tasks with migration has been increased largely comparing to normal tasks without migration.

Key words Real-Time OS, Multiprocessor, Task Migration

1. はじめに

近年の組込みシステムでは，アプリケーションの大規模化，複雑化の傾向が高まっている．この傾向から処理能力の向上が求められたが，高性能シングルプロセッサを適用する場合，消費電力の増大等の問題があった．そこで組込みシステムでもマルチプロセッサを利用することで消費電力やコストを抑制している．

組込みシステムでは，リアルタイム性保証や検証性の観点から，タスク配置を静的に行える方が良い．

ところが，一般的なマルチプロセッサで用いられる対称型マルチプロセッサ向けカーネル（SMPカーネル）は，高いスループットを得るために各プロセッサへの自動的なタスク分配を行う．

そこで，組込みシステムでは，タスク配置を静的に行うマルチプロセッサ向けRTOSとして，機能分散マルチプロセッサ向

けカーネル（FDMP カーネル）[注1]を用いるのが一般的である．

一方で，タスク配置を静的に行うことは，負荷変動する様な場合に時間制約が緩いタスクの処理を待たせる可能性が高い．この結果，負荷バランスを考慮して各プロセッサへタスクを最適配置した場合と比較して，スループットが大幅に低下する可能性がある．

これに対し，時間制約の緩いタスクをアイドル状態プロセッサで実行させた場合，より高いスループットを得られると考えられる [1]．

そこで本稿では，リアルタイム性を保証すると同時に，スループットを向上させることを目的として，FDMP カーネルへマイグレーション可能なタスクを導入する．なお，拡張ベースとして TOPPERS プロジェクト [2] で開発された TOPPERS/FDMP カーネル [3] を利用する．

2. TOPPERS/FDMP カーネル

TOPPESR/FDMP カーネルは μITRON4.0 仕様 [4] の標準的な機能セットであるスタンダードプロファイルに準拠して開発された，同期・通信機能拡張した FDMP カーネルである．

TOPPERS/FDMP カーネルではあるプロセッサが他のプロセッサの資源を操作する際に，直接操作法を用いている．この方法では，操作対象の資源が属するプロセッサのメモリを直接アクセスする．資源操作により資源が属するプロセッサでディスパッチが必要となった場合には，資源操作をしたプロセッサが対象プロセッサに対して割込みを発生させ，ディスパッチを要求する．

3. 仕様拡張

3.1 タスクの分類化

拡張カーネルではマイグレーション不可能なタスクでリアルタイム性を保証し，マイグレーション可能なタスクでスループットを向上させることを行う．そこで，両タスクはその特性が異なることから，次の様に分類する．

ローカルタスク :特定プロセッサ上でのみ実行可能なタスクであり，静的にタスク配置が決定される．

グローバルタスク :全てのプロセッサ上で実行可能なタスクであり，必要に応じて自動的にマイグレーションされる．

3.2 スケジューリング規則

両タスクがその特性を維持するためには，グローバルタスクが常にローカルタスクに実行優先権を譲る必要がある．

グローバルタスク間に優先度が存在しない場合，単純なスケジューリングしかできないため，拡張カーネルではグローバルタスク間にも優先度を設定している．そしてローカルタスクがリアルタイム性を保証可能であることを維持するため，次の様にスケジューリング規則を設定した．

（1） ローカルタスクは常にグローバルタスクより優先されて動作する．

（2） ローカルタスク間もグローバルタスク間も，設定された優先度順に動作する．

また，グローバルタスク間で同期・通信等の操作を行う際に，ローカルタスク間の同期・通信等の操作を妨害しない様，両タスクが扱う資源も異なるものを扱う必要があると考えられた．

更に同様の理由から，グローバルタスク用資源の専用のロックとして，グローバルタスクロックとグローバルオブジェクトロックを導入した[注2]．

そこで次の様に両タスクが扱う資源を分類し，その集合も異なるクラスとしてまとめた．

ローカルオブジェクト :各クラスに属し，ローカルタスク間同期・通信のためや，カーネル，操作対象のために用いるセマフォやミューテックスなどの資源．

グローバルオブジェクト :グローバルタスク制御や，グローバルタスク間同期・通信のために用いるセマフォやミューテックスなどの資源．

ローカルクラス :ローカルタスクとローカルオブジェクト及び，ローカルタスク用のロックやレディキュー等の集合．

グローバルクラス :グローバルタスクとグローバルオブジェクト及び，グローバルタスク用のロックやレディキュー等の集合．

3.3 サービスコールの仕様拡張

拡張カーネルではグローバルオブジェクト操作機能をユーザに提供するため，資源へのアクセス権を拡張する表 1 の様なサービスコール仕様拡張を行った．

表 1 サービスコール仕様拡張

	ローカルオブジェクト	グローバルオブジェクト
ローカルタスク	アクセス可能	アクセス可能
グローバルタスク	アクセス制限	アクセス可能

基本的には各タスクから各オブジェクトへのアクセスは可能であるが，リアルタイム性の保証を可能とするため，グローバルタスクはローカルオブジェクトへの不必要なアクセスを制限する必要がある．

3.4 ID 番号の拡張

サービスコール仕様拡張に合わせて，従来のサービスコールによってグローバルオブジェクトを操作可能とするため，操作対象を指定する引数となる ID 番号を次の様に拡張して区別した．

ローカルオブジェクト用 ID :最上位ビット 0

グローバルオブジェクト用 ID :最上位ビット 1

4. 実装環境

本稿の実装環境を表 2 に示す．そして図 1 に実装に対するメモリマッピングを示す．

実装では，ローカルタスクプログラムとデータは，ローカルメモリとして利用する，各プロセッサのオンチップ RAM に格

(注1)：対称型マルチプロセッサ向けカーネル（AMP カーネル）とも呼ぶ．

(注2)：ロックの取得にはスピンロックを用いている．

表 2　本研究で利用したマルチプロセッサ回路構成

命令セット	32 ビット RISC
プロセッサ数	3 個
各プロセッサ周波数	50MHz
クロック周波数	50MHz
メモリ	オンチップ RAM プロセッサ毎に 64KB 共有メモリ 88MB
キャッシュ	命令キャッシュ 4KB
排他制御機構	Mutex ロック 8 個
バス構成	Avalon バス

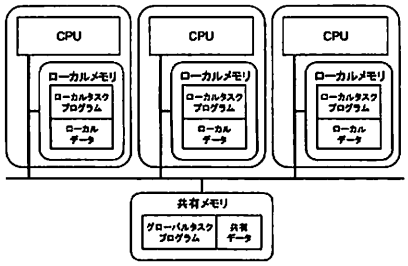

図 1　メモリマッピング

納する．またグローバルタスクプログラムや共有データは，共有メモリに格納する．

5. 実　　装

5.1 グローバルレディキューの設計と実装

設計と実装を最初に行ったのは，グローバルタスク用のレディキューである．グローバルタスク用のレディキューを実装する際には，幾つかの点で考慮が必要となった．その項目を次に挙げる．

（1） ローカルタスク用のレディキューとグローバルタスク用のレディキューを分離するかどうか（分離する/分離しない）

（2） グローバルレディキューを何処で所持するか（全体で1つ/プロセッサごと）

（3） 実行状態グローバルタスクと実行可能状態グローバルタスクをどう扱うか（同一キュー/異なるキュー）

（1）に関しては，ローカルタスクの実行をグローバルタスクが妨害しないことが重要であることから分離させた．そして（2）は，グローバルタスクも優先度を持ち優先度ベースに動作するため，全体で1つにした方が効率が良いことから，共有メモリ上にグローバルタスク用のレディキューを配置することにした．

この結果，ローカルタスク用のレディキューとグローバルタスク用のレディキューが明確に分離されたため，次の様に定義した．

ローカルレディキュー　:ローカルタスク用のレディキュー

グローバルレディキュー　:グローバルタスク用のレディキュー

また（3）は（1）と（2）より，ローカルレディキューと分離して全体で1つのグローバルレディキューとして設計したため，最大nプロセッサ数個存在する実行状態グローバルタスクを，一箇所で管理する必要が生じたことによる問題である．そこで同一キューのモデルと異なるキューのモデルを比較検討し，実装の簡易さと十分なスケーラビリティを持つと予測されることから，両タスクを異なるキューで管理することとした．

このグローバルレディキューと分離した，実行状態グローバルタスクを管理するためのキューを，次の様に定義する．

グローバルランキュー　:実行状態グローバルタスクのキュー

図2に，実装したグローバルレディキューとグローバルランキューを示す．グローバルランキューは，本稿の環境では実行状態グローバルタスクの数が少ないことから，優先度順に直列に連結するキューとした．一方で，グローバルレディキューは連結タスク数が多いと考えられることから，優先度順に並列に連結するキューとした．なお，実行状態タスクが連結されていない点を除くと，グローバルレディキューはローカルレディキューと同じ形式を採っている．

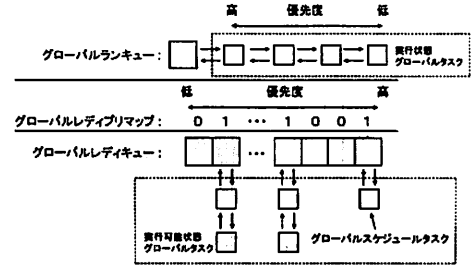

図 2　グローバルレディキューとグローバルランキュー

また拡張カーネルではローカルレディキューと同様に，管理を容易とするため最高優先度実行可能状態グローバルタスクをグローバルスケジュールタスクと呼び別個の変数に登録している．同様に，グローバルスケジュールタスクを探索し易くするため，探索用のビットマップとしてグローバルレディプリマップを設置している．一方，実行状態グローバルタスクもまた管理を容易とするため，グローバルランタスク配列を用意して状態遷移ごとに格納と消去を行っている．

5.2 グローバルタスクスケジューラ

グローバルタスクはスケジューリング規則により，ローカルタスクより優先度が常に低く，またグローバルタスク間でも優先度順に従ってスケジューリングされなければならない．この結果，次の2つの状況において，実行状態グローバルタスクより実行可能状態グローバルタスクの方が優先度が高い，優先度逆転状態が発生する可能性がある．

（1） グローバルタスクを実行中のプロセッサでローカルタスクが起動した場合

（2） グローバルタスクを起動・起床した場合

（1）ではローカルタスクによるリアルタイム性保証を可能とするため，グローバルタスクを実行中のプロセッサは即座にローカルタスク実行へ移らなければならない．その結果，実行していた高優先度グローバルタスクが実行可能状態に遷移させられる．また（2）では単純にグローバルタスクを起動・起床した際に，高優先度グローバルタスクが実行可能状態になる可能性がある．

（1）及び（2）の状況において，グローバルタスクを実行可能なアイドル状態プロセッサが存在する場合，そのプロセッサがグローバルタスクをスケジューリングするため，優先度逆転状態を解消させる操作は不要となる．一方，存在しない場合，特定プロセッサ上でグローバルタスクが実行されているならば，優先度逆転状態の可能性がある．この場合，実行状態グローバルタスク中の最低優先度のものより実行可能状態グローバルタスクの優先度の方が高いならば，優先度逆転状態である（この一連の判定を優先度逆転判定と呼ぶ（図3））．

そして拡張カーネルでは，リアルタイム性を重視することから，優先度逆転が発生確認された場合，ディスパッチを指示して即座に解消させる．

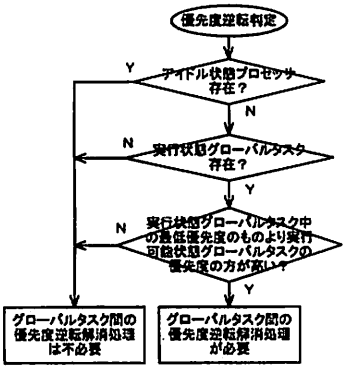

図3　優先度逆転判定

まず（1）の場合であれば，グローバルタスクを実行状態から実行可能状態に遷移させた後，図3の様に判定すると同時に，ディスパッチを実行させるプロセッサを決定する．そしてこの処理を実行させるため，共有変数として用意しているグローバルタスクディスパッチ要求フラグの対象プロセッサに対応するフラグを立て，対象プロセッサに割込みを発行する．割込みを掛けられたプロセッサは，自身が優先度逆転状態であるかを判定してからディスパッチする．

（2）の場合は（1）と異なり，自プロセッサが実行状態グローバルタスク中の最低優先度のものを実行している可能性が残る．もし優先度逆転判定の実行後にディスパッチ対象が自プロセッサであったならば，グローバルタスクの中断処理実行後にディスパッチャへと移行しディスパッチする．なお対象が他プロセッサの場合は，（1）と同様の処理を実行する．

5.2.1　優先度逆転の再判定

ところが，グローバルタスクはローカルタスクと異なり，ディスパッチ対象プロセッサが明確に定まっていない．そこで，実行状態グローバルタスク中の最低優先度のものを実行中のプロセッサが，ディスパッチ対象に選ばれる．

しかし，図4の様にほぼ同時に高優先度グローバルタスクを2つ起動した場合，実際にはPE2とPE3でディスパッチしなければならないが，PE2にのみグローバルタスクディスパッチ要求割込みを発行してしまう．そしてこのままではPE2でのみディスパッチが実行され，PE3の優先度逆転は解消されない．

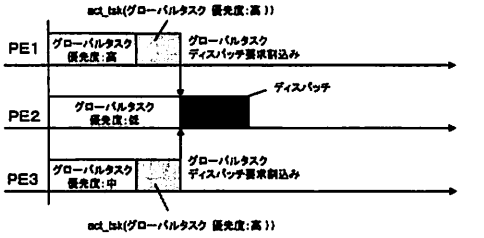

図4　グローバルタスク起動の重複

これを回避する最も簡単な方法として，グローバルタスクロックを優先度逆転判定からディスパッチ処理完了まで取得し続けることが挙げられる．しかしその様な処理は，ローカルタスクディスパッチ等の処理を妨害する可能性が高く，リアルタイム性の保証を不可能にする．

その結果，先の2つの条件によってディスパッチが発生した際には，他の場所でも優先度逆転が発生していないか必ずチェックし，優先度逆転状態となっているならば解消させなければならない．

拡張カーネルでは図5に示す様に，グローバルタスクディスパッチを実行したプロセッサ自身が再び優先度逆転判定を実行し，割込みを発行する．そして割込みを受けたプロセッサもまたディスパッチ実行後に同じ様に優先度逆転判定を実行する．この連鎖によって最終的に全てのグローバルタスク間の優先度逆転状態を解消させる．

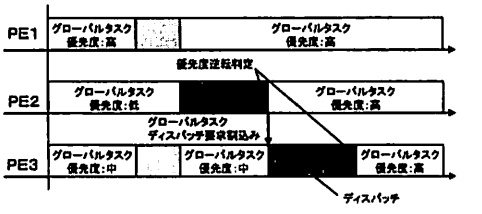

図5　優先度逆転の再判定

一方，グローバルタスクからローカルタスクへのディスパッチ中のプロセッサに対して，優先度逆転解消のための割込みが発行される場合がある．この場合，グローバルタスクディスパッチは実行しないが，割込みを掛けられたプロセッサは優先度逆転の再判定を行って再割込み発行を行わなければならない．

また，優先度逆転判定時に，あるプロセッサがアイドル状態だった場合，そのプロセッサがローカルタスクディスパッチしようとしていても，割込みは発行されない．この結果，グローバルタスクディスパッチが必要なプロセッサに割込みが発行されない場合がある．このため，アイドル状態からタスクディスパッチする際には，必ず優先度逆転判定を実行する必要がある．

5.2.2　多重ディスパッチ割込み

また，グローバルタスクのディスパッチが複数回発生する場合，拡張カーネルでは特定プロセッサへと割込みが集中してしまう．この結果，前述したグローバルタスクからローカルタスクへのディスパッチ中のプロセッサにも，グローバルタスク

ディスパッチ要求割込みが集中する可能性がある．そしてその場合，割込み処理による遅延の増加やロック取得妨害が行われ，ローカルタスクの動作が妨害される．

これを防ぐため，拡張カーネルでは他プロセッサにディスパッチを指示する場合，対象プロセッサに対応するフラグを立てて割込み発行することから，フラグを立てる際に既に立てられていた場合には，割込み発行しない方法を採っている．

そこでグローバルタスクディスパッチを要求する割込みは，同時に1つしか発行されず，ローカルタスクディスパッチ時の割込みによる遅延もまた最小限に抑えられる．

5.3 ディスパッチャ拡張

拡張カーネルでは，定義したスケジューリング規則に則って，グローバルタスクがローカルタスクの実行を妨害しない様に従来のディスパッチャを拡張した．拡張ディスパッチャでは，全ての実行可能状態ローカルタスクをスケジューリングし終えてから，グローバルタスクがスケジューリングされる．そして全てのローカルタスクとグローバルタスクを実行し終えた場合，ローカルタスクディスパッチ要求が出されるかグローバルタスクが実行可能状態になるまでアイドルルーチンを実行し続ける．

5.4 割込み出口拡張

割込み処理等の結果，割込みから通常処理へと帰還する際に，元のタスクではなく新しいタスクへとディスパッチしたい場合がある．グローバルタスク導入後にはこの際の処理が大きく3通りに分かれる．

（A） 実行状態ローカルタスクが存在する場合の処理
（B） 実行状態ローカルタスクは存在しないが，実行可能状態ローカルタスクが存在する場合の処理
（C） どちらも存在しない場合の処理

（A）の処理はほぼ従来の処理と変わりがない．一方，（B）と（C）では，グローバルタスクを実行中であると想定され，グローバルタスクを実行状態から実行可能状態へと遷移させるための処理が必要となる．また結果として，グローバルタスク間の優先度逆転の解消が必要である可能性もある．そこで優先度逆転判定を実行させるためのフラグである，優先度逆転判定実行フラグを立ててディスパッチャへと移行し，ディスパッチャでフラグを確認してクリアした後，優先度逆転判定を実行する．

（C）の場合はまた，様々な要因によって，ディスパッチが不要となっている可能性があるため，処理の途中で優先度逆転判定を実行し，もし優先度逆転状態でないのならば，元の処理へと帰還する．しかし，この判定によってディスパッチが必要だと判断された場合，ディスパッチャへと移行してグローバルタスクディスパッチを行う．そしてその場合にも，優先度逆転が発生していないか確認を取る必要がある．そこで，この処理では実質的には合計4回のキュー操作（グローバルタスクをグローバルランキューからグローバルレディキューに再連結させる操作と，グローバルレディキューからグローバルランキューに再連結させる操作）と2回の優先度逆転判定が必要となる．

6. オーバヘッド計測

ここでは拡張カーネルにおける各オーバヘッドを計測する．計測では処理実行直前と実行直後にそれぞれ時間計測を行い，差分値を取得する計測を各1000回実行した．そしてその分布に対して中央値となる時間とその値の近傍値を範囲として表にまとめた．また取得できた各オーバヘッドを，TOPPERS/FDMPカーネルのオーバヘッドと比較した．

6.1 割込みに関するオーバヘッド

ここでは割込み発生時の処理を，（1）割込み発生から割込み処理に入るまでの時間（割込み応答時間），（2）割込みが終了してから元の処理へ戻る際の時間（割込み帰還時間），（3）割込み処理の中でディスパッチが指定されていた場合にディスパッチする際の時間（割込みディスパッチ時間），として，拡張カーネルにおける各オーバヘッドを計測しTOPPERS/FDMPカーネルと比較する．

特に（3）の場合は，（A）実行状態ローカルタスクが存在する場合の処理，（B）実行状態グローバルタスクは存在しないが，実行可能状態ローカルタスクが存在する場合の処理，（C）どちらも存在しない場合の処理，それぞれについて計測した．

計測結果を表3〜5に示す．

表3 割込み応答時間

条件	時間（μs）	範囲（μs）
TOPPERS/FDMP カーネル	3	+2
拡張カーネル	3	+2

表4 割込み帰還時間

条件	時間（μs）	範囲（μs）
TOPPERS/FDMP カーネル	1未満	0
拡張カーネル	1未満	0

表5 割込みディスパッチ時間

条件	時間（μs）	範囲（μs）
TOPPERS/FDMP カーネル	3	+1
拡張カーネル（A）	8	+1
拡張カーネル（B）	14	+1
拡張カーネル（C）	27	+1

表3の結果は，TOPPERS/FDMPカーネルと拡張カーネルでは割込み応答時間に変化が無いことを示している。表4もまた，表3と同様にTOPPERS/FDMPカーネルと拡張カーネルで変化が無いことを示している．この両結果より，TOPPERS/FDMPカーネルと拡張カーネルでは，ディスパッチが発生しない場合の割込み処理は変化が無いことが確認された．

一方で，表5からはオーバヘッド増加が確認される．

（A）のオーバヘッド増加は，処理の大部分がTOPPERS/FDMPカーネルと同じだが，優先度逆転判定が加わっていることで発生している．それに対し（B）では，ローカルタスクでは必要としないキュー操作が，グローバルタスクでは実行状態から実行可能状態に遷移させる際に必要となることが影響し，（A）よりオーバヘッドが大きくなっている．

（C）は前述した通り，4回のキュー操作と2回の優先度逆転判定が必要となり，（B）の処理を2倍したに等しい．その結

果，(C) は (B) の2倍程度，オーバヘッドが増加している．

オーバヘッド増加に関して，重要だと考えられるのは割込みディスパッチ時間 (A)，(B) である．(A) の処理は割込みによるローカルタスクディスパッチ時間に用いられ，(B) の処理はグローバルタスクからローカルタスクへのディスパッチ時に用いられるため，この両処理のオーバヘッド増加はリアルタイム性に影響を及ぼしていると考えられる．

6.2 ディスパッチオーバヘッド

ここではディスパッチオーバヘッドを計測する．ただし，ディスパッチの指示対象は自プロセッサのみであり，他プロセッサに対する場合は無視する．これは他プロセッサに対しての指示の場合，割込みディスパッチ時間が影響し，自プロセッサに対する場合とは異なる傾向の値を示すためである．

処理は act_tsk サービスコールによって引き起こされた場合と，タスク終了によるスケジューリングの場合の2通り行う．そして拡張カーネルにおける，ローカルタスクからローカルタスク，ローカルタスクからグローバルタスク，グローバルタスクからローカルタスク，グローバルタスクからグローバルタスクへの各ディスパッチ処理時のオーバヘッドを計測し，TOPPERS/FDMP カーネルによる結果と比較する．

計測結果を表 6〜7 に示す．

表 6 ディスパッチオーバヘッド (act_tsk 起動)

条件	時間 (μs)	範囲 (μs)
TOPPERS/FDMP カーネル	8	+1
拡張カーネル (ローカル→ローカル)	9	+1
拡張カーネル (グローバル→ローカル)	30	-1, +4
拡張カーネル (グローバル→グローバル)	31	±2

表 7 ディスパッチオーバヘッド (タスクスケジューリング)

条件	時間 (μs)	範囲 (μs)
TOPPERS/FDMP カーネル	9	0
拡張カーネル (ローカル→ローカル)	9	+1
拡張カーネル (ローカル→グローバル)	18	+1
拡張カーネル (グローバル→グローバル)	17	-1, +3

表 6 から，TOPPERS/FDMP カーネルと，拡張カーネル (ローカル→ローカル) のオーバヘッドに変化が無いことが分かる．一方，拡張カーネル (グローバル→ローカル) や (グローバル→グローバル) では，オーバヘッド増加が起きている．ただし，割込みディスパッチの場合と異なり，更に act_tsk サービスコールのオーバヘッドが含まれているため，タスク起動処理で TOPPERS/FDMP カーネルとローカルタスクではキュー操作1回，グローバルタスクではキュー操作1回と優先度逆転判定1回が実行される．これに加え，拡張カーネル (グローバル→ローカル) ではキュー操作2回と優先度逆転判定1回，拡張カーネル (グローバル→グローバル) では，キュー操作4回と優先度逆転判定2回の処理が行われているため，グローバルタスクの影響する処理ではオーバヘッドが割込みディスパッチ時より大きい．なお，拡張カーネル (グローバル→ローカル) のオーバヘッドが，表 5 の傾向と異なり，拡張カーネル (グローバル→グローバル) に近い値となっているのは予想外であり，この点に関しては未だ解明できていない．

一方，表 7 からも，TOPPERS/FDMP カーネルと拡張カーネル (ローカル→ローカル) のオーバヘッドに変化が無いことが分かる．また，表 7 の場合は，表 6 よりも，グローバルタスクディスパッチ時のオーバヘッドが小さいことも分かる．この原因として，1つはグローバルタスク終了からのディスパッチであるため，優先度逆転判定が不要であることが挙げられる．またもう1つの原因として，グローバルタスクを終了させる際，グローバルランキューからのタスク排除だけで再連結処理が不必要であることが挙げられる．そして更に，サービスコールを実行していないことも関係する．

以上より，ローカルタスク同士での動作に関してディスパッチオーバヘッドの増加は無いことが確認された．一方で，割込みディスパッチ時の様に，グローバルタスク処理が関係すると，オーバヘッドが増加することが確認された．

7. おわりに

本稿ではリアルタイム性保証と同時に，マイグレーションによってアイドル状態プロセッサを活用し，スループットを向上させるための手法とその際のオーバヘッドを示した．

結果として，従来通りの処理ではオーバヘッドが変わらないが，グローバルタスクの処理が関係することでオーバヘッドが増加することが確認された．この増加の主な要因は2つ挙げられ，1つはグローバルランキューとグローバルレディキューの再連結操作であり，もう1つは優先度逆転判定の実行であった．

今後の課題としては，今回の手法によってリアルタイム性の保証が実際に可能であるかどうか，リアルタイムスケジューリング理論を適用したモデルを利用した検証によって明らかにすることが挙げられる．また，今回ではタスク以外のオブジェクトを操作した場合の影響などまでは調査できなかった．これらについてもまた，グローバルタスク独特の処理が必要となると考えられることから調査しなければならない．

文　献

[1] 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.

[2] TOPPERS project, http://www.toppers.jp/

[3] Shinya Honda, Hiroyuki Tomiyama, Hiroaki Takada. "RTOS and Codesign Toolkit for Multiprocessor Systems-on-Chip", In Asia and South Pacific Design Automation Conference(ASP-DAC), 2007 .

[4] 坂村 健 監修, 高田 広章 編. "μITRON4.0 仕様 Ver.4.02.00", トロン協会, 1999.