システムＬＳＩ設計技術　105－9
（２００２．３．５）

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

蛸井 基明　宮内 新　石川 知雄　　高田 広章
武蔵工業大学　　豊橋技術科学大学

あらまし　機能分散型マルチプロセッサ上にリアルタイムシステムを構築する際，タスクをタスククラスに分類し，システムの応答性を確保する手法が提案されている．我々はシステムスループット向上を目的として，新たにプロセッサ間を移動可能なタスククラスの導入を議論してきた．本タスククラスはアイドルプロセッサに対し動的にタスクを割り当てるもので，管理上新たな操作が必要である．本稿では，このタスククラスの導入に際して必要となる具体的な処理について検討を行い，実環境上へ実装を行った．また管理オーバヘッドの側面から性能検証を行い，検証結果に基づいた性能改善の手法についても議論・検証を行った．

Introduction of migratable task on function-distributed multiprocessor environment

Motoaki Takoi　Arata Miyauchi　Tomo Ishikawa　　Hiroaki Takada
Musashi Institute of Technology　　Toyohashi University of Technology

Abstract　When a real-time system is implemented on a function-distributed multiprocessor,the method classifies tasks into task class and improves the response of system has been suggested. As a means to increase the system throughput introduction of task class which tasks can migrate between processor has been discussed. This class allocates a task on the processor dynamically and needs new oparation to manage. In this study, the concrete operation which needed on introduction of this task class was dissucused, and implementation of system to real enviroment was carried out. Also, the system performance was verified on the side of a management overhead, and a method which improves overhead based on the verified result has been discussed.

1　はじめに

近年アプリケーションの複雑化・高機能化に伴い，リアルタイム制御システムに対する高性能化要求が高まっている．特に多数の入出力装置が接続されるようなシステムでは，図1に示すような機能分散型マルチプロセッサシステム (以下，機能分散 MP) の採用が有効な高速化手段の一つとして挙げられる．

リアルタイムシステムでは，時間要求の厳しい処理に対する応答性を保証する必要があり，特にマルチプロセッサ環境上では資源競合による影響が問題となる．そこで，タスクを処理の性質からタスククラスに分類し，実行管理を行う方法が提案されている [1].

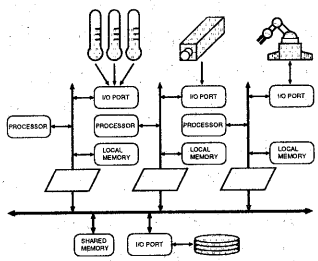

図 1: 機能分散型マルチプロセッサシステム

2　背景・目的

現在までにその効果が確認されているタスククラスの分類法はシステムスケーラビリティ確保を主眼としたもので，システム上の各タスクは，あらかじめ決められたプロセッサエレメントのみで

実行可能であることが前提であった．そのため，システム全体で見るとタスク実行に柔軟性が無く，またアイドル状態となったプロセッサはその処理能力を有効利用することができない．

そこで本研究では，新たに処理を行うプロセッサを限定せず，プロセッサ間を移動しながら処理を行うタスククラスを導入する．そして，処理プロセッサを限定する必要の無い低優先度タスクについてはこのタスククラスに分類し，実行するプロセッサに柔軟性を持たせることによってシステムスループットの向上を図ることが本研究の目的である．

また，我々は本タスククラスの導入手法について議論して来た [2][3] が，現段階ではシステム性能全体への影響が明らかになっておらず，具体的な実装による性能検証を必要としていた．

そこで本発表では，この新たなタスククラスであるグローバルクラスを実環境上への実装を行い，基本的なシステム性能の検証を行ったので報告する．また本件では実環境上での性能に基づき，現実的な管理操作の改良についても検討を行った．

3 タスククラスの概念

タスククラスとは，機能分散型マルチプロセッサシステムにおいて，高リアルタイム処理の応答性を保証するために導入された手法である．すなわち，共有資源の利用可能性とリアルタイム性要求を元にタスクをタスククラスに分類し，高リアルタイム処理へのプロセッサ数の影響を極力抑えることを目的としている．

本研究では，アイドルプロセッサ資源の有効活用を行うことが目的であるので，処理プロセッサ固定の処理に関しては分類を行わず，以下の 2 つのタスククラスを定義する．

3.1 ローカルクラス

処理プロセッサが固定である従来からのタスククラスが本タスククラスに該当する．したがって，以下の特徴を定義できる．

- 各プロセッサ毎に 1 つ存在 (システム全体ではプロセッサ数と同数)
- タスクの処理プロセッサは固定であるが，他 PE 上の資源にもアクセス可能．

ここでの他 PE 上の資源とは，各 PE 上において公開されているローカルクラスのタスク，タスク管理オブジェクト，同期通信オブジェクトを意味する．すなわち，各プロセッサ上のローカルクラスは他プロセッサ上のローカルクラスに対してシステムコール発行が可能である．これにより，管理オブジェクトに対して排他制御が必要になるが，他 PE からの資源アクセスの可能性は低いため，リアルタイム性はほぼ保証できる．

3.2 グローバルクラス

本研究で導入を提案するグローバルクラスには，以下の性質を定義する [2]．

- タスクの処理プロセッサをを限定しない．
- クラス間優先度はローカルクラスよりも低優先度
- タスク実行には上位クラスの実行から解放されたプロセッサを用いる
- 特定の PE が専有する非共有資源はアクセス不可とする

これにより各 PE 上の時間制約の厳しい処理に対するリアルタイム性を確保しつつ，アイドル状態の CPU 資源を有効活用することを可能としている．また，処理を行うプロセッサを限定しない性質を利用し，高優先度タスクにプリエンプトされた場合でも処理を他プロセッサに移すことで実行を継続する，“タスクマイグレーション”が可能である．

なお，クラス内のタスクスケジュールは他のタスククラスと同様，タスクに設定した優先度をもとに行う．

4 グローバルクラスの実装

グローバルクラスは上記の基本性質により，システム全体で一つのタスククラスを共有することとなる．そのため，ローカルクラスとは根本的に異なった実装方法が必要となる．

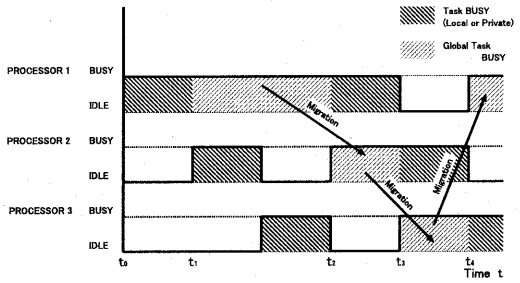

図 2: タスクマイグレーション (概念図)

4.1 メモリ空間への配置

グローバルクラスは，特定のプロセッサに負荷を与えることなく，また各プロセッサ上から共通した操作でアクセス可能であるよう，メモリ空間上に配置することが要求される．そこで共有メモリ空間を利用する事となるが，グローバルクラスの全資源を共有メモリ上に展開した場合に共有バス負荷が過剰となる問題が指摘されている [3]．そこで定数データと変数データに分割し，以下のように配置し，上記要求を実現する．

- **静的データ (定数・プログラムコード):**
  各 PE のプライベート空間上に同一アドレスで配置
- **動的データ (変数):**
  共有メモリ空間上に配置

4.2 タスク管理操作

タスク管理上から見たグローバルクラスの特徴として，以下が挙げられる．

- 1 つのレディーキュー上で同時に複数のプロセッサがグローバルクラスを実行
- グローバルタスクを実行可能なプロセッサが動的に変動

従って，グローバルクラスの実装に際しては，単に実行タスクを割り当てるだけでなく，レディーキュー上のタスクに対するの実行権割り当てが適切な状態となるよう，新たな管理情報・操作が必要になる．

そこで，以下の管理データ構造を定義し，これを基に管理操作を再設計した．

1. **TCB[1] (割当プロセッサ情報の追加)**
   実行状態タスク判別 (レディーキュー上の位置だけでは判別不能)，切替え操作時の操作対象情報を提供
2. **レディーキュー**
   他段優先度にまたがった線形探索を考慮し 1 本の単純なリンクキュー構造
   (優先度単位での分割を行わない)
3. **アイドルプロセッサリスト**
   アイドルプロセッサを保持．
4. **プロセッサ認識情報構造体**
   プロセッサ識別情報を保持．またアイドル時の管理のためキュー構造を用意

また，効率化のため以下のシステム状態変数を定義する．

5. **割込み対象プロセッサ変数**
   現時点でプリエンプト対象となるタスク実行中のプロセッサへのポインタ
6. **プリエンプト対象優先度**
   5. で実行中タスクの優先度．この変数により 5. 更新の必要性を判定可能．

なお優先度の管理上，5. がアイドルプロセッサであった場合はグローバルクラス下限優先度よりも低位優先度タスク実行状態，実行中タスクが存在しない場合には上限優先度より上位優先度タスク実行状態として取り扱う．以上により，グローバルクラス内部に閉じたタスク切替えに関しては，切替え先対象の探索は不要になる．

以上のデータ構造によるグローバルクラスの管理状態の一例を図 3 に示す

4.2.1 実行タスクに関する優先度整合性の確保

グローバルクラス管理上問題になるのは，処理に関与する複数のプロセッサに対し，ランダムに

[1]TCB:Task Control Block

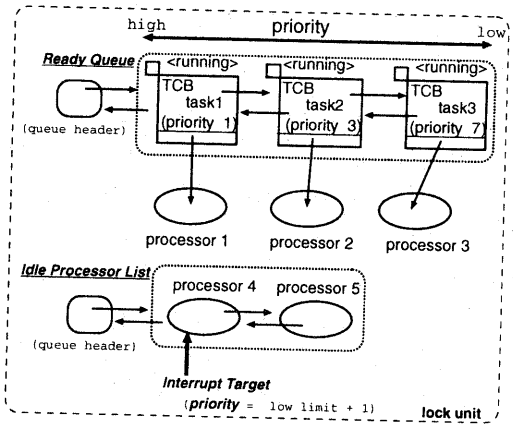

図 3: グローバルクラスの管理状態

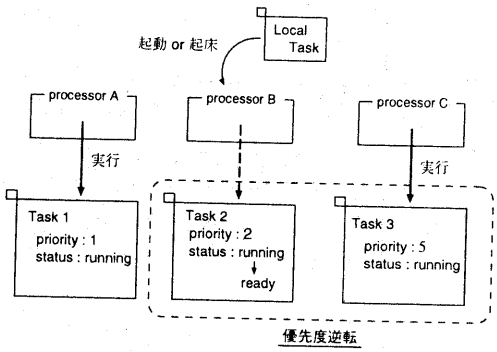

図 4: 優先度逆転状態

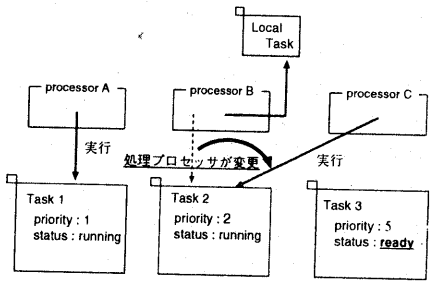

図 5: 優先度逆転の解消

上位優先度タスククラスからの実行権の横取りが発生する点である．

すなわち，グローバルタスクの実行に関与しているプロセッサのうち，非最低優先度のタスクを実行中のプロセッサに対してプリエンプトが発生した場合，グローバルクラス上には優先度逆転が生じてしまう (図 4)．

この現象の解消は，その時点で実行しているタスクの優先度が最も低いプロセッサが上位優先度の実行可能状態タスクにディスパッチすることによって実現する．従って優先度逆転を発生させるタスク切替操作側から，当該プロセッサに対しディスパッチャ起動要求を行えばよい．

以上の操作は，グローバルクラス内でのタスク実行の整合性を考慮した場合，発生した優先度逆転をリアルタイムに解消することが望ましい．しかしながら要求操作による管理オーバヘッドとトレードオフの関係が発生する．なおここでは，グローバルクラスのテスト実装としての位置付けから，管理オーバヘッドの影響検証を目的として，リアルタイムに解消操作を行う方針で実装する．

4.2.2 管理操作

タスクディスパッチ後の状態に不整合を残さないため，ローカルクラス内に閉じたディスパッチ以外は全て管理操作が必要である．そこで以下のデータ構造を用意し，タスクスケジュール機能をディスパッチャ上に集約実装する．

切替先状態ごとのグローバルクラス管理操作を図 6〜8 に示す．タスクディスパッチ処理の基本的な流れは，以下のようになる．

1. 切替先のタスククラスの認識 (ローカル／グローバル)
2. 実行中タスクの解放
3. スケジューラ実行
4. 管理情報更新
5. グローバルクラス内の優先度整合性の確認 (さらなる不整合があれば解消要求)

なお 5. は，割込み対象変数を基に処理の必要性を判断し，必要時のみ実行する．また，多重にディスパッチ要求を受け付けていた場合には，切替後に最低優先度実行中となるプロセッサに要求を転送する．

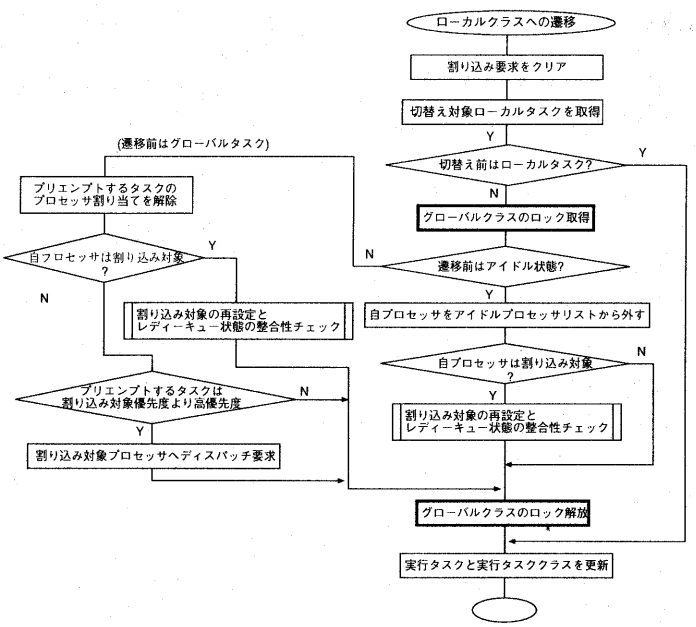

図 8: ローカルタスク実行状態へ移行時の処理

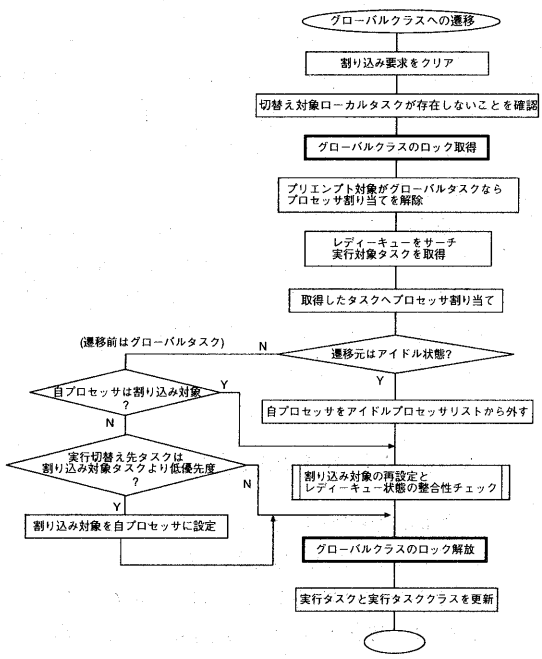

図 6: グローバルタスク実行状態への遷移

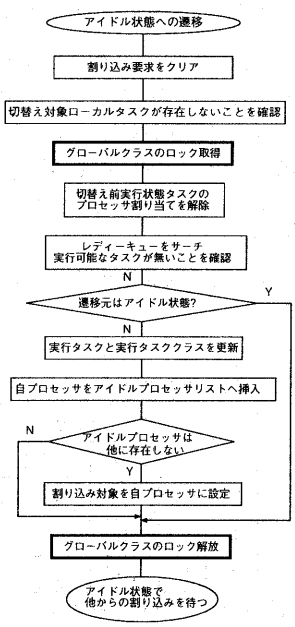

図 7: アイドル状態への遷移

5 評価実験

以下の評価環境にグローバルクラスを実装し，タスクディスパッチ時間を基準として性能評価を行った．

実験環境

- 電産 DVE68K/40 ×3
  (MC68LC040 33MHz)
- 共有メモリとともに VME バスで結合
- ソフトウエア開発・リモートデバッグ：
  PC 上のクロス開発環境（GNU 開発ツール（gcc, gdb 等）を利用して構築）
- 実装ベースカーネル：
  TOPPERS/JSP カーネルをマルチプロセッサ・タスククラス対応化したものを使用 (μ ITRON4.0 仕様準拠)

テスト条件

- 各プロセッサ上で周期的にローカルタスク起床
- その間に優先度の異なる 2 つのグローバルタスクを実行 (高優先度は休眠･起床を繰り返す)

また，上記のテストパターンの他，導入前後で処理の変わらないローカルクラス内に閉じたタスク切替についても比較のため測定を行った．なお，測定に際してはシステム時刻を提供するプロセッサを別途用意し，全プロセッサで統一した時刻基準で計測した．

各タスククラス実行状態間のタスク切替時間を表 1 に示す．

表 1: 各クラス間のディスパッチ時間 [μs]

		切替後		
		L	G	I
切替前	L	39	89	68
	G	102	128	91
	I	103	129	—

注）L:ローカル　G:グローバル　I:アイドル

表 1 より，非ローカルクラス実行状態 (グローバルクラス実行状態，アイドル状態) からローカルクラスへの実行切替には，ともにローカルクラス内でのディスパッチに比べ 3 倍程度の時間を要する事が判明した．このオーバヘッド増加分はグローバルクラスの優先度逆転の解消操作を含めた状態管理操作に起因しており，以下にて改善案を検討する．

5.1 優先度逆転解消のタイミング

グローバルクラス内でのタスク実行の整合性を考慮した場合，発生した優先度逆転は直ちに解消することが望ましい．そこでグローバルクラスのリファレンス実装となる上記の実装では，リアルタイムに解消操作を行う形態をとっている

しかしながら，これに伴ったローカルクラス応答性低下が確認でき，システムのリアルタイム性要求によってはこれが問題となる場合も想定される．特に、優先度逆転解消操作はグローバルクラスの共有資源へのアクセスを有するため，排他制御の影響による応答性低下が発生する．こうした場合には，リアルタイム性要求の低いグローバルクラスに関して，一時的な優先度逆転を限定的に容認し，タスク切替後処理（管理変数更新・優先度整合処理）を分離することで応答性が改善できる．なお，この場合には長期の優先度逆転存置を防止するため，分離した解消要求操作を別途システムコールとして実装する．一時的な優先度逆転を容認した場合の管理操作アルゴリズムを図 9～図 12 に示す．

なお，この方法では割込み対象変数は維持できなくなるため，切替後の整合状態の確認を省略できない．よってグローバルクラス内部の管理効率が低下し，タスク切替時間が容認前よりも増大する可能性がある．

5.2 改善効果の検証

性能評価実験で用いたモデル B を対象に本手法を適用し，性能比較を行った．

なお優先度解消システムコールは，プロセッサ 3 のローカルタスクから 245ms 周期で発行した．この平均実行時間は 56 μs である．

以上の手法により，ローカルクラスの応答性を改善できることは確認した．さらに詳細な動作検証を行った結果，数 10ms 程度の優先度逆転区間

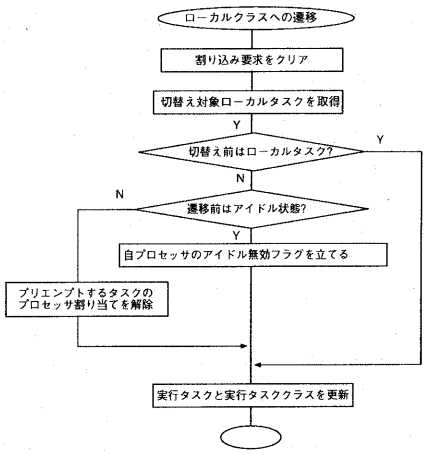

図 9: ローカルタスク実行状態へ移行時の処理

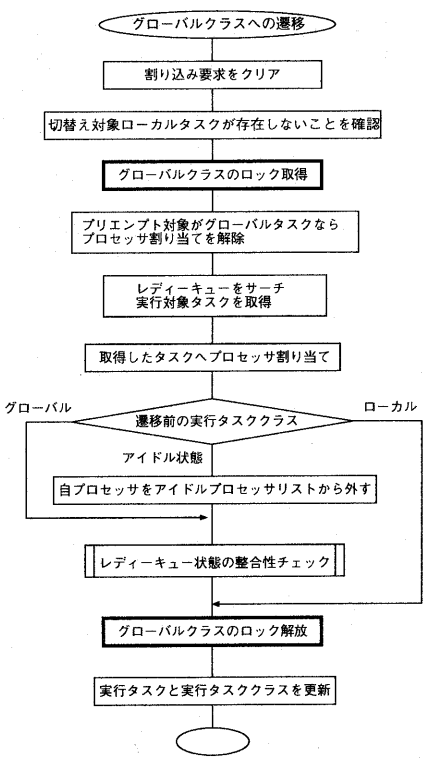

図 10: グローバルタスク実行状態への遷移

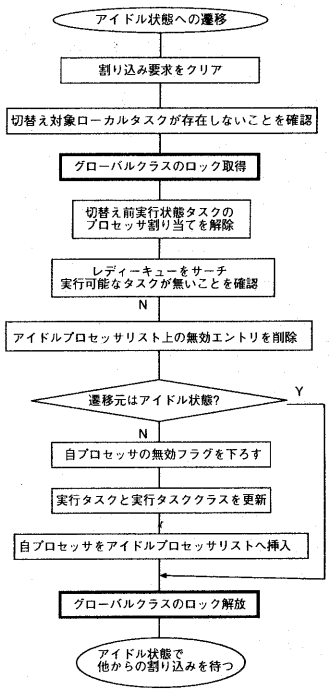

図 11: アイドル状態への遷移

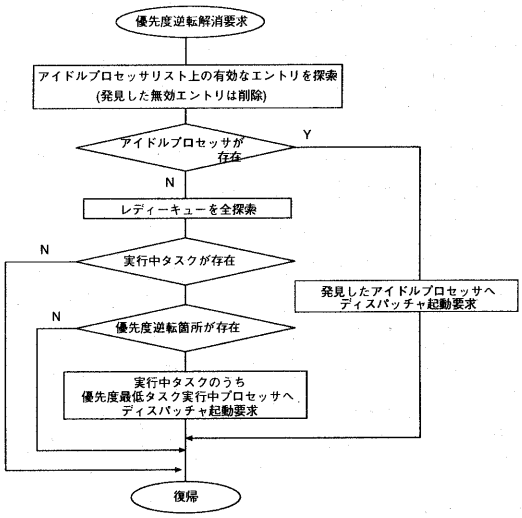

図 12: 優先度逆転解消システムコール

表 2: 一時的な優先度逆転容認の効果

タスク		切替時間 [μs]		改善率
前	後	当初	一時容認	(%)
L	G	89	84	-5.6
L	I	68	70	-2.9
G	L	102	51	50.0
G	G	128	138	-7.8
G	I	91	84	7.7
I	L	103	50	51.5
I	G	129	140	-8.5

L：ローカル G：グローバル I：アイドル

を確認した．ただし，いずれもシステムコール発行前に他の操作要因によって解決している．

6 その他の改善案

上記の実験では影響が顕在化しなかったが，優先度逆転の一部容認の有無にかかわらず，現状の管理アルゴリズムはレディーキューの線形探索区間を抱えている．そのため，タスク数，プロセッサ数の増加により探索区間が延び，システム規模増大につれて管理効率が悪くなる．特に最低優先度タスクの探索に関しては，これを保証可能な点まで行う必要があり，タスク数増に従って探索量が線形に増大してしまう．

そこでこれを低減する手法として，実行状態タスクの管理を行う新たな管理構造体 (ランキュー) の導入が考えられる [2][3]．具体的には実行状態タスクをレディーキューから外し，ランキューに優先度逆順につなぎ替えて管理する．これによりプリエンプト対象タスク決定の際の探索オーバヘッドが削減可能である．この方法は，特にタスク数，プロセッサ数が増加した場合に有効と考えられるが，その損益境界点については検証が必要である．

7 むすび

機能分散 MP 環境上に処理プロセッサを限定しないタスクを実行する機構を実装し，管理オーバヘッドの評価を行った．また，一時的にグローバルクラス内に優先度逆転を容認することにより，グローバルクラスの管理オーバヘッドによる応答性能低下の改善が図れることを示した．現状では，レディーキューの線形探索に起因するオーバヘッドがあり，これを改善することで性能改善の余地がある．同期通信機能への適用と合わせて今後の課題である．

参考文献

[1] 高田 広章，坂村 健，“非対称マルチプロセッサシステムのためのスケーラブルなリアルタイムカーネルの構想”，信学技報 (1995 年実時間に関するワークショップ RTP'95)，vol.94, no.573, pp.1-8,電子情報通信学会，Mar.1995.

[2] 鍋田 努，大畠 祐一，石川 知雄，高田 広章，“機能分散マルチプロセッサシステムにおけるマイグレート可能なグローバルタスクの導入”，信学技報 (1998 年 実時間処理に関するワークショップ RTP'98) vol.97, no.569, pp. 49-56,Feb.1998

[3] 大畠 祐一，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”，信学技報 (2000 年 実時間処理に関するワークショップ RTP2000),2000