Vol.2018-OS-144 No.2
2018/7/30

マルチコア*Tender*におけるメモリを介した遠隔手続呼出制御の方式の設計

藤戸 宏洋[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マルチコアプロセッサ環境で走行するオペレーティングシステム（以降，OS）は，共有データの不整合を防ぐ必要がある．共有データの不整合を避けるために排他制御を行うと，排他制御オーバヘッドのため OS の性能が低下する．***Tender*** では，OS が制御し管理する対象である OS 資源をコア毎に管理することにより，コア間での OS 資源の共有を防ぎ，排他制御を不要にする手法を実現している．しかし，この手法では，他コアが管理する OS 資源を操作することができない．そこで，コア間での OS 資源の操作依頼ができるように，他 OS に OS 資源の操作を依頼する遠隔手続呼出制御を拡張する．これにより，各コアの OS は，遠隔手続呼出制御を介し，計算機全体の OS 資源を同じインタフェースで利用できる．本稿では，遠隔手続呼出制御の改造量を少なくするため，コア間通信制御を計算機間通信制御と同じインタフェースで設計する方針について述べ，提案手法の実現における課題と対処について述べる．

Hiromi Fujito[1]　Toshihiro Yamauchi[1]　Hideo Taniguchi[1]

1. はじめに

マルチコアプロセッサが普及し，プロセッサに搭載されるコア数は増加傾向にある．マルチコアプロセッサでは，複数のコアによる並列処理により，オペレーティングシステム（以降，OS）の性能を向上させることが可能である．しかし，複数のコアが同じ共有データを同時に操作すると，処理に不具合が起きる．このため，排他制御が必要になり，共有データを利用する際にオーバヘッドが生じる．また，プロセッサに搭載されるコア数の増加により，同時に共有データを利用しようとするコアが増加している．これにより，排他制御のオーバヘッドが増加し，並列処理による性能の向上を妨げている [1]．

Tender[2] では，OS 資源（以降，資源）をコア毎に排他的に管理するマルチコア対応方式が実現されている（以降，個別型 ***Tender***）[3]．個別型 ***Tender*** では，資源をコア間で共有しないことにより，資源の排他制御を撤廃している．これにより，コア数の増加による排他制御のオーバヘッドの増加を抑制できる．

しかし，個別型 ***Tender*** では，各コアの OS は，他コアが管理する資源を操作することができない．これにより，各コアで動作するプロセスが計算機全体の資源を利用できない．

分散 OS である ***Tender*** には，資源操作を依頼する機能である遠隔手続呼出制御（以降，RPCC：Remote Procedure Call Controller）が実現されている [4]．RPCC を利用することにより，資源操作を依頼する計算機（以降，ローカル計算機）の資源と他の計算機（以降，リモート計算機）の資源を同じインタフェースで利用できる．

そこで，RPCC を拡張し，他コアに資源操作を依頼する機能を実現する．これにより，個別型 ***Tender*** において，計算機全体の OS 資源を同じインタフェースで利用できる．本機能は，資源をコア間で共有しないまま，他コアが管理する資源の操作を行える．また，コア数が増加した場合でも，各コアが個別に資源を管理するため，資源の排他制御は必要ない．

本稿では，コア間通信制御を計算機間通信制御と同じインタフェースで設計することで，RPCC の拡張による改造量を少なくする設計方針について述べる．また，他コアに資源操作を依頼する機能の実現方式における課題と対処について述べる．

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

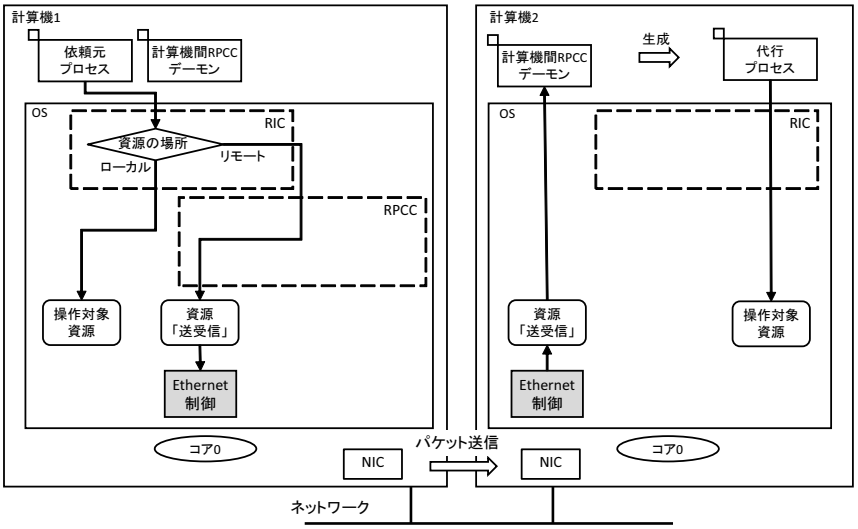


図 1 リモート計算機が管理する資源操作の様子

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では，OS が制御し管理する対象を資源と呼び，資源の分離と独立化を行っている．たとえば，既存の OS におけるプロセスを複数の資源に分割して管理している．それぞれの資源は独立化され，資源の種類ごとに管理表が存在する．また，資源を操作するプログラム（以降，プログラム部品）を資源の種類と操作内容ごとに用意している．資源の操作は，資源インタフェース制御（以降，RIC: Resource Interface Controller）を介して行われる．RIC は，資源の種類と操作内容により，呼び出すプログラム部品を決定する．

資源を識別するために，資源には資源名と資源識別子が付与されている．資源名は，資源の場所，資源の種類，および固有名を表す文字列で構成されている．資源識別子は，資源の場所，資源の種類，および通番を表す番号で構成されている．RIC への依頼において，資源の生成には資源名を利用し，生成した資源の操作には資源識別子を利用する．

2.2 遠隔手続呼出制御

2.2.1 基本構造

Tender は，RPCC により，ローカル計算機の資源とリモート計算機の資源を同じインタフェースで利用できる．リモート計算機が管理する資源操作の様子を**図 1** に示す．計算機間 RPCC デーモンは，***Tender*** の起動時に生成され，RPCC 操作依頼を受け取るデーモンプロセスである．また，代行プロセスは，リモート計算機上で代行して資源を操作するプロセスである．

RIC では，指定された資源名または資源識別子の場所を表す情報により，資源の場所を判定する．資源の場所がローカル計算機である場合，直接操作対象資源を操作する．資源の場所がリモート計算機である場合，RPCC に資源の操作を依頼する．RPCC は，他計算機との通信機能を管理する資源である資源「送受信」を利用し，他計算機に資源操作を依頼する手続き呼出しパケットを送信する．

表 1 コア間 RPCC を考慮した通信表

計算機番号	計算機名	状態	コア番号
0	tender	LOCAL_MACHINE	0
2	tender2	LOCAL_MACHINE	0
3	tender3	REMOTE_MACHINE	0
4	tender4	REMOTE_CORE	1

リモート計算機では，計算機間 RPCC デーモンが手続き呼出しパケットを受信する．計算機間 RPCC デーモンは代行プロセスを生成し，走行させる．代行プロセスにより，リモート計算機上で資源の代行操作が行われる．代行処理の実行結果は，依頼時と逆の経路で依頼元プロセスに返却される．

2.2.2 通信表

RPCC では，RIC において，通信表を利用して資源の操作の依頼先を判別する．文献 [3] において設計された，コア間 RPCC を考慮した通信表を**表 1** に示す．通信表には，計算機番号，計算機名，状態，およびコア番号が登録されている．計算機番号は，資源識別子の場所を表す番号と対応している．計算機名は，資源名の場所を表す文字列と対応している．状態は，LOCAL_MACHINE，REMOTE_MACHINE，および REMOTE_CORE の 3 つがある．それぞれ，登録されている計算機がローカル計算機，リモート計算機，および他コアであることを表す．コア番号は，資源を管理する個別型 ***Tender*** が動作するコア番号を表す．

2.2.3 資源「送受信」

資源「送受信」は，プロセスや OS に対して計算機間の

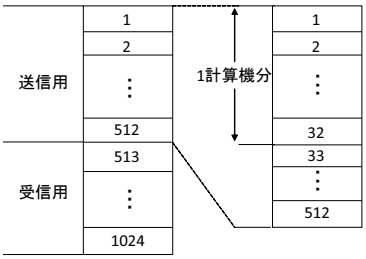


図 2 通信パス表

通信機能を提供する資源であり，指定された通信種類の入出力制御への依頼により計算機間通信を行う．資源「送受信」は送受信管理により管理される．送受信管理が提供する機能を以下に示す．

(1) 生成:資源「送受信」を生成する．このとき，通信先計算機と通信種類を指定する．通信種類として，Ethernet と Myrinet が実現されている．
(2) 削除：資源「送受信」を削除する．
(3) 送信領域と受信領域の登録：資源「送受信」に，送信領域と受信領域を登録する．
(4) 送受信相手装置のアドレス情報の登録：資源「送受信」に，通信先計算機のアドレス情報を登録する．この機能は，Ethernet による通信時に利用できる．
(5) 送信：資源「送受信」により，入出力制御にデータの送信を依頼する．入出力制御は，登録された送信領域のデータを通信先計算機に送信する．
(6) 受信：資源「送受信」により，入出力制御にデータの受信を依頼する．入出力制御は，登録された受信領域に，通信先計算機からデータを受信する．

資源「送受信」は，生成時に通信先計算機と通信種類を指定する．このため，通信先計算機と通信種類ごとに資源「送受信」を生成する必要がある．

2.3 入出力制御

入出力制御は通信種類ごとに通信機能を提供する機能であり，Ethernet 制御と Myrinet 制御が実現されている．また，各制御では，通信処理を高速化するため，送受信処理の前に通信パス表に領域を登録している [5]．

通信パス表を**図 2** に示す．通信パス表には，送信用と受信用の領域のエントリが存在し，各エントリは通信先計算機によって分割されている．一つの通信先計算機に対するエントリは図 2 では 32 個存在し，それぞれに別の領域を登録することが可能である．これにより，資源「送受信」への一度の依頼で，一つの通信先計算機に対して複数の領域の内容を送受信することができる．

通信パス表のエントリに格納される情報は，各制御で異なる．Myrinet 制御の通信パス表のエントリには，登録した領域の実アドレスが格納されている．Ethernet 制御の通信パス表のエントリには，登録した領域の実アドレスに加え，資源「仮想領域」の資源識別子と，資源「仮想領域」の先頭アドレスから登録した領域の先頭までのオフセットが格納されている．資源「仮想領域」は，実メモリあるいは外部記憶装置のデータ格納域情報を仮想化した資源である．

Ethernet 制御では，資源「仮想領域」を利用したゼロコピーを実現するため，登録する領域に対する資源「仮想領域」の情報を利用している．

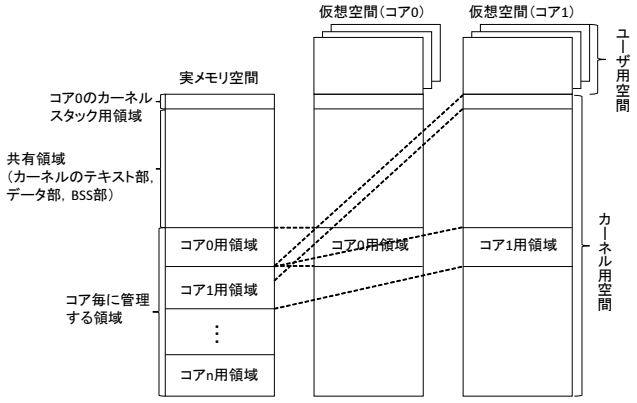


図 3 個別型 ***Tender*** における実メモリの割り当て

2.4 個別型 *Tender*

Tender では，マルチコアに対応する方式の一つとして，個別型 ***Tender*** が実現されている [3]．この方式では，各コアは異なる資源管理表を持ち，他コアが持つ資源管理表を参照しない．また，***Tender*** は，資源間の共有データを排除するように設計されている．個別型 ***Tender*** における実メモリの割り当てを**図 3** に示す．各コアのプロセスや OS が生成した資源は，コア毎に管理する領域から確保したメモリを利用する．このため，個別型 ***Tender*** では，資源を排他制御の対象とせず，並列に資源を操作することができる．

文献 [3] では，通信表を拡張し，RPCC により他コアが管理する資源を操作する基本方式のみが述べられている．しかし，コア間のデータの送受信の実現方式，およびコア間のデータの送受信を利用する RPCC と資源「送受信」は設計されていない．

3. メモリを介した遠隔手続呼出制御

3.1 設計方針

個別型 ***Tender*** において他コアが管理する資源の利用を可能にするため，RPCC を拡張し，メモリを介した RPCC を実現する．メモリを介した RPCC の設計方針を以下に示し，説明する．

（方針）***Tender*** における既存の RPCC の変更量の削減

RPCC を利用し，計算機間での資源「プロセス」の移動機能などが実現されている．実現済みの機能への影響を小さくするため，RPCC の変更量を削減する．

3.2 課題

コア間の RPCC を実現するための課題を以下に示し，説明する．

（課題 1）個別型 ***Tender*** におけるコア間通信の実現

コア間の RPCC では，資源操作依頼と資源操作結果の送受信において，コア間通信を行う必要がある．しかし，個別型 ***Tender*** ではコア間の通信機能が実現されていない．既存の RPCC では，他計算機との通信に資源「送受信」を利用している．コア間通信のために資源の種類を新規に追加して利用すると，RPCC の変更量が増加する．このため，コア間通信の実現は，既存の資源である資源「送受信」の拡張により行う．また，現在 Ethernet 制御と Myrinet 制御が実現されている入出力制御に対し，コア間通信制御を追加する．

コア間通信を実現するための課題として，以下の課題が考えられる．

(A) データの送受信

個別型 ***Tender*** では，各コアの仮想空間には，カーネルのテキスト部，データ部，BSS 部が格納された共有領域以外には，自コアのコア毎に管理する領域のみ存在し，コア毎に管理する領域をコア間で共有しない．コア間で通信を行うため，データの送受信機能が必要である．

(B) データ送信通知

データの送受信に利用する領域に送信するデータを書き込んだ時点では，送信先コアはデータを送信したことを確認していない．送信先コアで受信処理を行うため，送信データの書き込みを行った後に，データ送信通知を送信先コアに送信する必要がある．

(C) 送信されたデータの特定

送信先コアにおける受信処理において，送信されたデータの場所を特定する必要がある．

（課題 2）送信に利用する資源の特定

既存の RPCC では，一対一の資源操作を想定し，通信対象と通信手段を ***Tender*** の起動時に静的に指定している．しかし，コア間の RPCC を実現すると，依頼先はリモート計算機と他コアの複数存在する．このため，資源操作依頼と資源操作結果の送信時に，送信先に対する通信手段を選択する必要がある．

3.3 個別型 *Tender* におけるコア間通信の実現

3.3.1 対処

（対処 1-A）メモリを介したコア間でのデータ送受信

（課題 1-A）について，コア間で共有する領域を介することにより，コア間でのデータ送受信を実現し，対処する．この対処により，既存の資源「送受信」のインタフェースを生成以外変更せずに，拡張した資源「送受信」によるコア間でのデータ送受信を実現できる．

（対処 1-B）プロセッサ間割り込みによる送信先コアへのデータ送信通知

（課題 1-B）について，プロセッサ間割り込み（以降，IPI）により送信先コアへのデータ送信通知を行うことで対処する．送信元コアは，共有領域へのデータ送信後に送信先コアに IPI を送信することで，データ送信を通知する．

（対処 1-C）共有領域の静的な確保

（課題 1-C）について，***Tender*** の起動時に共有領域を静的に確保し，コア間通信に利用することで対処する．これにより，コア間の RPCC の実現において，個別型 ***Tender*** の変更量を小さくすることが可能である．

（対処 2）通信表の拡張による資源「送受信」の特定

（課題 2）について，通信表を拡張し，資源「送受信」の識別子を登録することで対処する．これにより，RPCC は，通信表と資源名または資源識別子により資源の場所を特定し，かつ利用する資源「送受信」の識別子を特定できる．

3.3.2 メモリを介したコア間でのデータ送受信

個別型 ***Tender*** におけるコア間でのデータ送受信方法として，以下の 3 つの方式が考えられる．

（方式 1）共有領域に送信領域を確保し，共有領域で受け渡しする方式

（方式 2）コア毎に管理する領域に送信領域を確保し，共有領域で受け渡しする方式

（方式 3）コア毎に管理する領域に送信領域を確保し，仮想空間の変更により参照する方式

（方式 1）では，送信元コアは共有領域に送信領域を確保し，資源「送受信」が送信先コアにデータ送信通知を送信する．また，送信先コアの資源「送受信」は受信領域を送信領域と同じ場所に設定し，データを読み取る．（方式 1）では，データコピーが不要である．しかし，共有領域を確保する機能を提供するため，資源「送受信」のインタフェースを大幅に変更する必要がある．

（方式 2）では，送信元コアはコア毎に管理する領域に送信領域を確保し，資源「送受信」が共有領域から領域を確保する．その後，送信領域から共有領域にデータをコピーして送信先コアにデータ送信通知を送信する．送信先コアでは，資源「送受信」が共有領域からコア毎に管理する領域に確保された受信領域にデータをコピーする．（方式 2）では，2 回のメモリ間コピーが必要である．しかし，資源「送受信」のインタフェースの変更量は小さくてよい．

（方式 3）では，送信元コアはコア毎に管理する領域に送信領域を確保し，送信先コアはコア毎に管理する領域に受信領域を確保する．また，各コアの資源「送受信」は，送信領域と受信領域の間で実メモリ交換を行う．（方式 3）では，データコピーが不要であり，資源「送受信」のインタフェースの変更量が小さくてよい．しかし，実メモリ交換のオーバヘッドが存在する．

表 3　コア間通信制御のインタフェース

	形式	機能
転送データ格納域の登録	core_entry_vaddr(vmid, vaddr, entry_num)	entry_num で指定した通信パス表のエントリに，vmid と vaddr で指定された仮想空間識別子と仮想アドレスから算出した実アドレスを格納する
送信	core_premitive_send(dst_node, dst_addr, src_addr, size)	src_addr で指定した送信領域から，dst_node で指定したコアの通信先へ送信する共有領域に，size バイトデータをコピーする．また，送信先コアに IPI を送信する．
受信	core_recv(flowid, dst_num, machine_num, size)	dst_node で指定した通信元から受信する共有領域から，dst_num で指定した通信パス表のエントリの受信領域に，size バイトデータをコピーする．

また，実メモリ交換では，通信先コアの仮想領域の情報を利用する．個別型 ***Tender*** では資源管理表をコア毎に保持するため，他コアの仮想領域の情報は直接参照できない．

設計方針により，資源「送受信」のインタフェースの変更が小さく，また資源「送受信」の変更に要する工数が小さい（方式 2）を採用する．コア毎に管理する領域に確保した送信領域と受信領域は，既存の計算機間通信と同様に通信パス表で管理する．通信パス表には，Myrinet 制御と同様に実アドレスのみ格納する．受け渡しに利用する共有領域は，カーネルのデータ部に確保する．

コア間でのデータ送受信時は，まず，それぞれのコア毎に管理する領域に送信領域と受信領域を確保し，通信パス表に登録する．送信は，通信パス表に登録された送信領域から，受け渡しに利用する領域へのデータのコピーにより行う．受信は，受け渡しに利用する領域から，通信パス表に登録された受信領域へのデータのコピーにより行う．

3.3.3　IPI による送信先コアへのデータ送信通知

コア間の RPCC では，依頼元の OS が利用するコアと依頼先の OS が利用するコアは異なる．このため，依頼先の OS が利用するコアに対してプロセッサ間割り込み（以降，IPI）を送信することにより，データ送信通知を実現する．IPI を受信したコアでは，IPI を送信したコアの情報を取得できない．このため，送信元のコアによって，送信する IPI の割り込みベクタ番号を変更する．つまり，コア間通信を行うコアの数だけ，コア間通信に利用する IPI の割り込みベクタ番号を用意する．これにより，受信した IPI の割り込みベクタ番号で，IPI を送信したコアを特定することができる．

3.3.4　共有領域の静的な確保

（対処 1-B）により送信先コアにデータを通知した場合，送信先コアでは送信元コアが送信した共有領域の場所を判断できない．このため，***Tender*** の起動時に，送信先コアと送信元コアに対しての共有領域を一意に決定する．通信するコアの数を n とすると，計算機全体で n × (n-1) だけ送信に必要な共有領域を確保する．

3.4　通信表の拡張による資源「送受信」の特定

送信時に依頼する資源「送受信」を特定するため，通信先に対する資源「送受信」の資源識別子を通信表に登録する．コア間 RPCC を実現後の通信表を**表 2** に示す．依頼の送信時は，資源の場所情報により通信先を特定し，通信表により資源「送受信」を特定する．返信の送信時は，遠隔手続呼出パケットの送信元コアの情報により通信先を特定し，通信表により資源「送受信」を特定する．

表 2　コア間 RPCC を実現後の通信表

計算機番号	計算機名	状態	コア番号	送受信識別子
0	tender	LOCAL_MACHINE	0	
2	tender2	LOCAL_MACHINE	0	
3	tender3	REMOTE_MACHINE	0	0x00140001
4	tender4	REMOTE_CORE	1	0x00140002

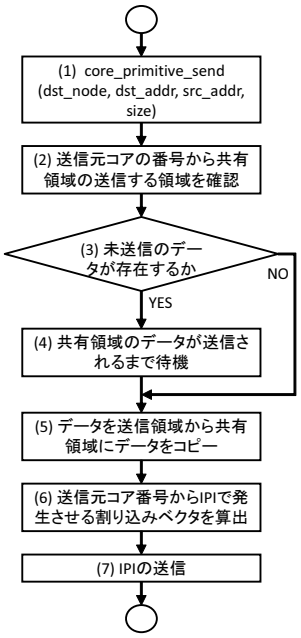
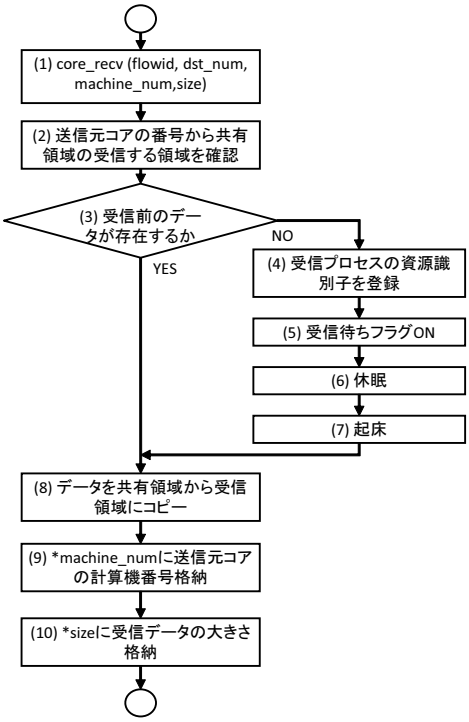


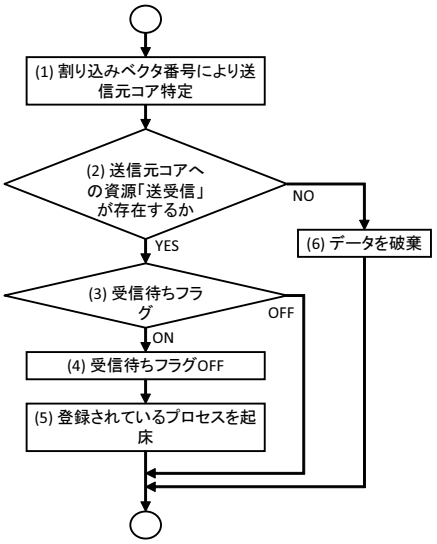
図 4　送信の処理流れ

3.5　コア間通信制御

コア間通信制御のインタフェースを**表 3** に示す．また，コア間通信制御における送信の処理流れを**図 4** に示し，処理の流れについて述べる．まず，送信元コアの番号から，共有領域の送信する領域を特定し，確認する．次に，確認した領域に未送信のデータが存在するか確認し，存在する場合は待機する．未送信のデータが送信された後で，送信領域から共有領域上の送信する領域にデータをコピーする．その後，送信元コア番号から発生させる割り込みベク



（A）資源「送受信」における受信処理



（B）IPI で発生する受信処理

図 5 受信の処理流れ

タ番号を算出し，IPI により送信先コアにデータ送信通知を送信する．

コア間通信制御における受信の処理流れを**図 5**(A) に示す．まず，共有領域の受信する領域を確認する．次に，確認した領域に受信していないデータが存在するか確認する．このとき，受信前のデータが存在しない場合は，受信プロセスの資源識別子を登録して休眠させ，受信待ちフラグを ON にする．

送信先コアが IPI を受け取ると，IPI により受信割り込みが発生する．IPI により発生する受信割り込みの処理流れを図 5(B) に示す．受信割り込みでは，まず割り込みベクタ番号によって送信元コアを特定する．次に，送信元コアを通信先に設定した資源「送受信」が存在するか，通信表を確認する．存在しない場合はデータを破棄し，割り込み処理を終了する．存在する場合は，受信待ちフラグが ON になっているか確認する．ON になっていない場合は，そのまま割り込み処理を終了する．ON になっている場合は，受信待ちフラグを OFF にし，登録されているプロセスを起床させる．

図 5(A)(7) において，資源「送受信」は，自コア宛てのデータを資源「送受信」に登録した受信領域にコピーする．その後，送信元コアの計算機番号と受信データの大きさを返却する．

3.6 呼び出しインタフェースの変更内容

提案手法を実現後の他コアへの資源操作依頼の様子を**図 6** に示し，以下に呼び出す関数のインタフェースの変更内容を説明する．

RIC に対する資源を操作するプログラムの呼び出しインタフェースと，RPCC に対する資源操作の依頼の呼び出しインタフェースは，変更しない．これは，RPCC の拡張により実現しているためである．

RPCC デーモンの生成に利用するカーネルコールのインタフェースは，引数を追加する．提案手法における RPCC デーモンの生成に利用するカーネルコールのインタフェースを**表 4** に示す．RPCC デーモンは，***Tender*** の起動時に，他コア全てとリモート計算機の数だけ生成される．従来のインタフェースからの変更点として，machine_num を引数に追加し，通信先を指定可能にしている．また，引数 kind において，コア間通信を指定可能にしている．

資源「送受信」の操作のインタフェースは，生成機能を変更する．資源「送受信」の生成は，RPCC デーモンの初期化処理において実行される．提案手法における資源「送受信」の生成のインタフェースを**表 5** に示す．各コアの RPCC デーモンは，通信先を指定して資源「送受信」を生成する．従来のインタフェースからの変更点として，引数 dev_no においてコア間通信を指定可能にしている．

資源「送受信」に対する領域の登録，送信，および受信のインタフェースは，それぞれ既存のインタフェースから

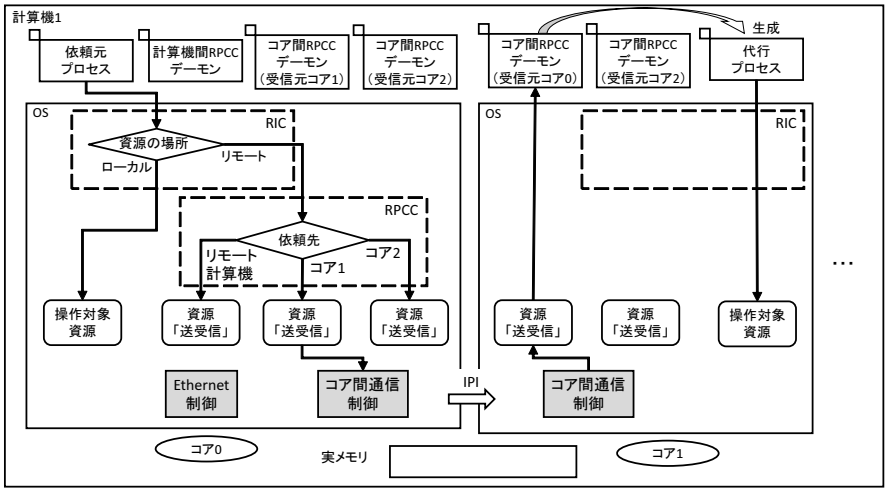


図 6　他コアへの資源操作依頼の様子

表 4　RPCC デーモンの生成に利用するカーネルコールのインタフェース

	形式	機能
変更前	rcvrpcc(rcv_num, kind, macaddr)	rcv_num の数だけ RPCC パケットを受信し，受信したパケットに応じて代行操作か資源操作結果の返却を行う．kind で通信種類として Ethernet か Myrinet を指定し，通信種類が Ethernet の場合は macaddr で通信先の MAC アドレスを指定する．通信先計算機は静的に指定されている．
変更後	rcvrpcc(rcv_num, kind, macaddr, machine_num)	rcv_num の数だけ RPCC パケットを受信し，受信したパケットに応じて代行操作か資源操作結果の返却を行う．kind で通信種類として Ethernet，Myrinet，**コア間通信**のいずれかを指定し，通信種類が Ethernet の場合は macaddr で通信先の MAC アドレスを指定する．machine_num **で通信先を指定する．**

表 5　資源「送受信」の生成のインタフェース

	形式	機能
変更前	get_sendrecv(dev_no, numofinput, numofoutput, sendrecvid)	dev_no において，送受信相手装置番号，ボード内の通信番号，ボード番号，通信種類を指定する．指定可能な通信種類は，Ethernet，Myrinet である．numofinput と numofoutput にはそれぞれ登録する送信領域と受信領域の数を指定する．sendrecvid には生成する資源「送受信」の識別子を指定する．
変更後	get_sendrecv(dev_no, numofinput, numofoutput, sendrecvid)	dev_no において，送受信相手装置番号，ボード内の通信番号，ボード番号，通信種類を指定する．指定可能な通信種類は，Ethernet，Myrinet，**およびコア間通信**である．numofinput と numofoutput にはそれぞれ登録する送信領域と受信領域の数を指定する．sendrecvid には生成する資源「送受信」の識別子を指定する．

変更しない．それぞれの機能は，生成時に通信種類で指定されている入出力制御に依頼することにより実行される．

4. 関連研究

遠隔手続呼出を利用した排他制御の削減手法の研究として，文献 [6] がある．文献 [6] では，クリティカルセクション専用のハードウェアスレッドを用意し，ロックの取得を遠隔手続呼出で行うことにより，ロック取得時間の短縮とクリティカルセクションの実行速度の向上を実現している．

分散 OS アーキテクチャの研究として，文献 [7] がある．文献 [7] では，異なる計算機や異なる命令セットのコアの環境における，OS レベルでの協調処理を提案している．

マルチコアにおける資源の管理手法の研究として，文献 [8] がある．文献 [8] では，並列アプリケーションに対して，適切なスレッド数の判断とコアへのマッピングを行っている．

5. おわりに

個別型 ***Tender*** におけるコア間 RPCC の設計について述べた．RPCC を拡張してコア間 RPCC を実現することにより，他コアの資源を自コアの資源と同じインタフェースで操作可能になる．提案手法の設計方針について，既存の RPCC の改造量を少なくすることにより，RPCC を利用した計算機間での資源「プロセス」の移動機能などへの影響を小さくすることを述べた．

データ送受信機能は，コア間での共有領域によるデータの受け渡しと，IPI による送信先コアへのデータ送信通知により実現する．また，コア間での共有領域は，送信元コ

アと送信先コアごとに静的に割り当てる．これにより，資源「送受信」のインタフェースは，生成機能を改造するだけで，他の機能を改造せずにコア間通信制御を実現可能である．

また，通信に利用する資源「送受信」を特定するために，通信表を拡張し，それぞれの通信先に対する資源「送受信」の資源識別子を登録する．これにより，依頼先への手続き呼出しパケットの送信や依頼元への呼出し結果パケットの送信において，通信先に応じた資源「送受信」が選択可能である．

残された課題として，コア間の RPCC の実現と評価がある．

参考文献

[1] Boyd-Wickizer，S.，Kaashoek，M. F.，Morris，R. and Zeldovich，N.: Non-scalable locks are dangerous，Proceedings of the Linux Symposium (2012).
[2] 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏：資源の独立化機構による ***Tender*** オペレーティングシステム，情報処理学会論文誌， Vol. 41, No. 12, pp. 3363–3374 (2000).
[3] 堀井基史，山内利宏，谷口秀夫：***Tender*** におけるコアごとに資源を用意し個別に管理する OS 構造の設計，情報処理学会研究報告，Vol. 2014-OS-131, No. 7, pp. 1–8 (2014).
[4] 石井陽介，谷口秀夫：位置透過な資源操作方式によるプロセス生成機構，情報処理学会論文誌コンピューティングシステム (ACS)，Vol. 44，No. SIG 10(ACS 2)，pp. 62–75 (2003).
[5] 下崎誠，中島耕太，谷口秀夫，牛島和夫：***Tender*** オペレーティングシステムにおける通信機構，情報処理学会第 60 回全国大会講演論文集，Vol. 2000，No.1，pp. 7 – 8 (2000).
[6] J.-P. Lozi，F. David，G. Thomas，J. Lawall，and G. Muller：Fast and Portable Locking for Multicore Architectures，ACM Transactions on Computer Systems，Vol.33，No. 4，pp. 1–62 (2016).
[7] Pierre Olivier，Sang-Hoon Kim，Binoy Ravindran：OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter，HotOS ’17 Proceedings of the 16th Workshop on Hot Topics in Operating Systems，pp. 174 – 179 (2017).
[8] Daniel Olsen，Iraklis Anagnostopoulos：Performance-Aware Resource Management of Multi-Threaded Applications on Many-Core Systems，GLSVLSI ’17 Proceedings of the on Great Lakes Symposium on VLSI 2017，pp.119 – 124 (2017).