計算機アーキテクチャ 119－26
（1996．8．28）

光バスクラスタシステム対応の分散タスク／スレッド機構の実現

鈴木 茂夫†,☆　福井 俊之†,☆　数藤 義明†　柴山 茂樹†,☆

我々は，複数台の計算機を光ファイバケーブルで結合した「光バスクラスタシステム」の研究を行なっている．本稿では，光バスクラスタの特徴である分散共有メモリを活用し，並列処理／協調分散処理アプリケーションを効率よく動作させることを目的としたシステムソフトウェアの実現について報告する．前述のようなアプリケーションのプログラム実行環境として分散タスク／スレッドモデルを採用して，そのメカニズムを光バスクラスタの一次試作機Euphoria上で実現した．そして，実際にアプリケーションを分散タスク／スレッドを用いて作成し，その動作テストを行なうことにより，分散タスク／スレッド機構の正常動作，そして並列処理や協調分散処理に対する本モデルの適用性を確認した．

Design and Implementation of Distributed Task／Thread Mechanism for Optical Bus Computer Cluster

SHIGEO SUZUKI†,☆　TOSHIYUKI FUKUI†,☆　YOSHIAKI SUDOU†
and SHIGEKI SHIBAYAMA†,☆

This paper presents an overview of the system software designed and implemented for an Optical Bus Computer Cluster system 'Euphoria', which consists of nodes (workstations) connected to each other by optical fiber links. This software aims to execute parallel／distributed processing applications efficiently using the distributed shared memory on the prototype hardware. We adopted a distributed task／thread model for the program execution environment and implemented its mechanism on the hardware. We have confirmed that the mechanism is operational and that the model is applicable to parallel／distributed processing as we intended by actually running several tests on the system.

1. はじめに

キヤノン情報メディア研究所では，複数台のワークステーションクラスの計算機を光ファイバケーブルで結合した形態のコンピュータシステムの研究を行なっている．本システムを我々は光バスクラスタ[1),2)]と呼んでいる．光バスクラスタは，従来のイーサネットなどのネットワークとは異なり，計算機の内部バスに匹敵する帯域幅を持つ光ファイバケーブルを活用することで，より高度な並列分散処理を目指している．

本稿では，光バスクラスタの特徴であるハードウェアによる分散共有メモリ機構を活用し，並列処理／協調分散処理アプリケーションを効率よく動作させることを目的としたシステムソフトウェアの実現について報告する．まず，光バスクラスタの一次試作機であるEuphoria[1),2)]の外部仕様について簡単に説明した後，前述のようなアプリケーションのプログラム実行環境として採用した分散タスク／スレッドモデル[3)]について説明する．その上で，Euphoria上で実装した分散タスク／スレッド機構の実現方式，およびシステム動作の確認結果について述べる．

†キヤノン株式会社情報メディア研究所
Media Technology Laboratory, Canon Inc.
☆現在，キヤノン株式会社CyberMediaプロジェクト
CyberMedia Project, Canon Inc.

2. 光バスクラスタEuphoriaの概要

光バスクラスタは，独立して動作可能な複数台の計算機ノード（以下ではノードと呼ぶ）を，光回線（波長多重データ回線，アービトレーション回線）によりスター型に接続したクラスタシステムである．スター型の中心には，回線の分配・調停を行なうコンセントレータが設置される．

本システムの特徴は，一般のコンピュータネットワークにおけるデータパケット交換レベルでのノード間接続とは異なり，ノード間通信プロトコルをノード内部のバスプロトコルを基本とするハードウェアに近いレベルに設定している点にある．これにより，クラスタ内の各ノード上のメモリが，すべてのノード上のCPUから同一の方法でアクセス可能な構成となり，分散共有メモリが実現できる．

光バスクラスタの一次試作機Euphoria[1),2)]の構成を述べる．各ノードは，PowerPC601（66MHz）を2個，メモリを128MB実装し，単体でも2プロセッサのマルチプロセッサワークステーションとして動作可能である．クラスタシステム全体としては，ノード数の上限を7とし，アドレス空間を**図1**に示すように割り当てている．ノード数の上限は固定ではなく，本試作用に設定した値である．

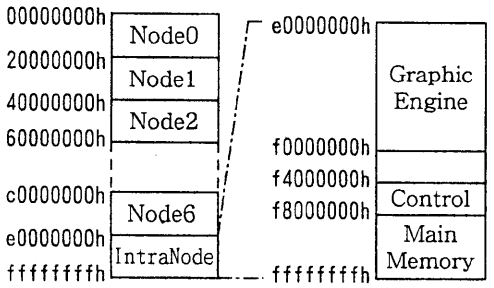

図1　Euphoriaアドレスマップ

3. 分散タスク／スレッドモデル

一方，当研究所では，光バスクラスタの研究開発と並行して，ソフトウェアによる分散共有メモリをベースとした並列分散処理の研究も進めている．このシステムをPARSEC (PARallel Software Environment for workstation Cluster)[3] と呼ぶ．PARSECでは，ソフトウェア分散共有メモリを活用したプログラム実行環境として，分散タスク／スレッドモデルを提案し，分散スレッドスケジューリングの研究を進めている．

分散タスクとは，スレッドの実行環境であるタスクと同様のものが，**図2**のようにあたかも複数のノード上にまたがって存在しているように扱える実行環境のことを指す．この分散タスク内で動作するスレッドが分散スレッドであり，複数のノード上に分散配置される．これらを各ノード上のCPUで並列実行させることで，複数台のノード上CPU資源を有効活用した並列処理が実現可能となる．

また，分散タスクは動的な拡張，圧縮が可能である．拡張とは，他のノードに分散タスクが広がることを指し，圧縮はその逆である．拡張，圧縮は，通常，負荷分散を図るためシステムにより自動的に行なわれる．しかし，ユーザが望めば，ユーザプログラム自身が指定したノードに対して，自らの分散タスクを拡張，圧縮することも可能である．また，分散スレッドを利用して実現されるユーザレベルスレッドのノード間移動

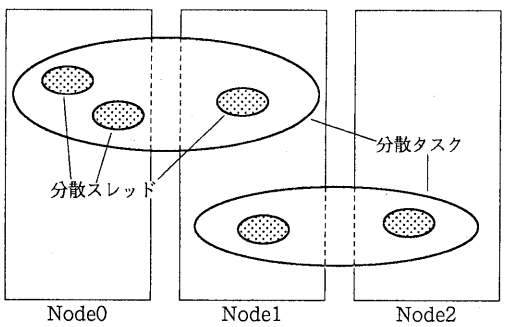

図2　分散タスク／スレッド

(第4.5節で後述) に関しても，ユーザプログラムが移動を明示的に指示することが可能である．こうした機構を活用することで，分散タスク／スレッドは，分散ファイルシステムや分散データベースといった分散システムの実現においても有用なモデルとなる．

例えば分散データベースなどの場合，データベースサーバを分散タスクで実現すると，一つのタスク環境内でありながら，データを分散配置して管理することができる．そして，各ノード上のクライアントからアクセス要求が起これば，**図3**のようにそれぞれのノードにサーバを拡張し，分散スレッドを起動することで要求を直接，そして並行に受け付けることが可能となる．要求を受け付けた各分散スレッドが共有データへアクセスする場合には，通常のマルチスレッドプログラミングと同様に，mutexロックなどのロック機構による排他制御を行なうことで，直接共有データをアクセスすることができる．

今回，光バスクラスタのハードウェアによる分散共有メモリ機構を有効に活用する並列分散処理環境の実現を目指し，この分散タスク／スレッドモデルを光バスクラスタにも適用することにした．

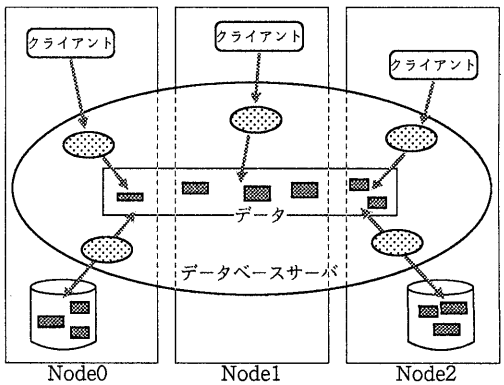

図3　分散システムの実現例

4. システムソフトウェアの実現

4.1 全体構成

本システムの構成を**図4**に示す．マイクロカーネル (MK) 上に，機能ごとに役割り分担したサーバ群が協調動作し，分散タスク／スレッド環境を実現する．MKとしては，ユーザタスクレベルで外部ページャを作成できるMach3.0MK (CMUが開発) を採用した．このMK上で外部ページャとして動作するPDSM manager (Physical Distributed Shared Memory manager) が，ハードウェアによる分散共有メモリ機構を用いて，異なるノード上のタスク間でのメモリ共有を実現する．PDSM managerと協調して動作するDtask manager (Distributed task manager) が，異なるノード上の複数のタスクを一つの分散タスクと

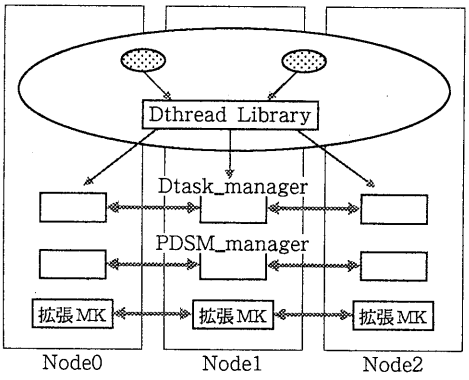

図4　システム構成

して扱える環境を実現する．そして，分散タスク内で動作するDthread Libraryが，ユーザレベルスレッドであるアクティビティの生成，移動などのユーザレベルでのスケジューリングを行なう．

4.2　マイクロカーネルの拡張

前述したようにPDSM managerはハードウェアによる分散共有メモリ機構を利用する．これを可能とするためには，メモリ管理を行なうMKの拡張が必要となる．そこで，ローレベルなメモリ共有機構を以下のインタフェースでMKに付加した．

・dsm_allocate
・dsm_refer

簡単には，dsm_allocateによって，あるタスクのメモリ領域を共有メモリ領域として登録しKEYを得る．別のあるタスクは，そのKEYをもとにdsm_referによって，自タスクの空間に共有メモリ領域をマップする．

この実現のため，まず，異なるノード上のMK間でのデータ交換と同期を可能とするMK間コミュニケーション機構を実現した（**図5** 参照）．データ交換は，それぞれのMK管理下のある固定物理メモリ領域を，互いのMK内の仮想メモリ空間にマップすることで可能とした．また，通信先MKへの非同期呼出しは，ハードウェアで実装されているノードに跨がるCPU間割込み機構を利用して実現した．次に，これらのMK間コミュニケーション機構の上に，Mach3.0MKの仮想記憶機構を拡張して，DSM管理機構を実現した．

dsm_allocateは，まず指定したタスクの仮想アドレス空間に指定量の領域を確保し，その領域に対して物理メモリをマップする．次にそれら物理メモリ情報をDSM管理機構に登録する．DSM管理機構は，登録した共有メモリを一意に特定するKEYを生成し，それを返す．dsm_referは，まず指定したタスクの仮想アドレス空間に領域を確保し，次に確保した領域に対して，KEYにより特定される共有メモリをマップする．この際，DSM管理機構に対して，KEYを引数に共有

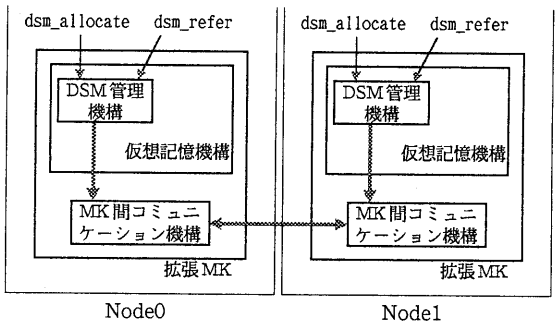

図5　MKの拡張

メモリ情報の獲得を要求する．この要求を受けたDSM管理機構は，KEYにより特定される共有メモリを管理する他ノード上のDSM管理機構と通信して，登録されている共有メモリの物理アドレス情報などを獲得する．この通信の際，MK間コミュニケーション機構を利用する．

4.3　PDSM manager

PDSM managerは，前述のローレベルなメモリ共有機構を利用することで，異なるノード上の複数のタスクが一つのメモリ領域を共有できるようにする．現在の仕様では，実行効率を考慮して，テキスト領域とRead Onlyのデータ領域については，各ノード上にコピーし，Read／Writeのデータ領域だけを共有するようにしている．また，スレッドの実行のためのスタック領域については，各ノード上の物理メモリを割り当てている．これらの処理により，Dtask managerは，分散する複数のタスクを全体で一つのタスクのように管理することが可能となる．

上記の処理を実現するために，PDSM managerは分散タスクを構成する各タスクの外部ページャとして動作し，以下のようなデマンドページング処理を行なう．分散タスクを構成するタスク内のスレッドがページフォールトを発生すると，まず，対応するページが，分散タスクを構成する他のタスクの仮想アドレス空間（タスク空間）にすでにマップされているかを調べる．まだいずれのタスクにもマップされてない場合には，dsm_allocateにより1ページ分の共有メモリ（共有メモリページ）を確保し，そのページに対応する内容を埋め，フォールトを発生したページにマップする．また，対応するページが，すでにいずれかのタスク空間にマップされていた場合には，それに対応する共有メモリページを探し出し，dsm_referにより共有可能としてから，フォールトを発生したページにマップする（**図6** 参照）．

以上の処理を，各ノード上に動作するPDSM managerが協調動作することにより行なっている．最初にタスクが生成されたノード上のPDSM managerがHost managerとなり，それ以外がSub manager

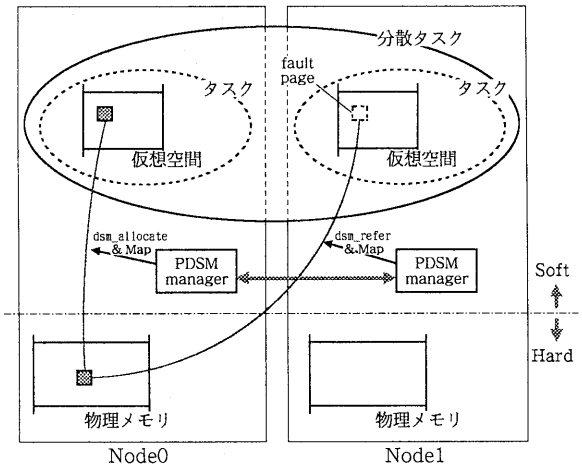

図6　PDSM managerの処理

となる．Host managerは，分散タスクの仮想アドレス空間を構成する複数のページのマッピング状況，そしてマップ済みの場合には対応する共有メモリページの物理アドレス情報などを集中管理している．Sub managerは，Host managerに問い合わせることで，指定のページのマッピング状況や共有メモリページ情報を知ることができる．

また，PDSM managerは，すでにマップ済みのページに対する共有メモリページのマッピングを変更する機能を持つ．これにより，分散タスクが共有使用する物理メモリを，ページごとに動的にノード間で切り替え可能となる．この機構は，第6節で後述する資源割当てスケジューリングで利用される．

4.4 Dtask manager

Dtask managerは，分散タスクの登録，削除，拡張，圧縮や，分散スレッドの生成，同期処理などを行なう．

分散タスクに関するインタフェースとして，

・dtask_entry

・dtask_terminate

をユーザタスクに提供する．ユーザタスクは，dtask_entryを呼び出すことにより，分散タスクとなり，他ノードへの拡張が可能となる．Dtask managerは，dtask_entryが呼び出されると，そのタスクを分散タスクとして登録し，PDSM managerを呼び出すことでタスク空間を共有領域として扱えるようにする．また，登録された分散タスクは，dtask_terminateにより削除され，消滅する．

分散スレッドに関するインタフェースは，Mach3.0のスレッドインタフェースとほぼ同様で，dthread_create，dthread_suspend，dthread_resume，dthread_set_stateなどからなる．分散スレッドの生成を行なうdthread_createに，分散スレッドの生成先ノードを指定する引数を付け加えた以外は，通常のスレッドインタフェースと同じ呼出し仕様となっている．dthread_createを呼び出すと，分散タスクの拡張が自動的に行なわれる．例えば，ノード0上で，あるタスクがdtask_entryを呼び出し分散タスクとなった後，dthread_createを呼び出すことでノード1上に新しい分散スレッドを生成しようとすると，まず分散タスクがノード1に拡張され，そのあとノード1上に分散スレッドが生成される．

Dtask managerもPDSM managerと同様に，各ノード上に分散配置され，それらが協調動作する．dtask_entryが呼び出されたDtask managerがHost managerとなり，拡張先のDtask managerがSub managerとなる．分散タスクの拡張が要求されると，Host managerは，拡張先ノード上のDtask managerと通信して，そのDtask managerをSub managerとする．Sub managerは，まず，自ノード上にタスクを生成し，次にPDSM managerを呼び出すことで，そのタスク空間の各メモリ領域の属性を拡張元タスクと同じ属性に設定し，また外部ページャの設定を行なう．これにより，分散タスクを構成する各タスクに対するデマンドページング処理が，それぞれのノード上のPDSM managerによって行なわれるようになり，タスク空間が共有され，あたかも一つのタスクのように扱うことができるようになる．

4.5 Dthread Library

Dthread Libraryは，Mach3.0のCthread Libraryを改造する形で実現した．Cthread Libraryは，Mach3.0MKのカーネルスレッドを利用して，より軽いスレッドであるユーザレベルスレッドを実現している．Dthread Libraryもそれと同様に，Dtask managerにより提供される分散スレッドを利用してユーザレベルスレッドを実現した．これをアクティビティと呼ぶ．アクティビティは，**図7**に示すように，分散スレッドへの割り当てを変更するだけで，ノード間をローコストで移動可能である．この機構は，ページマッピング変更機構と同様に，第6節で後述する資源割り当てスケジューリングで利用される．

プログラマに提供するインタフェースは，Cthread Libraryとほぼ同仕様で，dthread_fork，dthread_detach，dthread_join，dthread_exitなどからなる．同期メカニズムであるcondition変数操作やmutexロ

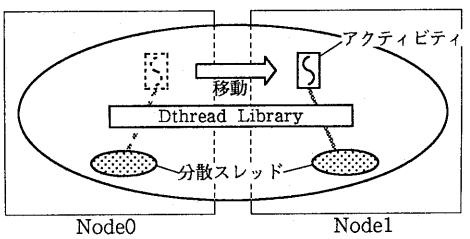

図7　アクティビティの移動

ック操作などは，Cthread Libraryと同じ仕様で利用可能である．

4.6 サーバ間通信機構

各ノード上のサーバ（PDSM manager，Dtask manager）間の通信に関しては，以下のように実現した．サーバ間通信に必要となる機構は，サーバ間でのデータ交換機構と同期機構である．データ交換に関しては，MKのローレベルなメモリ共有機構（dsm_allocate，dsm_refer）を各サーバが直接呼び出し，確保した共有メモリ領域をデータ交換領域として利用することで実現した．同期に関しては，MK間コミュニケーション機構を，MK間だけでなくユーザタスクであるサーバ間でもノード間同期が利用可能なように拡張し，各サーバがその機構を呼び出すことで実現した．

5. システム動作の確認

現在，Mach3.0MKの拡張，各サーバおよびDthread Libraryの試作が終了している．しかし，基本的なメカニズムを実現しただけで，アクティビティや共有メモリページをどのように配置するか，といった資源割当てのスケジューリングの実現はまだ行なっていない．これについては次節で後述する．現段階のシステムを，我々は光バスクラスタの一次試作機Euphoriaに実装し，その上でいくつかの分散タスクプログラムを実験的に動作させている．並列処理の評価用プログラムとして，クイックソート，迷路探索，SPLASH-2（Stanford Parallel Applications for Shared Memory）を実装し，また協調分散処理の一例として，複数のノードから多人数参加型の簡単なゲームプログラムを実現し，動作テストを行なった．しかし，現在安定動作しているシステムは，リモートメモリアクセスの際にキャッシュをオフにするという過渡的なバージョンであり，ハードウェアの性能をフルに活用した仕様ではないため，定量的な評価には至っていない．したがって，今回のテストプログラムの実装および動作テストは，実現した分散タスク／スレッド機構の動作確認，そして，並列処理，協調分散処理に対する分散タスク／スレッドモデルの適用性の確認が主な目的である．

(1) 並列処理について

本システムの利用者はDthread Libraryを利用してプログラミングを行なう．Dthread Libraryは，前述したように，機能的にMach3.0のCthread Libraryの上位互換性を保っているため，Cthread Libraryであればもちろんのこと，一般的なマルチスレッド機構を用いた並列処理プログラムであれば，容易に移植，実装することができる．例えば，SPLASH-2などもマクロファイルの書き換えのみで実装することができた．また，これらのプログラムをEuphoria上で実際に動作させたことにより，3ノード構成で最大6CPUで並列動作することも確認できた．

(2) 協調分散処理について

協調分散処理に関しては，複数のノードから多人数参加が可能なゲームプログラムを作成し動作テストすることで，分散タスク／スレッドモデルの適用性を検証した．本プログラムは，各ノードを利用する参加者がマウス操作により共有のフィールド上で陣地を取り合う簡単なものだが，各ノード上でのマウス操作を受け付け，それによって起こる変化を共有のフィールド情報に反映し，結果を各ノード上のディスプレイに表示する，といった基本的な協調分散処理が要求される．このようなプログラムをメッセージ通信ベースの環境で実現すると，(a) ある一つのプログラムが共有情報を集中管理し，各ノード上のプログラムはメッセージ通信を介して共有情報にアクセスする方式か，(b) 各ノード上のプログラムに情報を分散配置しそれらのコンシステンシ保持のためメッセージ通信を行なう方式をとるのが一般的である．いずれの方式も複数のプログラムが協調動作する形態となるためシステムが複雑になり，特に方式 (b) におけるコンシステンシ維持のための複雑な処理は，開発効率や拡張性の低下の原因となる．

分散タスク機構を利用したプログラムの基本的な構造を**図8**に示す．各ノード上に分散配置した分散スレッドが，各ユーザが行なうマウス操作を読み取り，その値にしたがって，分散タスクの共有データ領域内のフィールド情報に対してRead／Write処理を行なう．当然，このときのRead／Write処理は，ロック機構により排他的に実行される．フィールド情報が更新されると，各ノード上の分散スレッドがそれを各ノード上のディスプレイに反映する．このように，分散タスク機構を利用した場合，協調分散処理をマルチスレッドの一つのプログラムとして実現できるため，複数のプログラムで協調動作するような複雑な構成をとる必要がなくなる．また，共有データを分散タスク内共有領域で集中管理できるため，分散管理によるコンシステンシ維持といった処理も必要ない．その結果，プログラムはシンプルな構造となり，短時間で開発，デバッグすることができた．

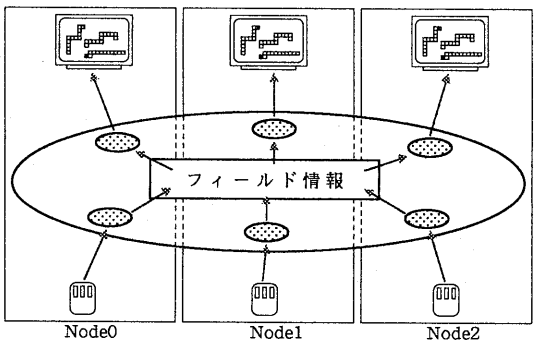

図8 協調分散処理プログラム例

6. 資源割当てスケジューリング

光バスクラスタは，ハードウェア上でグローバルなメモリをアクセスできる機構を備えるが，もちろんローカルなメモリアクセスに比べ，他ノードへのリモートアクセスにはコストがかかる．Euphoriaでのメモリリード（バースト転送時）のレスポンスタイムの実測で約7.5倍（600nsと4516ns）であった．もちろん，キャッシュの導入により，こうしたリモートアクセスの遅延をかなり隠ぺいすることは可能である．しかし，そのような場合でも，複数のノード上のCPUから共有アクセス（Read／Write）が発生すると，キャッシュコンシステンシ維持のための通信がノード間で発生し実行効率の低下を招く．

本システムでは，資源割当てスケジューリングにより，このような効率低下を最小限に押さえる．そのためにまず，動的スケジューリングを可能とするメカニズムとして，前述のページマッピング変更機構とアクティビティ移動機構を実現した．これらを利用して，以下のような処理を行なうスケジューラを実現し評価することが今後の課題となる．

アクティビティが多数生成され並列度が増すと，スケジューラは，基本的にはCPU資源を求めて分散タスクを他ノードに拡張していく．しかし，この拡張の弊害として，**図9** (a) のようにリモートアクセスが頻繁に発生し，かえって実行効率が低下してしまう場合もある．こうした状況を検知すると，ページマッピング変更機構により，リモートアクセスの原因となる共有メモリページを，**図9** (b) のように切り替えることでリモートアクセスの頻度を軽減する．また，**図10** (a) のようにノード間でのコンシステンシ維持動作が頻繁に発生するような状況を検知すると，アクティビティ移動機構とページマッピング変更機構を利用して，共有アクセスされている共有メモリページとそれをアクセスするアクティビティ群を，**図10** (b) のように一つのノード上にまとめることで実行効率の低下を押さえる．まとめ先ノードとしては，CPUやメモリ資源に余裕のあるノードや，マッピング変更やアクティビティ移動が少なくてすむノードなどを優先して選択する．

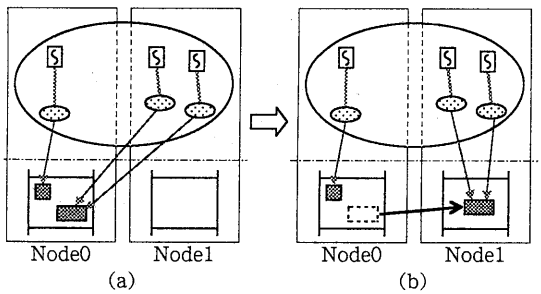

図9　リモートアクセスの頻度が高い状況

こうしたスケジューリングを実現するためには，ソフトウェアだけでなく，リモートアクセスやノード間コンシステンシ維持動作の頻度を計測するハードウェア機構が必要となる．Euphoriaでは，コンセントレータ[2]によりこうした情報を集計することが可能であるが，それを各ノード上のシステムソフトウェアに通知する機構が実装されていない．これらのハードウェア機構を実現することも今後の課題となる．

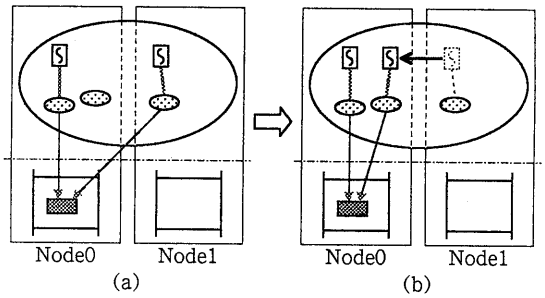

図10　コンシステンシ維持動作の頻度が高い状況

7. おわりに

本稿では，光バスクラスタの特徴である分散共有メモリ機構を活用して，並列分散処理アプリケーションを効率よく動作させることを目的としたシステムソフトウェアの実現について報告した．光バスクラスタの一次試作機Euphoriaの外部仕様，プログラム実行環境として採用した分散タスク／スレッドモデルの概要について述べた後，実現方法の詳細について述べた．

分散タスク／スレッド機構の実現のため，Mach3.0 MKの拡張，PDSM manager，Dtask manager，Dthread Libraryの試作をEuphoria上で行なった．そして，並列処理，協調分散処理を行なう分散タスクプログラムを数件作成し，実機上で動作テストしたことにより，分散タスク／スレッド機構の正常動作を確認し，また並列処理や協調分散処理に対する分散タスク／スレッドモデルの適用性を確認した．そして，今後の課題となる資源割当てスケジューリング方式とその実現方法について簡単にまとめた．

参考文献

1)福井俊之，濱口一正，下山朋彦，小杉直人，柴山茂樹：光バスクラスタ計算機Euphoriaの開発（1）概要，第49回情報処理学会全国大会5K-05（1994）．
2)福井俊之，鈴木茂夫，中村秀一，下山朋彦，数藤義明，濱口一正，柴山茂樹：光バスクラスタシステムの仕様と基本性能評価，情処研報ARC-119（1996）．
3) 数藤義明，鈴木茂夫，長健二朗，柴山茂樹：分散共有メモリ上の分散スレッド実行環境とスケジューリング，信学技報，CPSY95-63,pp.103-110（1995）．