計算機アーキテクチャ　80－11
（１９９０．１．２６）

可変構造型並列計算機の ローカル／リモート・メモリ・アーキテクチャ

甲斐康司　森 眞一郎　村上和彰　福田 晃　富田眞治
（九州大学大学院総合理工学研究科）

我々は，実行すべき並列アルゴリズムの構造に対して，柔軟に計算機構成を適応させる『可変構造型並列計算機（Reconfigurable Parallel Processor)』の開発を進めている．本システムは，128台のプロセッシング・エレメント（PE）を128×128のクロスバー網で相互結合したマルチプロセッサ・システムである．

このようなマルチプロセッサ・システムを構成するにあたっては，プロセッサ－プロセッサ間およびプロセッサ－メモリ間をいかに結合するかが，システム成功の鍵を握っている．特にプロセッサ－メモリ間結合に関しては，メモリ共有型密結合マルチプロセッサとメッセージ交換型疎結合マルチプロセッサという２つの選択肢があり，これらのいずれを選択するかが大きな課題となる．そこで，本システムでは，相互結合網およびメモリを次のように可変構造化した“ダイナミック・アーキテクチャ”を採用している．

① 可変構造型ネットワーク・アーキテクチャ：プロセッサ－プロセッサ結合およびプロセッサ－メモリ結合として任意の結合形態を提供する．

② 可変構造型メモリ・アーキテクチャ：任意のプロセッサ間でメモリの共有／非共有を可能とする．

本システムの物理的なメモリ構成自体は，各PEにローカルメモリ（容量4MB）を持たせる“分散メモリ配置”となっており，全PEにとって均等な距離に位置するグローバルメモリは存在しない．しかしながら，各ローカルメモリを自PEに単に専有させるのではなく，他PEからもクロスバー網を経由してアドレスにより直接アクセス可能とする“ローカル／リモート・アーキテクチャ（分散共有メモリ構成)”を採用している．また，そのアドレッシング機構として2レベル・アドレス変換を採用して，効率よいメモリ管理を可能としている．さらに，ローカル／リモート・アーキテクチャを基本に，プロセッサ間結合形態として密結合／疎結合マルチプロセッサ，および，プロセッサ－共有メモリ間時間距離として均一／不均一メモリ・アクセスのいずれをも，それぞれ採ることを可能にしている．

本稿では，まずメモリ・アーキテクチャの設計方針を整理したあと，本システムで採用したローカル／リモート・アーキテクチャに関して，そのアドレッシング機構と特長を述べる．また，本システムで提供するメモリ・アーキテクチャの可変構造性，および，リモート・アクセスの実現方法について述べる．

The Local／Remote Memory Architecture of the Kyushu University Reconfigurable Parallel Processor

(in Japanese)

Koji KAI, Shin-ichiro MORI, Kazuaki MURAKAMI, Akira FUKUDA, and Shinji TOMITA
Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1, Kasuga-koen, Kasuga-shi, Fukuoka, 816 Japan
e-mail : kai@kyu-is.is.kyushu-u.ac.jp

The reconfigurable parallel processor has been developed at Kyushu University. It is an MIMD-type multiprocessor which consists of 128 processing-elements (PEs) fully connected by a 128×128 crossbar network.

In general, a multiprocessor system is intensely characterized by memory and network architectures, that define inter-PE (i.e., processor-memory and／or processor-processor) paths. But such specialization is not preferable for our system, to be a general／multiple-purpose parallel processor. Our system, therfore, exploits unique reconfigurability in these architectures.

Our system employs a distributed memory organization by providing 4M bytes of local memory in each PE, and therefore there is no global memory situated at a location equally accessible to all processors. To exploit the memory reconfigurability, however, the system allows more then one processor to share a local memory. A processor of each PE can directly access the local memories of the other PEs, which memories are called “remote memories”, via the crossbar network. The memory architecture is referred to as “local／remote architecture”.

In this paper, we describe the design philosophy and the local／remote memory architecture, employed in our system, and its implementation.

1. はじめに

我々は，実行すべき並列アルゴリズムの構造に対して，柔軟に計算機構成を適応させる『可変構造型並列計算機(Reconfigurable Parallel Processor)』の開発を進めている．[1]～[3] 本システムは，128台のプロセッシング・エレメント(PE)を128×128のクロスバー網で相互結合したマルチプロセッサ・システムである．

このようなマルチプロセッサ・システムを構成するにあたっては，プロセッサ－プロセッサ間およびプロセッサ－メモリ間をいかに結合するかが，システム成功の鍵を握っている．特にプロセッサ－メモリ間結合に関しては，メモリ共有型密結合マルチプロセッサとメッセージ交換型疎結合マルチプロセッサという2つの選択肢があり，これらのいずれを選択するかが大きな課題となる．そこで，本システムでは，相互結合網およびメモリを次のように可変構造化した“ダイナミック・アーキテクチャ”を採用している．[1] [2]

① 可変構造型ネットワーク・アーキテクチャ：プロセッサ－プロセッサ結合およびプロセッサ－メモリ結合として任意の結合形態を提供する．

② 可変構造型メモリ・アーキテクチャ：任意のプロセッサ間でメモリの共有／非共有を可能とする．

本システムの物理的なメモリ構成自体は，各PEにローカルメモリ（容量4MB）を持たせる“分散メモリ配置”となっており，全PEにとって均等な距離に位置するグローバルメモリは存在しない．しかしながら，各ローカルメモリを自PEに単に専有させるのではなく，他PEからもクロスバー網を経由してアドレスにより直接アクセス可能とする“ローカル／リモート・アーキテクチャ（分散共有メモリ構成）”を採用している．[3] [5] あるPEのプロセッサから見たとき，クロスバー網を経由してアドレスにより直接アクセス可能な他PEのローカルメモリを“リモートメモリ”と呼ぶ．また，クロスバー網を経由するメモリ・アクセスを“リモート・アクセス”と呼ぶ．

また，本システムのクロスバー網は回線交換方式を採用しており，回線接続要求の競合に対して，“デマンド・モード”と“プリセット・モード”と呼ぶ2種類の調停法を提供している．デマンド・モードでは通常の共有バス同様，動的にリモート・アクセス競合を調停する．一方，プリセット・モードでは，プログラム実行前（コンパイル時）にリモート・アクセス競合の調停を済ませるので，実行時には競合が生じ得ない．しかし，回線接続パターンを時分割により変えて行くので，[4] 1個のメモリ・トランザクションのアトミック性を保証する必要がある．

本稿では，まずメモリ・アーキテクチャの設計方針を整理したあと，本システムで採用したローカル／リモート・アーキテクチャに関して，そのアドレッシング機構と特長を述べる．また，本システムで提供するメモリ・アーキテクチャの可変構造性，および，リモート・アクセスの実現方法について述べる．

2. 設計方針

2.1 メモリ・アーキテクチャに関するパラダイム

マルチプロセッサ・システムのメモリ・アーキテクチャに関しては，様々なパラダイムが存在する．[1] [2] ここでは，これらを以下の観点から整理する．

(1) メモリ配置形態

メモリをシステムのどこに配置するかに関して，以下の2つの形態に大きく分かれる．

① 集中メモリ配置：プロセッサとは別個にメモリだけをまとめて配置する（図1（a）参照）．

② 分散メモリ配置：メモリを個々のプロセッサに分散して配置する（図1（b）参照）．

これらを併用したメモリ配置形態も可能である．

(2) 対プロセッサ空間距離

あるメモリから見た場合のプロセッサとの間の物理的な距離に関して，メモリ自身は以下の2種類に分類される．

① グローバルメモリ：そのメモリから見て全プロセッサが均等な距離に位置する．集中メモリ配置におけるメモリに相当する（図1（a）参照）．

② ローカルメモリ：そのメモリから見て特定の（1個あるいは複数個）プロセッサが他のプロセッサより近距離に位置する．分散メモリ配置において，各プロセッサに分散配置されたメモリに相当する（図1（b）参照）．

ある1つのメモリは，グローバルメモリかローカルメモリのいずれかに排他的に分類される（ただし，例外はある；クラスタ構成におけるメモリ，など）．

(3) メモリ所有形態

あるメモリから見て，それを所有するプロセッサ数の単複に応じて，メモリ自身はさらに以下の2種類に分類される．

① 専有メモリ：1個のプロセッサに専有されている．

② 共有メモリ：複数（全部あるいは一部）のプロセッサに共有されている．

先のグローバルメモリは，必然的に共有メモリである．一方，ローカルメモリは共有および専有いずれの所有形態も可能である（ローカルメモリ≡専有メモリとすることが多いが，これは正しくない）．

(4) プロセッサ間結合形態

ある2個以上のプロセッサの結合形態に関して，以下の2つの形態に大きく分かれる．

① 密結合マルチプロセッサ（TCMP）：ある2個以上のプロセッサが共有メモリを介して結合されている．よって，メモリ共有型密結合マルチプロセッサともいう．データを移動させることなく，プロセッサ間でデータの受渡しを行える．

② 疎結合マルチプロセッサ（LCMP）：ある2個以上のプロセッサが共有メモリを介しては結合されていない．プロセッサ間でデータの受渡しを行う場合は，これをメッセージとして実際に移動させなければならない．よって，メッセージ交換型疎結合マルチプロセッサともいう．

これらを併用したプロセッサ間結合形態も可能である．なお，このプロセッサ間結合形態は（1）のメモリ配置形態とは完全に直交関係にあるので，これらの4通りの組合せはいずれも可能である（密結合≡集中メモリ配置，および，疎結合≡分散メモリ配置とすることが多いが，これは正しくない）．

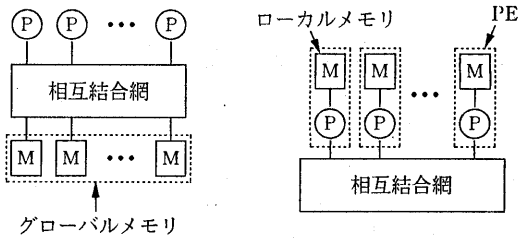
(5) 対プロセッサ時間距離

ある共有メモリから見た場合のプロセッサからのアクセス時間に関して，メモリ・アクセスは以下の2種類に分類される．

① 均一メモリ・アクセス（UMA：Uniform Memory Access）：どのプロセッサからも同じアクセス時間で，その共有メモリにアクセスできる．

② 不均一メモリ・アクセス（NUMA：Non Uniform Memory Access）：プロセッサによって，その共有メモリへのアクセス時間が異なる．

集中メモリ構成におけるグローバルメモリへのアクセスは，必然的に均一メモリ・アクセスとなる．一方，分散メモリ構成においてローカルメモリを共有メモリとする場合，それへのアクセスは均一および不均一のいずれにもなり得る．



(a) 集中メモリ配置　　(b) 分散メモリ配置

図1. メモリ配置形態

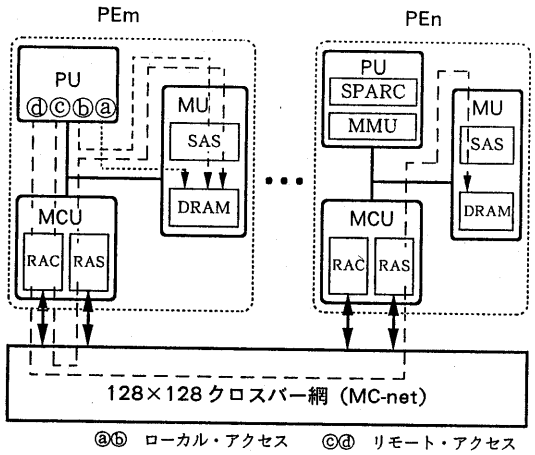


図2. 可変構造型並列計算機のシステム構成

2.2 メモリの可変構造化

メモリの可変構造化にあたり，2. 1節で述べた各パラダイムに対して以下の設計方針を採った．

① メモリ配置形態：分散メモリ配置のみとする．
② 対プロセッサ空間距離：ローカルメモリのみで，グローバルメモリは設けない．
③ メモリ所有形態：各プロセッサがそのローカルメモリを専有することも，あるいは，複数プロセッサで共有することも許す．
④ プロセッサ間結合形態：密結合および疎結合のいずれも許す．
⑤ 対プロセッサ時間距離：ローカルメモリを共有する場合，それへのアクセスは均一および不均一のいずれも許す．

これは，実行すべきプログラムにとって意味があるのは，それが直接アクセスできる記憶空間の構成，および，そのアクセス時間であるからである．したがって，①と②は固定構造であるが，③④および⑤に関しては可変構造としている．以下，③については3章で，④と⑤については4章でその実現方法を詳述する．

3. ローカル／リモート・アーキテクチャ

3.1 原理

本システムでは，図2に示すように，各PEに4MBのローカルメモリを分散させる分散メモリ配置を採る．このとき，各ローカル・メモリをそれぞれのPEに単に専有させるのではなく，複数PE間での共有も可能にしている．[5]

このように，分散メモリ配置を採るマルチプロセッサ・システムにおいて，複数プロセッサ間でのローカルメモリ共有を許す方式を“ローカル／リモート・アーキテクチャ”[9]あるいは“分散共有メモリ（distributed shared-memory）構成”と呼ぶ（“分散グローバルメモリ（distributed global-memory）構成”と呼ぶこともあるが，これは2. 1節のグローバルメモリの定義に馴染まない）．

ローカル／リモート・アーキテクチャでは，図2に示すように，次の2種類のメモリ・アクセス法がある．

① ローカル・アクセス：相互結合網を経由せずにメモリにアクセスする（図2のⓐとⓑ）．
② リモート・アクセス：相互結合網を経由してメモリにアクセスする（図2のⓒとⓓ）．

なお，あるPEのプロセッサから見たとき，リモート・アクセス可能な他PEのローカルメモリを“リモートメモリ”と呼ぶ．

3.2 アドレッシング機構

ローカル／リモート・アーキテクチャの実現方法には，3. 4節で述べるいくつかの方式が考えられる．本システムでは，仮想／実／物理の3種類のアドレスを導入した2レベル・アドレス変換によりこれを実現する（図3参照）．[4] [5] このアドレス変換過程は，以下のようになる．

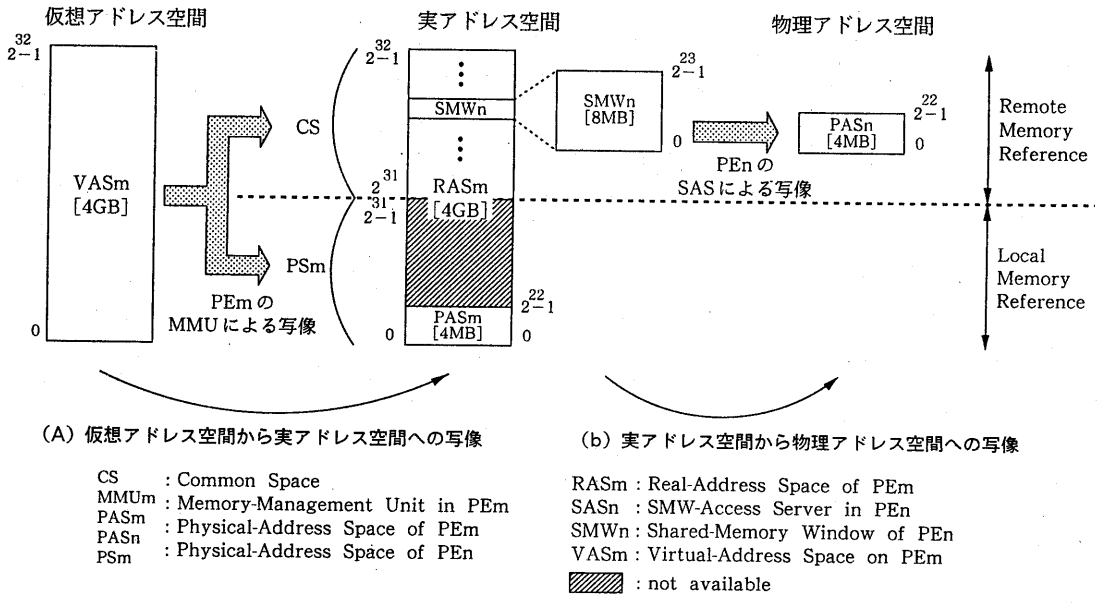


図3. アドレス空間の写像

(1) 仮想→実アドレス変換（第1レベル・アドレス変換）

プロセッサ・ユニット（PU）が出力する32ビット仮想アドレスをメモリ管理ユニット（MMU）のページング機構により32ビット実アドレスに変換する．この4Gバイトの実アドレス空間を次の2つの空間に分割する．

① プライベート空間：下位2Gバイトの空間であり，それぞれのPEの私有領域である．実アドレスがプライート空間を指定していれば，第2レベルのアドレス変換を経ずに，実アドレスを物理アドレスとして直接自ローカルメモリにアクセスする．

② コモン空間：上位2Gバイトの空間であり，全PEの共通領域である．コモン空間は，各PE対応に256個の“共有メモリ・ウインド（SMW）”に分割される（現在SMW128～255は未使用）．個々のSMWの容量は8Mバイトである．このSMWにアクセス（これを“SMWアクセス”と呼ぶ）しようとすると，当該SMWに対応するPEのローカルメモリ（すなわち，リモートメモリ）へクロスバー網を介してリモート・アクセスすることになる．

ページ・フォールトが発生した場合，それは当該メモリ・アクセスを行おうとしたPUに通知される．

(2) 実→物理アドレス変換（第2レベル・アドレス変換）

SMWアクセスが起動されると，当該SMWを提供するPEにおいて，SMWアクセス・サーバ（SAS）が実アドレスをページングにより物理アドレスに変換してメモリ・アクセスを遂行する．ページ・フォールトが発生した場合，それは当該メモリ・アクセスを行おうとしたPU，および，当該SMWを提供しているPEのPUの双方に通知される．

3.3 OSのメモリ管理機構

2レベル・アドレス変換を導入したことで，OSのメモリ管理機構に対して，以下の2つの要件が新たに生じる．

① 第2レベルのアドレス変換テーブルの管理

② 第2レベル・アドレス変換で発生したページ・フォールトの処理

我々が現在開発中の並列／分散OS [16] [17] において，メモリ管理機構が上記要件に具体的にどう対処しているかについては，参考文献［18］を参照されたい．以下，上記要件に関する一般的な対処法を述べる．

(1) アドレス変換テーブル管理

第1レベルおよび第2レベルともにページングを採用しており，それぞれは以下の独立したアドレス変換テーブルを用いる．

① 第1レベル・アドレス変換テーブル：仮想→実アドレス変換は2レベル・ページングとなっており，セクション・テーブルとページ・テーブルと呼ぶ2種類のアドレス変換テーブルを用いる．セクション・サイズおよびページ・サイズは，それぞれ1Mバイトおよび4Kバイトである．これらのテーブルはメモリ・アクセスする側のPEの実アドレス空間に置かれる．

② 第2レベル・アドレス変換テーブル：個々のSMWに対する実→物理アドレス変換は1レベル・ページングとなっており，SMW変換テーブルと呼ぶアドレス変換テーブルを用いる．ページ・サイズは第1レベル同様4Kバイトである．このテーブルは，当該SMWを提供するPEのI／O空間のひとつ（物理的にはSAS内の専用メモリ）に置かれる．

まず，セクション・テーブルおよびページ・テーブルの管理は通常の仮想記憶OSのそれと同様である．一方，SMW変換テーブルの管理が，通常のOSにはないものである．しかし，その管理自体は，セクション・テーブルおよびページ・テーブルの管理と基本的に変わらない．しかも，テーブルの格納場所を予め専用メモリにとってあるので，他のテーブルのようにその格納領域を随時確保する必要がない．

(2) ページ・フォールト処理

3.2節で述べたように，2レベル・アドレス変換では次の2種類のページ・フォールトが発生する可能性がある．

① 第1レベル・ページフォールト：仮想→実アドレス変換で生じるページ・フォールトである．これは正確には，セクション・フォールトとページ・フォールトに分かれる．いずれも，当該メモリ・アクセスを行おうとしたPUに通知される．

② 第2レベル・ページフォールト：実→物理アドレス変換で生じるページ・フォールトである．SMWページ・フォールトと呼ぶ．これは，当該メモリ・アクセスを行おうとしたPU，および，当該SMWを提供しているPEのPUの双方に通知される．

まず，セクション・フォールトおよびページ・フォールトの処理は通常の仮想記憶OSのそれと同様である．ただし，コモン空間のページ・リプレースメントにおいては，当該ページが複数仮想アドレス空間間で共有されている可能性があるので，ページ・テーブルの一貫性を保証するよう注意が必要である．[9]

一方，SMWページ・フォールトの処理が，通常のOSにはないものである．この処理は，当該SMWページ・フォールトを受け取ったPUの立場により，次のように異なる．

① 当該メモリ・アクセスを行おうとしたPU：当該メモリ・アクセスを要求したプログラムの実行（OSの管理上は，プロセスないしタスク）をサスペンドする．そのあと，別のプロセスをディスパッチするか，あるいは，ウェイトするかはOS次第である．

② 当該SMWを提供しているPEのPU：通常のページ・フォールトの処理同様，ページ・リプレースメントを行う．ページ・インが完了したら，その旨を当該メモリ・アクセスを行おうとしたPUに通知する．

SMWページ・フォールト処理の一連の流れを以下にまとめる（図4参照）．なお，下記の番号の右肩に「'」が付いたものは，当該SMWを提供しているPE（PEn）側の処理である．

① PEmがPEnの提供するSWMへメモリ・アクセスを行ったところ，SMWページ・フォールトが発生した．

② SMWページ・フォールトをPEmへ当該メモリ・アクセスの結果として通知する．これ以降のPEm側の処理は，先に述べた通りである．

②' 同時に，SMWページ・フォールトは，PEnのプロセッサ（PU）へ割込みとして通知される．

③' PEn側のメモリ管理機構により，ページ・リプレースメントを行う．

④' ページ・インが完了したら，その旨をPEmに（割込み等で）通知する．

⑤ PEmは，当該SMWページへ再アクセスする．

3.4 他の実現方式との比較

ローカル／リモート・アーキテクチャを採用したマルチプロセッサ・システムとしては，米CMUのCm* [11]～[13]，米IBMワトソン研のRP3，[14] [15] 米BBN社のButterfly，などがある．いずれも，そのアドレッシング機構が異なる．以下，RP3，Cm*，および，本システムのアドレッシング機構について，アドレス変換のレベルおよび変換場所に注目して比較する．

① RP3：1レベル・アドレス変換である．すなわち，アクセスする側のPME（Processor-Memory Element；本システムのPEに相当）において，仮想→実アドレス変換（2レベル・ページング）を行い，その実アドレスでそのまま物理メモリにアクセスする（なお，実際には実アドレスをさらにインタリーブして，その結果の絶対アドレスを用いているが，このインタリーブはここでいうアドレス変換には相当しない）．

② Cm*：2レベル・アドレス変換である．第1レベル・アドレス変換はアクセスする側のCM（Computer Module；本システムのPEに相当）において，仮想アドレスを物理アドレス，あるいは，システム仮想アドレス（本システムにおけるコモン空間の実アドレスに相当）に変換する（1レベル・ページング）．システム仮想アドレスはさらに，物理アドレスに変換される（セグメンテーション方式）．この第2レベル・アドレス変換は，アクセスされる側のCM（正確には，CMのクラスタ毎に設けられているKmapと呼ばれる部

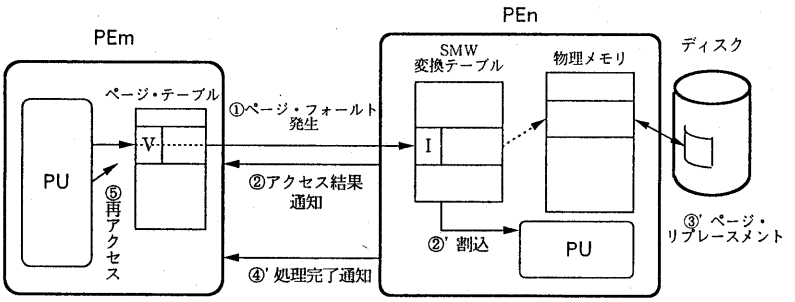


図4. SMWページ・フォールト処理

分）において行う.

③ 本システム：2レベル・アドレス変換である. Cm*同様, 第1レベル・アドレス変換はアクセスする側のPEで, 第2レベル・アドレス変換はアクセスされる側のPEで行う.

以上の相違は, ページ・リプレースメント時の処理量に影響を与えることがわかる. すなわち, RP3のような1レベル・アドレス変換の場合, 物理メモリ上のページ・リプレースメントの度に, 当該ページを含む全ページ・テーブルの更新を行う必要がある.

一方, 本システム（あるいは, Cm*）のような2レベル・アドレス変換においては, 唯一, 第2レベル・アドレス変換テーブルのみを更新すればよい. このページ・リプレースメント処理は, アクセスされる側のPE内で閉じて行われ, 第1レベル・アドレス変換テーブルのいずれにも変更を与えない. ただし, コモン空間（Cm*では, システム仮想アドレス空間）上でページ・リプレースメントを行う場合は, 当該ページを含む全ページ・テーブルの更新を行う必要がある. しかし, コモン空間の大きさ（2Gバイト）と物理メモリ容量（512Mバイト）を比較すると, 物理メモリ上でのページ・リプレースメントの頻度が大きくなることが予想される. よって, 物理メモリ上のページ・リプレースメントの処理量を抑えることは, 極めて重要な要件である.

4. 可変構造型メモリ・アーキテクチャ

ローカル／リモートアーキテクチャを採用したことで, 分散メモリ配置されたローカル・メモリのプロセッサ間での共有が可能となった. そこで, これら共有メモリへのアクセスを制御することで, 可変構造型メモリ・アーキテクチャを実現する.

4.1 密結合／疎結合マルチプロセッサ

3. 1節で述べたように, 実アドレス空間中のコモン空間の1つのSMWへアクセス（SMWアクセス）すると, 共有メモリへのアクセスが行われる. よって, このSMWアクセスの可／不可を各PE毎にSMWページ単位で制御することで, システム全体として任意の形態の共有メモリが実現される. そのために, 各PEのMMU内に, SMWページ単位のアクセス可／不可を指定する256Kビット（＝1Gバイト／4Kバイト）のビットマップ・テーブルを設けた. [2]

システム全体で合計128個あるこのビットマップ・テーブルの設定如何で, 以下のプロセッサ間結合形態のいずれをも採ることが可能となる.

① 密結合マルチプロセッサ：図5（a）に示すように, 全PEにおいてすべてのSMWページへのアクセスを許可し, プライベート空間へのアクセスを行わない場合, 本システムはメモリ共有型密結合マルチプロセッサとなる. ただし, プライベート空間のページ0は, プリフィクス領域として各PEが個別に専有する.

② 疎結合マルチプロセッサ：図5（b）に示すように, 全PEにおいてすべてのSMWページへのアクセスを禁止し, プライベート空間へのアクセスのみを行う場合, 本システムはメッセージ交換型疎結合マルチプロセッサとなる.

なお, 上では「全PE」と記したが, これに限らず任意のPE群でクラスタを構成し, そのクラスタを上記のように密結合／疎結合マルチプロセッサとすることも可能である. また, 2. 1節で述べたように, 密結合と疎結合の両者を併用したプロセッサ間結合形態も当然可能である.

4.2 均一／不均一メモリ・アクセス

3. 1節で, あるプロセッサがSMWアクセスを行った場合, それは当該SMWに対応するPEのローカル・メモリ（すなわち, リモート・メモリ）へのクロスバー網を経由するリモート・アクセスとなる, と述べた. ここで, 当該SMWに対応するPEが自分自身である場合, そのSMWアクセスの実現法に関して, 次の2つの選択肢が存在する.

ⓐ リモートSMWアクセス：3. 1節の記述通り, クロスバー網をいったん経由したのち, 自分自身のローカル・メモリ（この場合は, リモート・メモリ）へアクセスする. 図2のⓒに相当する.

ⓑ ローカルSMWアクセス：クロスバー網を経由しないで, 自分自身のローカル・メモリへ直接アクセスする. 図2のⓑに相当する.

この点に関して, 本システムでは, 上記のいずれの選択肢も実現している. いずれを選択するかは, システム立上げ時に設定する. よって, この設定如何で, 以下のプロセッサ－共有メモリ間時間距離のいずれをも採ることが可能となる.

① 均一メモリ・アクセス（UMA）：ローカルSMWアクセスを禁止する. すなわち, コモン空間へのすべてのアクセスはクロスバー網を経由する. よって, どのプロセッサからも同じアクセス時間で, コモン空間の任意のロケーションにアクセスできる.

② 不均一メモリ・アクセス（NUMA）：ローカルSMWアクセスを許可する. すなわち, コモン空間へのアクセスの内, 自PEに対応するSMWページへのアクセスはクロスバー網を経由しない. それ以外のSMWページへのアクセスはクロスバー網を経由する. したがって, アクセスするプロセッサ, および, アクセスされるコモン空間のロケーション次第で, アクセス時間が異なる.

5. リモート・アクセス

5.1 メモリ・トランザクション

ローカル／リモート・アーキテクチャにおいては, アクセス対象のメモリのローカル／リモートを問わず, 命令フェッチ, および, データの読出し／書込みが, PU（プロセッサ・ユニット）から見て厳密に同じに出来なければならない. 許されるのは, 唯一アクセス時間の違いだけである.

したがって, リモート・アクセスが起動された場合, PUに代わってそれを遂行する機構が必要となる. 本システムにおいては, MCU（メッセージ通信ユニット）がリモート・アクセスを代行する. [4] このとき, MCUはPUに対して, あたかもリモートメモリのように振舞う.

以下に, リモート・アクセスの際のメモリ・トランザクションについて, そのおおまかな処理の流れを示す.

① あるPEmのPU（PUm）が, コモン空間中のあるSMWnへアクセスしようとした.

② PEmのMCU（MCUm）はこれを認識して, SMWアクセス・メッセージを作成する. そして, SMWnを提供するPEnへクロスバー網を経由してメッセージを送信する.

③ SMWアクセス・メッセージを受信したPEnのMCU（MCUn）は, その指示に従ってMUn（メモリ・ユニット）へアクセスする.

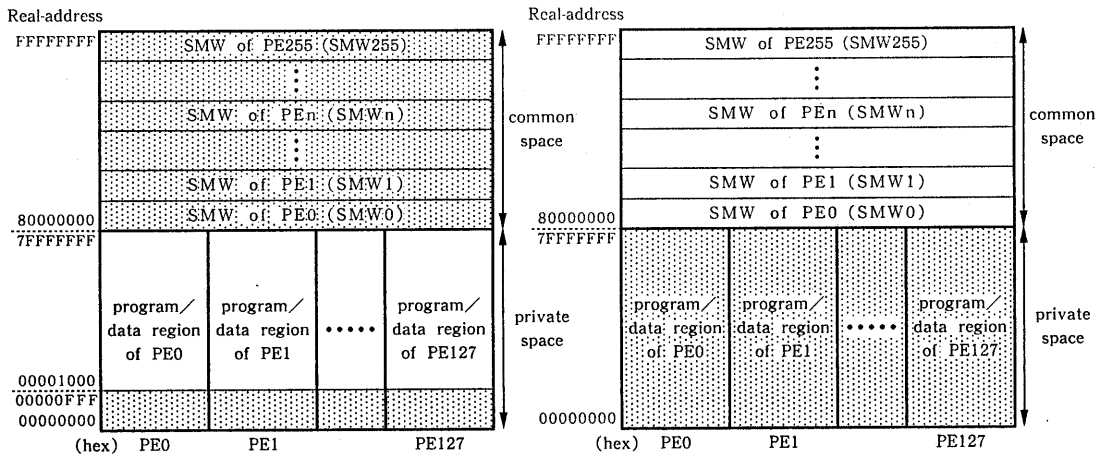


図5. 密結合／疎結合マルチプロセッサ

④ MCUnは，MUnのアクセス結果からREPLYメッセージを作成し，送信元PEmに返信する．
⑤ MCUmは，受信したREPLYメッセージの内容をアクセス結果として，PUmに渡す．
なお，この間PUのバスサイクルはホールドされる．

5.2 PE間通信プロトコル

MCUはPE間の通信手段として，リモート・アクセス，および，プロセス間メッセージ交換といった2つの異なる手段を提供する．これらの実現にあたって，本システムのPE間メッセージ通信機構では図6に示すように，PE間通信プロトコルを次の3つの階層に分けている．[4]

① PE間メッセージ通信プロトコル：最上位層のプロトコルであり，次の2つに分かれる；
i) リモートアクセス・プロトコル：通常のメモリ・アクセスの際のバス・プロトコルと同じ．
ii) プロセス間メッセージ交換プロトコル：OSの通信ハンドラにより定まる．
② メッセージ伝送プロトコル：中間層のプロトコルで，SMWアクセス・メッセージやREPLYメッセージといったプリミティブ・メッセージの組立，送受，解釈などの手順を定める．
③ メッセージ転送プロトコル：最下層のプロトコルで，クロスバー網上での回線接続／切断，データ転送，方向切換えなどの手順を定める．

以下，リモート・アクセスに関連する部分のプロトコルについて，その処理を述べる．

(1) プロトコル変換

5. 1節で述べたように，アクセス対象メモリのローカル／リモートを問わず，そのアクセス・プロトコルは，PUから見て厳密に同じでなければならない．したがって，リモートアクセス・プロトコルは，通常のメモリ・アクセスの際のバス・プロトコルと同一である．

しかし，リモート・アクセスを実際に行うには，5. 1節で述べたように，SMWアクセス・メッセージおよびREPLYメッセージの送受を行う必要がある．このメッセージ送受は，1つ下の層のメッセージ伝送プロトコルに従って行う．よって，リモートアクセス・プロトコルとメッセージ伝送プロトコルとの間のプロトコル変換が必要となる．

このプロトコル変換に要する時間は，リモートメモリへのアクセス時間に大きな影響を与える．そこで，以下の2つの専用ハードウェアを設けている．

① RAC (Remote Access Client)：PUがSMWへアクセスすると起動される．PUがバス上に出力したアドレスおよびアクセスタイプから，SMWアクセス・メッセージを作成する．これをメッセージ伝送プロトコルに従って伝送する．また，その応答であるREPLYメッセージを解析して，要求されているデータ（READアクセス時）をPUに渡す．
② RAS (Remote Access Server)：SMWアクセス・メッセージを受信すると起動される．その指示に従って，MUへアクセスする．このとき，i) 第2レベル・アドレス変換において検出されるページ・フォールトや属性違反，ii) パリティエラー，などの事象が発生する可能性がある．これらアクセス結果，および，要求されたデータ（READアクセス時）からREPLYメッセージを作成し返送する．

(2) メッセージ伝送プロトコル

リモート・アクセス時の基本的なメッセージ伝送プロトコルは図7に示すように，以下の5つのフェーズから成る．[10]

① 回線接続：クロスバー網上で物理的な回線接続を行う．
② リンク設定，データForward転送：RAC−RAS間で論理的なリンク設定を行い，SMWアクセス・メッセージをForward転送する．
③ メモリ・アクセス：受信側のRASがMUへアクセスする．
④ データBackward転送：REPLYメッセージをBackward転送する．
⑤ リンク解放，回線切断：一連のプロトコルを終了する．

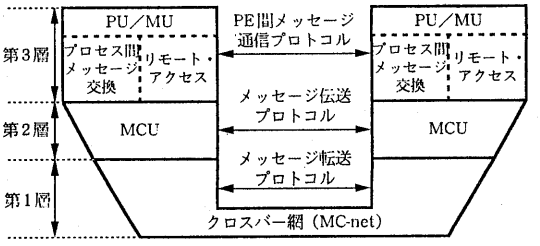


図6. PE間通信プロトコルの階層モデル

5.3 WRITEアクセスの突き放し制御

5.2節で述べたメッセージ伝送プロトコルでは，WRITEアクセスの際でも，リモートメモリでのWRITE動作が終了し，そのアクセス結果であるREPLYメッセージをRACが受信するのを待って，一連のプロトコルを終了する．よって，この間PUのバスサイクルはホールドされ，PUは次の処理へ進めない．

ところが，アクセス側のRACおよびPUにとっては，WRITEアクセスが正常に終了するという保証が得られさえすれば，本来REPLYメッセージを待つ必要はない．さらに，本システムのキャッシュ・システムではメモリ更新アルゴリズムにストアスルー方式を採用しているため，[8] WRITEアクセスの頻度が高く，その高速化が望まれる．

そこで，WRITEアクセスに関し，次の2種のモードを設けた．いずれのモードを選択するかは，システム立上げ時に設定する．

① 同期WRITEモード：図9に示す通り，リモートメモリへのWRITEアクセスの結果であるREPLYメッセージを受信してから，リンクの解放を行う．

② 非同期WRITEモード：RACは，WRITEアクセスの突き放し制御を行う．PUに対しては，メッセージ伝送プロトコルの①の回線接続フェーズが終了した時点で，バスサイクルのホールドを解く．また，クロスバー網に対しては，④のデータBackward転送フェーズを省略し，REPLYメッセージを待たずにリンクの解放を行う．実際のリモートメモリへのWRITEアクセスは，②のデータForward転送フェーズ終了後の適当な時期にRASが独自に行う．

5.4 メモリ・トランザクションの無矛盾性保証

メモリ・トランザクションの処理にあたっては，その結果メモリ内容に矛盾を与えたり，あるいは，プロセッサが誤ったデータを使用したりしないよう，メモリ・トランザクションの無矛盾性を保証する必要がある．この保証は，1個のメモリ・トランザクション内において，および，複数メモリ・トランザクション間において，必要である．

以下，本システムで保証する無矛盾性について，その実現方法を述べる．

5.4.1 メモリ・トランザクションのアトミック性保証

本システムのクロスバー網は回線交換方式を採用しており，回線接続要求の競合に対して，デマンド・モードとプリセット・モードの2種類の調停法を提供している．[4] 後者のプリセット・モードでは，ネットワーク・コントローラ（NETC）の制御下で，回線接続パターンを時分割により切り換えて行く．[7] よって，もしリモート・アクセス中に回線接続パターンが切り換わると回線自体が切断されることになる．すなわち，転送中のデータが失われる．このような事態を防ぐためには，1個のメモリ・トランザクションのアトミック性を保証する必要がある．そこで，回線接続パターンの切換え処理に際して，次の2点を保証するようにする．

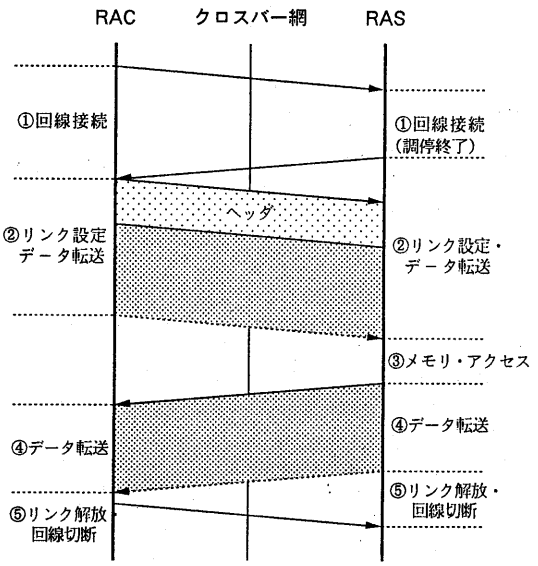


図7. PE間メッセージ伝送プロトコル

① NETCは回線接続パターン切換えに先立って，その旨をMCUに予告する．MCUは予告を受け取ったら，新たなリモート・アクセスのメモリ・トランザクションを開始しない．

② 予告に対して各MCUは，現在処理中のメモリ・トランザクションがなければ，直ちにNETCに回線接続パターン切換えの許可を与える．それ以外の場合，現在処理中のメモリ・トランザクションの終了後，許可を与える．NETCは，全MCUからの許可を受理したときのみ，回線接続パターンを切り換える．

5.4.2 メモリ・トランザクションの逐次性保証

本システムを外側から眺めた場合，複数プロセッサからの複数メモリに対する複数個のメモリ・トランザクションが並列に処理されているように見える．しかし，以下の状況においては，関連する複数のメモリ・トランザクションを逐次化（serialization）しなければならない．

(1) データ依存関係

同一プロセッサによる同一メモリ・ロケーションに対する2個のメモリ・トランザクションは，次の発行順序に逐次化されなければならない．

- WRITE→READ（フロー依存）
- READ→WRITE（逆依存）
- WRITE→WRITE（出力依存）

この理由は，データ依存の定義から明らかである．

本システムでは，RACおよびRASのI/Oポートが1個しかないこと，および，メッセージ伝送プロトコルの定義（5.2節参照）から，上記の2個のメモリ・トランザクションは必然的に逐次化される．

なお，WRITEアクセスの突き放し制御をWRITEバッファを用いて行う場合，上記の逐次化処理はやや複雑になる．これは，WRITEバッファの検索などの機能が必要となるからである．本システムのWRITEアクセスの突き放し制御では，WRITEバッファを用いないので，このような機能は不要である．

(2) 明示的に指定された逐次化

同一プロセッサにより発行された次の2種類の命令は，①→②の順序で逐次化されなければならない．

① WRITEアクセスを行うメモリ・トランザクション（1個ないし複数）

② 1個の不可分メモリ・トランザクション

これは，FETCH&OPのような不可分メモリ操作命令（SPARCではATOMIC LOAD-STORE命令）は，明示的にメモリ・トランザクションの逐次化を規定しているからである．この場合，両者がアクセスするメモリ・ロケーションの一致/不一致に関わらず，逐次化が要求される．

上の(1)同様，WRITEバッファを用いたWRITEアクセスの突き放し制御を行う場合，この逐次化処理は複雑になる（同じ理由）．また，ストアイン方式のキャッシュの場合，コヒーレンス制御方式次第では，すべてのdirtyラインをメモリに書き戻す必要が生じる．

本システムでは，WRITEバッファを用いておらず，また，ストアスルー方式を採用しているので，[8] 容易に逐次化を実現している．

(3) キャッシュ・コヒーレンス

各PEにプライベート・キャッシュを有する場合，2個のプロセッサ（PmとPn）による以下の一連のメモリ・トランザクションにおいて，②→④および②→⑤の順序で逐次化されなければならない．

① あるメモリ・ロケーションに対するPmのREADアクセスの結果生じた，Pmのキャッシュへの当該ライン（L1）のフェッチ
② ①と同一のメモリ・ロケーションに対するPnのWRITEアクセスの結果生じた，ラインL1に対するパージ（あるいは，更新）
③ ①と同一のメモリ・ロケーションに対するPmのREADアクセス
④『任意』のメモリ・ロケーションに対するPmのREADアクセスの結果生じた，Pmのキャッシュへの当該ライン（L2）のフェッチ
⑤ ①と同一のメモリ・ロケーションに対するPmのREADアクセス

このうち，②→⑤の逐次化の必要性は，キャッシュ・コヒーレンス処理の必要性から自明である．ここで，i) ②→③の逐次化が（一見必要そうであるが）不要である点，および，ii) ②→④の逐次化が必要である点，に注意されたい．

まず，②→③の逐次化を行うことは，キャッシュのコヒーレンスの観点からは理想的である．しかし，その実現は，2個のプロセッサが並列動作する限り不可能である（たとえば，②と③が同時に発生した場合を考えればわかる）．一方，②→④の逐次化は，キャッシュ内の各ラインが有する（メモリの）コピーのバージョンを一致させるため不可欠である．すなわち，④でラインL2をフェッチするまでにラインL1をパージしておかないと，キャッシュ内に古いバージョンのコピーL1と新しいバージョンのコピーL2が混在することになるからである．

この②→④の逐次化処理の難易度は，キャッシュ・コヒーレンス処理方式に依存する．本システムでは，“分散グローバル・ディレクトリ法”を採用していることから，[8] ④における「『任意』のメモリ・ロケーションに対するPmのREADアクセス」を検出するのは困難である．したがって，ラインL1とL2が同一のローカルメモリに所属している（つまり，同一のグローバル・ディレクトリで管理される）場合にのみ②→④の逐次化を保証する，といった制限を設けている．これは，リモート・メモリにおいて，④のメモリ・トランザクションが②のメモリ・トランザクションを追い越さないように制御することで，実現できる．

6. まとめ

以上，可変構造型並列計算機のメモリ・アーキテクチャについて述べた．本システムでは，分散メモリ配置構成においてローカルメモリを共有可能とするために，“ローカル／リモート・アーキテクチャ”を採用した．また，そのアドレッシング機構として2レベル・アドレス変換を用いることで，OSによる効率的なメモリ管理を可能としている．さらに，このローカル／リモート・アーキテクチャを基本とすることで，プロセッサ間結合形態として密結合／疎結合マルチプロセッサ，また，プロセッサ－共有メモリ時間距離として均一／不均一メモリ・アクセス，のいずれをも提供する．さらに，クロスバー網を経由するリモート・アクセスの実現法についても述べた．

謝辞

現在我々とともに設計・開発に携わっている蒲池，廣谷，福澤，岩田，草野，恒富，上野，杉山の各氏，および，日頃ご討論いただく富田研究室の皆様に感謝致します．

本システムの開発にあたって多大なるご協力を戴いている以下の方々に，厚い感謝の意を表します．

①クロスバーLSI開発：
・（株）東芝・総合研究所・情報システム研究所の小柳滋博士
②PE開発：
・富士通（株）本体事業部・スーパーコンピュータ開発部の内田啓一郎氏，高村守幸氏
・京セラ（株）LSIデザイン事業部の重村慎二氏
・（株）物産システムテクノロジー・ASIC事業部の小柴繁一氏
③基板開発：
・ニシム電子工業（株）の和田勲夫氏，ディジタル化推進プロジェクトチームの諸氏
・凸版印刷（株）精密電子事業本部の綿井潔氏
・住商機電販売（株）の服巻泰雄氏
④システム実装：
・アジアエレクトロニクス（株）の作田信彦氏，システム事業部の諸氏

参考文献

[1] 村上ほか：“可変構造型並列計算機のシステム・アーキテクチャ，”情報処理学会「コンピュータ・アーキテクチャ」シンポジウム論文集，pp.165-174（1988年5月）．
[2] K.Murakami et al.：“The Kyushu University Reconfigurable Parallel Processor－Design Philosophy and Architecture－，”Proc. IFIP 11th World Computer Congress, pp.995-1000, Sept. 1989.
[3] K.Murakami et al.：“The Kyushu University Reconfigurable Parallel Processor－Design of Memory and Intercommunication Architectures－，”Proc. 1989 Int'l. Conf. Supercomputing, pp.351-360, June 1989.
[4] 森ほか：“可変構造型並列計算機のPE間メッセージ通信機構，”情報処理学会論文誌，vol.30, no.12, pp.1593-1602（1989年12月）．
[5] 蒲池ほか：“可変構造型並列計算機のメモリ・アーキテクチャ，”情報処理学会第38回全国大会講演論文集，3T-2（1989年3月）．
[6] 蒲池ほか：“可変構造型並列計算機のネットワーク制御方式，”信学技法，CPSY89-16（1989年8月）．
[7] 甲斐ほか：“可変構造型並列計算機のネットワーク制御，”情報処理学会第39回全国大会講演論文集，5X-3（1989年10月）．
[8] 岩田ほか：“可変構造型並列計算機のキャッシュ・システム，”情報処理学会計算機アーキテクチャ研究会資料，89-ARC-79-3（1989年11月）．
[9] 甲斐ほか：“可変構造型並列計算機のPE間通信プロトコル，”情報処理学会第40回全国大会投稿中．
[10] M.A.Holiday：“Page Table Management in Local／Remote Architectures，”Proc. 1988 Int'l. Conf. Supercomputing, pp.1-8, July 1988.
[11] R.J.Swan et al.：“Cm*－A Modular, Multi-Microprocessor,”Proc. 1977 NCC, pp.39-46, 1977.
[12] R.J.Swan et al.：“The Implementation of the Cm* Multi-Microprocessor,”Proc. 1977 NCC, pp.645-655, 1977.
[13] E.F.Gefinger et al.：“Parallel Processing：The Cm* Experience,”Digital Press, 1987.
[14] G.F.Pfister et al.：“The IBM Research Parallel Processor Ptototype（RP3）：Introduction and Architecture，”Proc. 1985 Int'l. Conf. Parallel Processing, pp.764-771, Aug. 1985.
[15] W.C.Brantley et al.：“RP3 Processor'Memory Element,”Proc. 1985 Int'l. Conf. Parallel Processing, pp.782-789, Aug. 1985.
[16] 福田ほか：“可変構造型並列計算機の並列／分散オペレーティング・システム，”情報処理学会オペレーティング・システム研究会資料，89-OS-43-8（1989年6月）．
[17] 福澤ほか：“可変構造型並列計算機の並列／分散オペレーティング・システム－スレッドの実現－，”情報処理学会オペレーティング・システム研究会資料，89-OS-45-6（1989年11月）．
[18] 草野ほか：“可変構造型並列計算機のオペレーティング・システム－メモリ管理－，”情報処理学会第40回全国大会投稿中．