《小特集》

共有メモリ型並列計算機

松　本　　尚*

ABSTRACT Shared-memory parallel architecture (shared-memory multiprocessors) has been widely used in computers from multimicroprocessors to mainframes. This paper gives an outline of architectural issues on shared-memory parallel processing systems. The merits of adopting shared-memory model to parallel computer systems are described, and the methods of constructing shared-memory systems are illustrated from the view points of processor-memory interconnection networks and memory caching technique. After explaining sychronization issues which are important for parallel processings, mechanisms for reduction of the overheads caused by synchronization are discussed. Then, approaches for future extensions of shared-memory parallel computer systems are mentioned.

1. はじめに

複数のノイマン型プロセッサと共有メモリシステムを持つ共有メモリ型並列計算機は，マルチマイクロプロセッサから大型計算機まで，世の中で最も多く使われている並列計算機である[1)~3)]．この共有メモリ型並列計算機のアーキテクチャについて解説を行う．本稿では，共有メモリを広く解釈し，ユーザに共有メモリ空間を提供する並列計算機システムはすべて共有メモリ型並列計算機に包含されるものとする．また，議論の単純化のために，要素プロセッサはすべて均質であることを仮定する．共有メモリ型並列計算機はマルチユーザやマルチジョブが容易に実現できるという汎用性に大きな特長があるため，プロセスという1台の実プロセッサを仮想化した概念を説明に用いる．アプリケーションプログラマはプロセスを用いて処理を記述し，プロセスはオペレーティングシステムの下で実プロセッサの割り当てを受けて処理を実行する．並列処理は複数のプロセス間で通信と同期を行いながら実行される処理として記述される．

共有メモリ型並列計算機は，平易に言えば，単にすべてのプロセッサが同じメモリシステムを使用する並列計算機である．しかし，多数のプロセッサが同時に高速に読み書きできるようなメモリシステムの実現は困難である．規模が大きくなるとメモリアクセスの遅延（レイテンシ：latency）も大きくなり，キャッシュ技術やスケジューリングの最適化等を駆使しなければ，実用に耐える共有メモリシステムは構成できない．では，何故共有メモリモデルを採用して並列計算機を構成するのか．まず，本稿では共有メモリ型並列計算機のメリットについて述べる．そして，最大の特徴である共有メモリの構成法について，プロセッサとメモリ間の相互結合網の視点と性能を引き出すためのキャッシュ技術の視点とに分けて解説する．さらに，並列処理に不可欠なプロセス間の同期について説明し，同期のオーバヘッドを減らして高性能化するための機構を紹介する．最後に共有メモリ型並列計算機をさらに高性能化するためのアーキテクチャに関連する研究について紹介する．

2. 共有メモリ型並列計算機のメリット

2.1 単一プロセッサモデルとの互換性

ユーザが並列処理を行わない場合，そのユーザにとっては単一プロセッサによる計算モデルとまったく差異がなく，同じスタイルでプログラミングが可能である．また，単一プロセッサのマシンにおける並行処理（複数のプロセスを1個のプロセッサで時分割に実行すること）は成熟した技術となっており，このプロセスのスケジューリングをマルチプロセッサ[注1)]上の異なったプロセッサに割り当てるように変更するだけで，システムの処理性能を向上させることができた．共有メモリがハードウェアで実現されている場合は，プロセッサに対するメモリシステムの見え方が従来と同じであるため，従来のオペレーティングシステムを

Shared-Memory Parallel Computer Systems. By *Takashi Matsumoto* (Dept. of Information Science, Faculty of Science, Univ. of Tokyo).

*東京大学理学部情報科学科

マルチプロセッサへ移植することも簡単であった．実用化という面では，この単一プロセッサモデルとの互換性は，過去30年以上かけて培われた逐次型ソフトウェアの技術をそのまま利用できるので大きな利点である．実際に，初期の市販のマルチプロセッサはシングルジョブの高速化のために市場に受け入れられたのではなく，複数の従来の単一プロセッサ用のジョブを同時に処理することでスループットを向上させるために購入されたといわれている．ただし，このようにプロセス間の通信や同期を頻繁に行う必要のない使用形態では，高速なネットワークで結合し分散オペレーティングシステムを持つ複数の単一プロセッサシステムと等価である．マルチプロセッサとして1つのシステムにまとめることによるメリットは入出力装置や筐体や電源装置の共用によって価格性能比を改善する程度である．(逆にこの入出力装置の共用がボトルネックになって性能が向上しない場合もある.)

プロセッサモデルというよりプロセッサハードウェアの互換性と言った方が正しいが，従来の単一システム用のプロセッサを要素プロセッサに流用して比較的小規模の並列計算機システムを構成する場合，プロセッサ間通信のバンド幅を大きくするのに共有メモリが有利である．つまり，従来型のプロセッサではプロセッサ・メモリ間のデータ転送能力が最も高く，入出力ポートを使ったデータ転送能力はそれより大幅に低い．このため，ハードウェアとして共有メモリを実現することがプロセッサ間の通信能力を高めることになる．ただし，プロセッサ間通信用の専用高速入出力ポートを持ったプロセッサを開発すれば，この議論は成立しない．

2.2 プログラミングの高い自由度

共有メモリ型並列計算機を本来の並列処理で利用する場合のメリットはプログラミングの自由度が高い点である．高自由度を示す特徴の1つに，共有メモリモデルに基づいているため，プロセス間の通信に広大な共有メモリが利用できることが挙げられる．例えば，数値計算で主要な役割を担うデータ構造である配列や，ヒープ上に作られるLISPのリスト構造といったプログラム言語の用いるメモリ上の構造体そのものを，明示的な通信処理なしに共有変数としてプロセス間で受け渡すことができる．ただし，共有変数のアクセスには後に述べる同期の問題が生じる場合がある．プロセス間で明示的な通信を行う場合でも，通信用のバッファを自由に共有メモリ上に設けることができる．さらに，従来の手続き型言語を並列処理用に拡張した場合は，common文やポインタといった大きな単一のメモリ空間の存在を仮定する概念が存在し，分散メモリモデルに比べずっとナイーブにこれらの言語のマッピングが可能である．また，高自由度を示す他の特徴にMIMD型であることが挙げられる．SIMD型とは異なり，各プロセッサを同時に別々の命令ストリームで制御できるので複雑な動作を行わせることができる．

共有メモリ型並列計算機のメリットとしてプログラミングの自由度の高さについて述べてきたが，これは逆にデバッグの難しさや最適化されたコードの生成の難しさを引き起こす．しかし，最近のソフトウェア技術（特に並列化コンパイラや並列言語）の進歩によって，アプリケーションプログラムを作成するユーザは低レベルのプロセス間通信や同期を直接記述する必要がなくなりつつある．

2.3 マルチユーザ・マルチジョブ実現の容易性

並列計算機で効率の良いマルチユーザやマルチジョブ機能が実現できるためには，実プロセッサが仮想化可能であり，プロセスのプリエンプション[注2)] (preemption) が可能であり，システム性能を向上させるためにプロセスを動的に負荷分散できる必要がある．共有メモリ型並列計算機では，通信や同期を共有メモリをベースにして行えば，任意のプロセッサから同じメモリシステムが見えているため，オペレーティングシステムがプロセスを任意のプロセッサに割り付けることができる．また，単一プロセッサと同様にプロセスのプリエンプションも可能である．さらに，プロセスの動的負荷分散，つまりあるプロセッサで実行途中のプロセスを他のプロセッサに移動させて実行させるプロセスマイグレーション (process migration) についても，レジスタコンテクストのみを移送することで可能である．

また，共有メモリモデルを採用することがキャッシュ技術の使用を容易にしている．ノイマン型のプロセッサはプログラムの局所性を利用することで高速化を達成してきた．頻繁に参照するメモリを自動的にキャッシュに格納することで，メモリアクセスのレイテンシを防ぎ高速化している．共有メモリ型並列計算機においても各プロセッサごと（もしくは数台のプロセッサごと）にキャッシュを用意して，共有メモリをキャッシングすることでレイテンシを減らし高速化が達成できる．ただし，この場合後述するキャッシュの一貫性の問題を解決する必要がある．キャッシュはもともと実行時の参照の頻度の高いメモリの内容をプロセッ

サの近傍に自動的に移送（マイグレーション）する機構なので，マルチユーザ・マルチジョブのためのプロセッサの仮想化と問題なく組み合わせることができる．つまり，プロセスマイグレーションが行われた場合，移送されたプロセスの実行は新しいプロセッサにおいてもキャッシュによる高速化の恩恵を受けることができる．

3. 共有メモリの構成法

共有メモリの構成法は大きく分けて集中共有メモリと分散共有メモリの2つのタイプがある．集中共有メモリはハードウェアで共有メモリを実現しており，共有メモリの実装上の設置場所も各プロセッサからアクセス時間の意味で等距離の位置に配置される（図1）．性能上の観点から，プロセッサと共有メモリの間の通信バンド幅は十分に大きく設計され，すべてのプロセッサからのメモリアクセス要求に対して，飽和することなしに応答できるようになっていることが望ましい．分散共有メモリは必ずしもハードウェアのみで共有メモリシステムが実現されるわけではなく，メモリの実装上の設置場所は各プロセッサに分散配置されている（図2）．そして，他のプロセッサの設置場所のメモリへのアクセスを可能にすることで共有メモリを構成しており，共有メモリへのアクセス時間は非対称になっている．

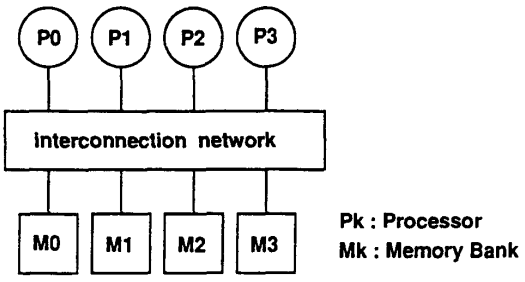

図1　集中共有メモリの実装上のモデル

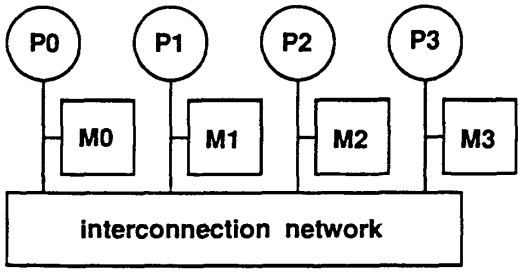

図2　分散共有メモリの実装上のモデル

3.1 集中共有メモリの構成法

集中共有メモリの構成法について，代表的なプロセッサとメモリ間の相互結合網[4),5)]を具体的に示して説明を行う．

高価なマルチポートメモリを使用せずになるべく同一のメモリモジュールにプロセッサのアクセスが集中することを防ぎ，かつメモリ入出力のバンド幅を広げる方法として，共有メモリを複数の独立したモジュールに分離するメモリインタリーブ（メモリ多重化）という方法がある．n個のプロセッサがある場合にメモリのアクセス競合を減らすために共有メモリをmバンク（多い方が競合の可能性は減り，バンド幅も増える）に分割する．分割の方法はアドレスの下位ビット等で指定するのが一般的である．プロセッサとメモリバンク間の接続方法は，あらゆるパスを直接接続する方式（完全結合）が最もナイーブなものである．この場合，nm本の接続（しかも性能を確保するためには1接続当たり数十本の信号線が必要）で結ばなければならなくなる（図3参照ただし，図は8プロセッサ，8メモリバンクの場合）．このような完全結合はプロセッサ台数が多くなると競合を避けるためにメモリバンク数も増やさざるを得ず，信号線数が膨大になり現実的ではない．

実装の簡便さから現実のマルチプロセッサに最もよく用いられている結合方式に共有バス結合がある．この方式は共有バスという資源をすべてのプロセッサで共用（時分割使用）しており，共有メモリも共有バスに接続されている（図4）．メモリアクセスする際は

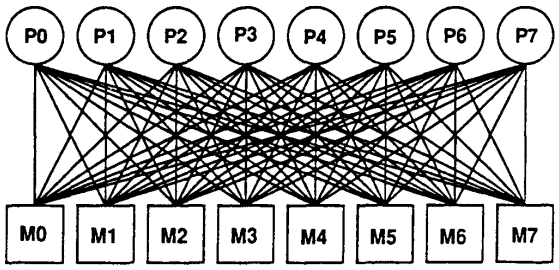

図3　プロセッサ・メモリ間の完全結合

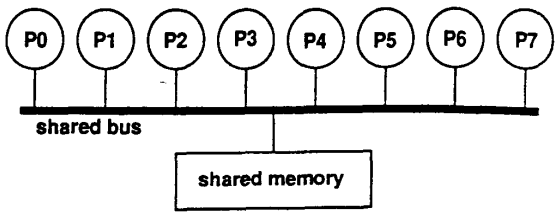

図4　共有バス結合

バスの使用要求を出し，使用権を獲得（使用権を与えるプロセッサを選ぶ作業をバスアービトレーションという）した唯一のプロセッサがそのバスに接続されているメモリをアクセスできる．すべてのプロセッサが共有メモリアクセス時に共有バスを使用するため，すべてのプロセッサの要求を満足させる程度にバスのバンド幅が広い必要がある．接続形態としては共有バス1本のシステムは1メモリバンクのプロセッサとメモリバンク間の完全結合にすぎない．しかし，後述するスヌープキャッシュの技術により共有バスの使用頻度を劇的に減らすことができるようになり，比較的多数のプロセッサが接続できるようになった．

上述のプロセッサとメモリバンク間の完全結合と共有バス結合の接続形態は静的なものであったが，スイッチを介して動的に接続する方法もある．**図5**に示すクロスバー網はプロセッサから1段のスイッチを経て任意のメモリバンクにアクセス可能になるものである．また，任意のプロセッサとメモリバンクの組合せを同時に接続可能であり，接続能力は完全結合と同じである．ただし，この方式もnプロセッサ，mメモリバンクでnm個のスイッチを必要とする（完全結合の接続ラインがスイッチに置き換わっている）ため，プロセッサ台数が多くなると実現が困難である．

接続能力が少し低くなるが，大幅にスイッチの数を減らした結合網の中の代表的なものとして，**図6**のオメガ網（Omegaまたは別称シャッフルエクスチェンジ：shuffle-exchange網）を取り上げる．オメガ網は複数のスイッチを経由してプロセッサがメモリバンクにアクセスが可能になる多段結合網で，Nプロセッサ，Nメモリバンク時でlog N段のスイッチを経由することになる．プロセッサとメモリバンクの接続経路は接続要求しているメモリバンクが使用されていなくても，他のプロセッサとメモリバンクとの接続によって経路の競合が起こり，確保されるとは限らない．また，log N段のスイッチを経由してメモリへのアクセスが行われるため，メモリアクセスのレイテンシも問題となる．

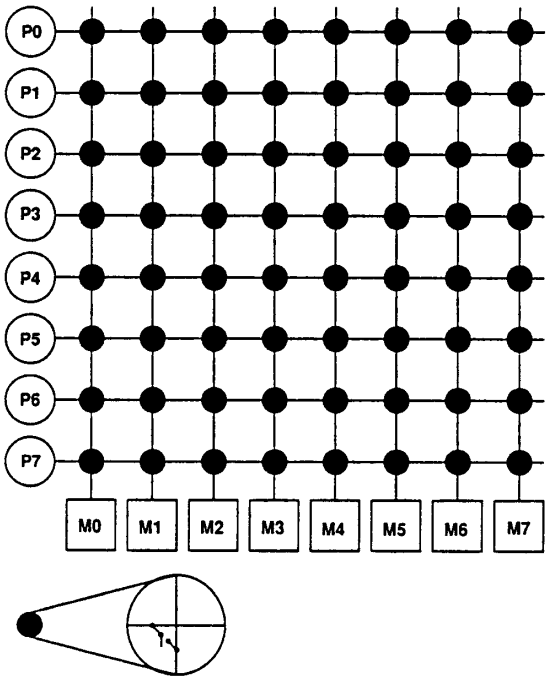

図5　クロスバー結合網による接続

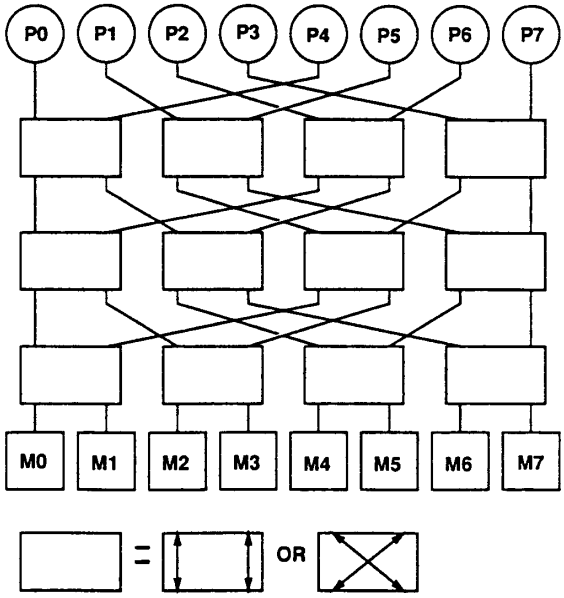

図6　オメガ結合網による接続

3.2　分散共有メモリの構成法

分散共有メモリではプロセッサとメモリの間を相互結合網で接続するのではなく，プロセッサとメモリを組合わせたエレメント間を接続する．集中共有メモリに用いるような高性能な結合網を使用するシステムもあるが，通常分散共有メモリでは実装の容易さと拡張性が重視されるため，集中共有メモリのプロセッサとメモリ間の相互結合網に比べ，簡易で性能の劣る相互結合網がプロセッサ間結合に採用されることが多い．結合形態としてはメッシュ（near-neighbor mesh）結合（**図7**）やリング（ring）結合（**図8**）や2進nキューブ（binary n-cube）結合（**図9**はn=4）といった接続法がよく用いられる．これらの場合，通信や同期のオーバヘッドが大きいため，粒度[注3)]の細かい並列計算の実行はあきらめ，ユーザにメモリシステム

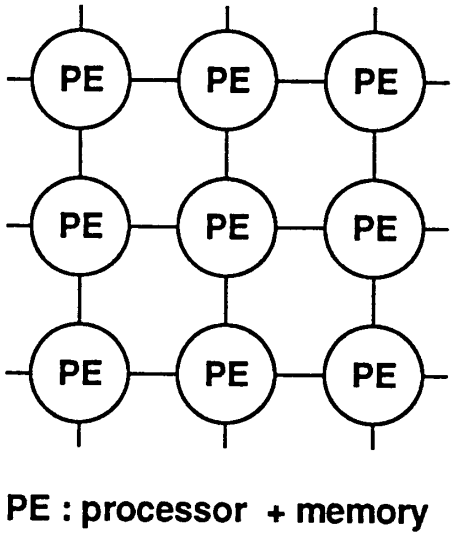

図7　メッシュ結合

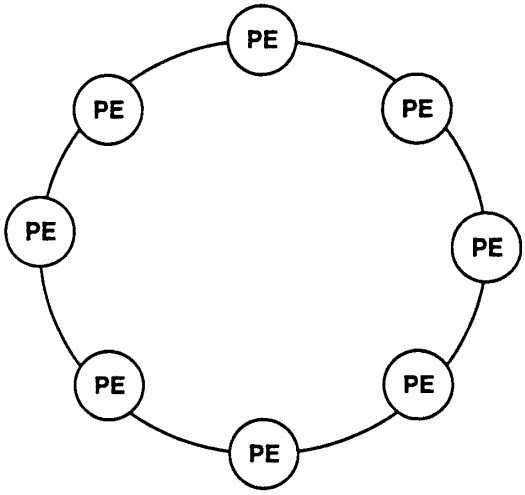

図8　リング結合

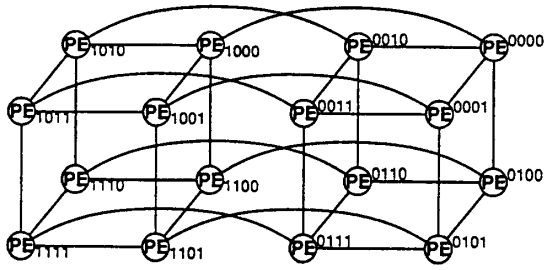

図9　2進4キューブ結合

を論理的な単一共有メモリとして見せることによる使い易さの向上と，マルチユーザやマルチジョブに伴う負荷分散を容易にすることを主眼としている．

3.3　共有メモリにおけるキャッシュ技術の利用

並列計算機の共有メモリは最初に述べたようにユーザに様々な便宜や論理的な自由度を与えるが，反面実装が難しくメモリアクセスのレイテンシを増加させる傾向がある．共有メモリ型の並列計算機では基本的にメモリアクセスはすべて共有メモリへのアクセスなので，この共有メモリアクセスのレイテンシを減らさなくては，効率が上がらず使用に耐えない．実装上は分散メモリアーキテクチャである分散共有メモリでは，プロセッサローカルのメモリに自分の使うデータを割り当てることで，レイテンシを減らすことができる．しかし，このメモリ割り当てをアプリケーションプログラマに明示的に行わせたのでは共有メモリモデルを採用するメリットが消えてしまう．そこで，集中共有メモリと分散共有メモリの両方式共に，レイテンシの削減のためキャッシュ技術が用いられる．ただし，分散共有メモリの場合はファームウェアでキャッシュが実現されているシステムもある．

3.3.1　キャッシュの一貫性の問題

共有メモリに対して複数のキャッシュが存在する場合には以下のような問題点がある．つまり，図10のように複数のプロセッサが同一のメモリ領域（共有変数X）をアクセスする場合，それぞれのキャッシュにその領域のコピーが存在することになる．この状況で，キャッシュとして従来の単一プロセッサ用のものを仮定すると，あるプロセッサが共有変数に書き込みを行っても，書き込みの処理がその共有変数のコピーを保持している他のキャッシュに伝わらない．このため，共有変数の更新後に，他のプロセッサが共有変数を読

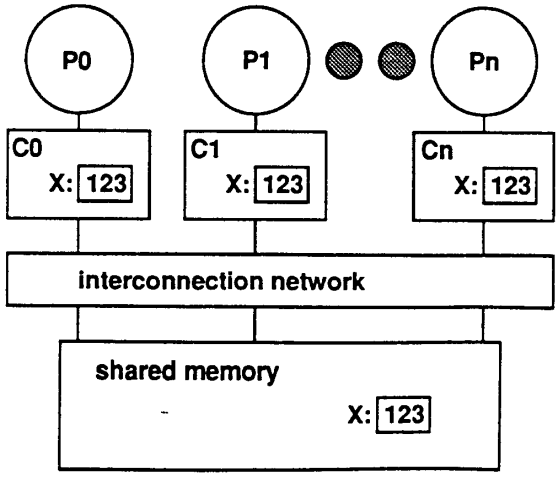

図10　集中共有メモリのキャッシュモデル

み出しても，古い値のコピーがキャッシュに存在しているため，この古い値を読み出してしまう．この状況を避けて，正しい値が読み出せるようにすることをキャッシュの一貫性（コンシステンシ：consistency）の保持という．キャッシュの一貫性を守るキャッシュをコヒーレントキャッシュ（coherent cache）と呼ぶ．また，キャッシュの一貫性を保持するために，キャッシュ間である決められた手順を守る必要がでてくる．この手順のことをキャッシュコヒーレンスプロトコル（cache coherence protocol）と呼ぶ．

3.3.2 ディレクトリキャッシュ

共有メモリに使用される代表的なコヒーレントキャッシュ方式を簡単に紹介する．歴史的に古く，多くの大型計算機に使用されている方式としてディレクトリ（directory）方式がある[6)~8)]．この方式は基本的に共有メモリのブロックのコピーをどのキャッシュが持っているかを管理する方式で，その管理のための情報をディレクトリと呼ぶ．フルマップディレクトリ（full-map directory）と呼ばれる方式では共有メモリのブロックごとにコピーを持っているキャッシュを示すための1ビットのタグをキャッシュ数分だけ持っている（このタグがディレクトリ）．プロセッサが読み出しを行って，自分のキャッシュへコピーを作る時に，該当する共有メモリのブロックのタグがセットされる（図11(a)，(b)参照）．プロセッサの書き込みに関してはさらに二種類の方式に細分される．ライトスルー（write-through またはストアスルー：store-through）と呼ばれる方式では，必ずデータは共有メモリに書き込まれ，タグを参照しそのブロックのコピーを持っているキャッシュに対しそのブロックの無効化メッセージを送って，そのコピーを無効化することで一貫性を保持する．しかし，ライトスルーではプロセッサの書き込みはすべて主記憶へのアクセスになるため，キャッシュによる主記憶へのアクセス頻度の削減効果は少ない．そして，プロセッサ数が多くなると相互結合網が飽和してしまうため，小規模のマルチプロセッサにしか用いられない．ライトバック（write-back またはストアイン：store-in）と呼ばれる方式では，通常はキャッシュに対してのみデータが書き込まれ，共有メモリには必要最小限しかデータが書き戻されない方式で，書き込みに対してもトラフィック削減の効果が期待できる．フルマップディレクトリでライトバックを用いる場合は，共有メモリのブロックと各キャッシュのブロックに対してダーティビット（dirty bit）と呼ばれる1ビットのタグが追加される．ダーティビットは D（dirty）と C（clean）のどちらかの値を取り，D状態はキャッシュのブロックを共有メモリに書き込みを行わずに局所的に更新したことを意味し，C状態はそのブロックが最新の値であることを意味する．プロセッサの書き込み時に該当ブロックがC状態である場合（図11(b)），もしくはキャッシュ内にコピーがない場合は，他のキャッシュの該当ブロックのコピーを無効化し，共有メモリの該当ブロックをD状態にして，自分のキャッシュのブロックを更新しD状態

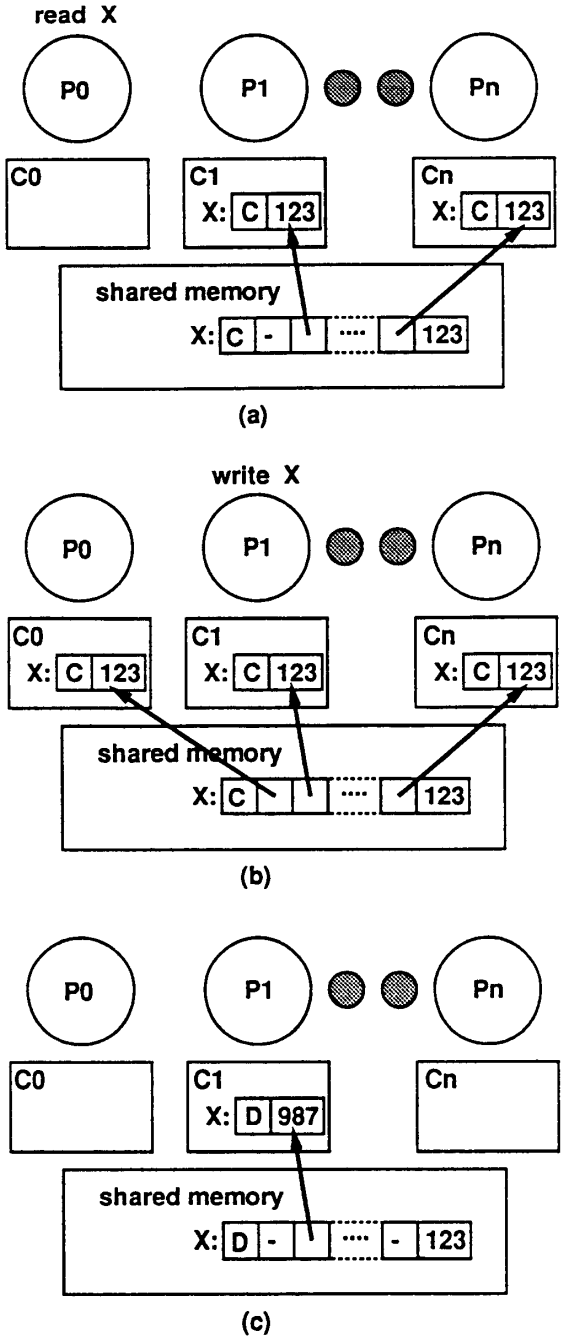

図11 フルマップディレクトリキャッシュの状態遷移

に変更する（図11(c)）．書き込み時にブロックがD状態であれば，そのブロックを持っているキャッシュが他にないことが保証されているため，単にキャッシュ内でブロックを更新する．読み出し時はキャッシュにコピーがあれば，その値を使用する．コピーがない場合（キャッシュミス）は共有メモリを参照し，共有メモリの該当ブロックがC状態ならブロックの値をそのまま読み出し，D状態ならディレクトリを参照して更新したブロックを持っているキャッシュに最新の値を共有メモリに書き込ませた後にブロックを読み出す．このキャッシュの管理手順をフルマッププロトコルと呼ぶ．

フルマップディレクトリは主記憶のブロックごとにシステムのキャッシュ数に比例するタグを必要とする．このため，それほど多くのプロセッサ（キャッシュ）に対応したシステムが構成できない．この難点を緩和する方法として，同時にコピーを保持できるキャッシュの数を制限する方式(リミテッドディレクトリ：limited directory）やキャッシュへのポインタを1個のみ保持するディレクトリを主記憶とキャッシュに用意して，ポインタチェインでコピーを保持するキャッシュを表現する方式（チェインドディレクトリ：chained directory）が提案されている．

分散共有メモリにも基本的にディレクトリ方式のキャッシュ法が使用され，共有メモリが分散配置されているため，ディレクトリ情報も分散配置される[9)~11)]．分散共有メモリを構成するシステムでは粒度の細かい並列処理を扱うことを諦めている場合が多く，ハードウェアキャッシュを持たずにファームウェアでキャッシュの制御（プロセッサローカルなメモリにデータをコピーまたは移送）を行なうシステムや，管理を行う単位であるブロックのサイズが集中共有メモリに比べると大きいシステムが多い．また，ブロックのサイズを仮想記憶の管理の単位であるページと同じ大きさとして，ページミスをキャッシュミスに流用することで付加ハードウェアなしで効率の良いキャッシュ管理を行っているシステム[9)]もある．

3.3.3 スヌープキャッシュ

共有バス結合のマルチプロセッサに一般的に利用されているコヒーレントキャッシュ方式にスヌープキャッシュ（snoop cache）方式がある[12)~15)]．ディレクトリ方式ではブロックのコピーがどのキャッシュに存在するかというディレクトリ情報が管理されているが，これは無効化メッセージの送付を該当ブロックを持っているキャッシュに対してのみに限定することで，相互結合網のトラフィックを減らすためである．もし，キャッシュから共有メモリへのトランザクションがすべてのキャッシュに放送（broadcasting）されていれば，ディレクトリは必要ない．代わりに，各キャッシュは他のキャッシュの共有メモリへのトランザクションを監視（snooping）して，コンシステンシが保持できるようにキャッシュ内のブロックを管理する．共有バス結合ではバスがすべてのキャッシュに共用されており，共有メモリへのトランザクションはバスを監視することですべて把握することができる．また，共有バス結合ではすべてのキャッシュが主記憶へのアクセス時にバスを利用するため，バスの使用頻度をできるだけ下げるキャッシュコヒーレンスプロトコルが有利である．この意味ではライトバック方式のキャッシュの方が適しており，ライトバック方式に基づく多くのプロトコルが考案されている．

ここでは，Dragonプロトコルを例にとって説明する．ディレクトリ方式のコヒーレントキャッシュでは共有ブロックに対する書き込みに対して，書き込みを行ったプロセッサ以外のキャッシュを無効化していたが，共有バスを用いるシステムでは新しい値に更新（update）するという戦略も容易に行える．Dragonプロトコルはこの更新型のプロトコルの一種である．Dragonプロトコルではディレクトリ方式のフルマッププロトコルで使用したダーティビットの他にキャッシュブロックごとにブロックの共有状態を示す共有ビットと呼ばれる1ビットのタグが追加されている．この共有ビットはS（shared）とP（private）の二状態のいずれかを取る．S状態は自分の他にもそのブロックのコピーを持っているキャッシュが存在していることを示し，P状態は自分だけがコピーを持っていることを示している．バストランザクションごとにすべてのキャッシュはバスを監視し，自分がそのトランザクションに該当するブロックのコピーを持っているかどうかの信号を出力する．この信号によってS状態かP状態かが動的に決定される．この共有ビットとダーティビットによりキャッシュブロックは4状態（P&C, P&D, S&C, S&D）で管理される．S状態（S&CまたはS&D）のブロックに対してプロセッサが書き込みを行う場合は，キャッシュはバスを獲得してデータを他のキャッシュにも放送し，該当ブロックを持っているキャッシュは値を更新する．ただし，Dragonプロトコルではこのキャッシュ更新のバストランザクション時に主記憶のデータの更新は行われず，P&D状態またはS&D状態のブロックがリプレースによ

表1　主なスヌーププロトコル

名称	タイプ	書き込み方式	主記憶へのダーティブロックの書き戻し時期
Write Once	無効化型	ライトバック（最初はライトスルー）	バス使用時はすべて書き戻す
Berkeley	無効化型	ライトバック	ブロックのリプレース時のみ書き戻す
Illinois	無効化型	ライトバック	バス使用時はすべて書き戻す
Dragon	更新型	ライトバック（共有ブロックはバスで放送）	ブロックのリプレース時のみ書き戻す
Firefly	更新型	ライトバック（共有ブロックはライトスルー）	バス使用時はすべて書き戻す

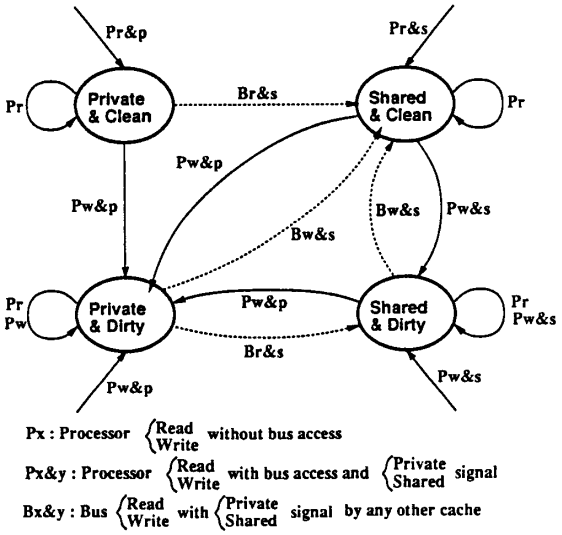

図12　Dragon プロトコルの状態遷移図

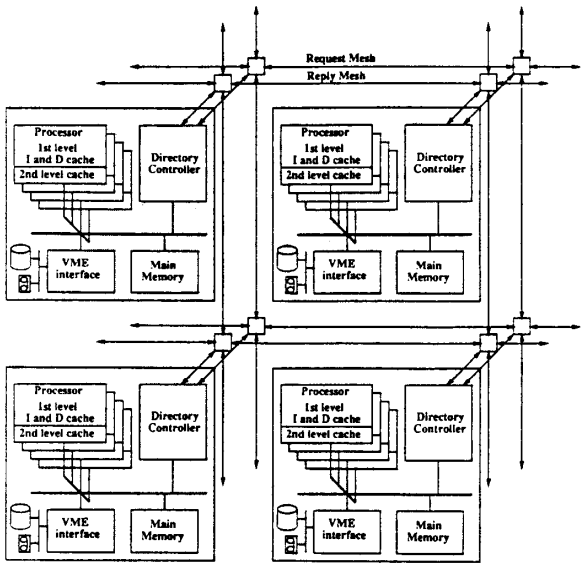

図13　DASH システム

ってキャッシュから追い出されるときにデータは主記憶に書き戻される．また，プロセッサからの書き込み時に該当ブロックがキャッシュにない場合は，一般にプロセッサの書き込むデータの大きさとキャッシュのデータ管理の単位であるブロックの大きさが異なるため，キャッシュは該当ブロックを読み出した後に書き込みのための手順を行う（fetch-on-write）．Dragon プロトコルのキャッシュブロックの状態遷移図を図12に示す．

主なスヌーププロトコルを表1に示す．無効化型のプロトコルは書き込みを行う場合に，該当ブロックをキャッシュに独占し，キャッシュ間で共有されているブロックの数を減少させるので，比較的粒度の大きな並列処理や独立したプロセスをプロセッサ間でマイグレーションしながら実行させる場合に都合が良い．更新型のプロトコルは共有ブロックを無効化することなしに更新し続けるので，頻繁に通信を行う粒度の細かい並列処理や複数のプロセッサで頻繁に参照更新する同期変数の処理に都合が良い．

3.4　階層化による共有メモリの大規模化

共有メモリの構成法についてプロセッサとメモリ間の相互結合網とキャッシュ技術を中心に説明してきた．共有メモリ型並列計算機でプロセッサの数を増やして大規模並列化や超並列化を目指す場合，集中共有メモリでは相互結合網の複雑さや物理上の実装の制約からプロセッサとメモリの距離が離れ，性能の向上が図れなくなる．このため，共有メモリ型並列計算機は階層構造を採ることで，大規模化を図るものが多い．つまり，比較的小数のプロセッサは集中共有メモリで結合し，そのプロセッサとメモリのグループ（クラスタ：cluster）をさらに相互結合網で結び，クラスタ間では分散共有メモリを構成する．このようなシステムの例として Stanford 大学の DASH[16]（図13）と Paradigm[17]（図14）が挙げられる．しかし，階層化されたシステムでは，クラスタをまたがるプロセッサ間の通信や同期はクラスタ内に比べコストがかかるため，オペレーティングシステムは密に協調動作を行うプロセスを同じクラスタ内のプロセッサに割り付ける等の配慮を行う必要がある．

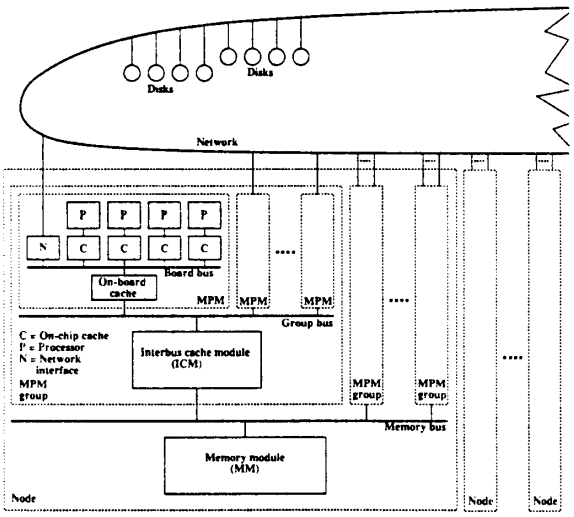

図14 Paradigm システム

4. 同期機構

並列計算機のアーキテクチャではプロセッサ間の通信と同期のオーバヘッドをいかに小さく抑えるかが大きな課題である．共有メモリ型並列計算機ではプロセッサ間の通信は共有メモリを介して行う．プロセッサ間の同期に関しても，共有メモリ上に同期変数を取って操作を行えば，特別な機構がなくても同期処理を実現することはできる．しかし，多くの並列計算機システムは性能を向上させるためにハードウェアで実現された同期のための機構を用意している．同期は大きく分けて排他制御（相互排除：mutual exclusion）と条件同期の2種類に分類できる．

4.1 排他制御

排他制御は共有資源を複数のプロセスで共用する場合に，1つのプロセスがその資源を使用中には他のプロセスが使用できないことを保証する同期である．排他制御は純粋にソフトウェアだけでも行える[18]が，複雑でコストがかかる．そこで，排他制御を簡単に実現するために，複数のメモリ操作を不可分に行う不可分命令が考案され，多くのプロセッサに実装されている．単一プロセッサによる並行処理では不可分命令の実行中はプロセス切替ができないのでプロセス間でこの命令の排他性が保証されるが，並列計算機の場合はさらに他のプロセッサからの排他性の確保，つまりあるプロセッサが不可分命令でメモリ操作を行っている場合は他のプロセッサがそのメモリに対して操作ができないことの保証を行う必要がある．もっとも単純には，不可分命令の期間は他のプロセッサの共有メモリへのアクセスを完全に止める方法があり，共有バス結合システムではかなり利用されている．他の解決法としては相互結合網または主記憶側で他のプロセッサからのメモリアクセスを調べて，不可分命令で操作している最中のメモリを使うトランザクションを延期または再試行させる方法もある．後者の方法は実装コストが大きいが，結合網を止める必要がなく，他のプロセッサも共有メモリへのトランザクションを続けられるため，多段結合網を用いる場合や実行時間のかさむ不可分命令を使用する場合には性能上有利である．

不可分命令の例としてTest & Set命令[19]（TS命令）を説明する．TS命令は命令で指定されるメモリの場所Lの内容を調べてその後にLに値1を書き込む命令である．調べた内容はプロセッサ内のフラグZに格納され，Lの内容が値0であればZはセットされ，それ以外の場合はリセットされる．Lに初期値0を格納しておき，複数のプロセスでTS命令を実行しても，唯一のプロセスしか値0を読み出す（ロック：lockを獲得するという）ことができず，Zフラグがセットされない．そこで，このZフラグがセットされたプロセスのみが排他的に実行すべき処理（クリティカルセクション：critical sectionと呼ぶ）を実行し，実行後にメモリLに値0を書き込む．TS命令でZフラグがリセットされた(ロックが獲得できなかった)プロセスは繰り返しTS命令を実行し，フラグのセットを確認した時点でクリティカルセクションを実行する．この処理の流れを図15(a)に示す．

プロセッサがキャッシュを持っていても，他のプロセッサからの排他性を保証するために，TS命令等の不可分命令に伴うメモリ操作は外部のキャッシュや主記憶へ伝達される．このため不可分命令は必ずプロセ

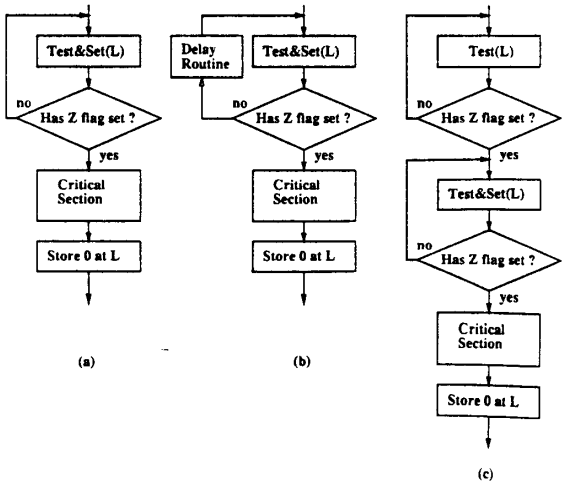

図15 Test & Set命令の使用法

ッサと主記憶の間の相互結合網を利用する必要がある．図15(a)のようにロックが獲得できなかったプロセスがTS命令を繰り返し実行すると，相互結合網のトラフィックを著しく増加させ，システムの性能を低下させることがある．この事態を防ぐ方法として，ロックを獲得できなかった場合は故意にしばらく時間を経過させた後にTS命令でロックの獲得を試みる方法(図15(b))，キャッシュを利用して普通のメモリ読み出し（キャッシュ内にコピーがあれば相互結合網は使用されない）でLの値が0であることを確認した後にTS命令を実行する方法(Test & Test & Set 図15(c))などがある[20]．

ただし，クリティカルセクションの処理量が大きい場合やクリティカルセクション実行中にプリエンプトされる可能性が多い場合，ロックを獲得するまで繰り返しTS命令でメモリを調べる処理はプロセッサ資源を浪費する．この場合は条件同期で述べるようにオペレーティングシステムを介した同期を行い，ロックが取れない際はプロセスを中断させて他のプロセスを実行させる方が効率が良い．

多数のプロセスが同時期にメモリの同一の場所に対して不可分命令を発効する処理では，不可分命令で操作するメモリへトランザクションが集中し，プロセッサとメモリ間の相互結合網を飽和させてしまい，他の共有メモリへのトランザクションが効率良く行われなくなる場合がある．多段結合網を持つシステムで，この事態を避けるために考案された方式にFetch & Addと呼ばれる方式がある[21]．メモリMを読み出して内容をプロセッサのレジスタにロードし，その内容に命令で指定された定数eを加えた値をMに書き戻すF&A(M, e)という不可分命令を用意する．今，2つのプロセッサP1とP2がF&A(M, e1)とF&A(M, e2)という命令を同時に行ったとすると，この2命令の実行後にMの内容（実行前の値m）はどちらが先に実行されてもm+e1+e2になる．そこで，多段結合網のスイッチに付加ハードウェアを設け，同一アドレスへのF&A命令を同時に受け取った場合に命令を結合（combining）する機能を加えることで，メモリの更新回数を減らすことができる．前記の例の場合，スイッチからはF&A(M, e1+e2)として次段のスイッチまたは主記憶に出力され，結合を行ったスイッチにメモリMの値mが返った時点で，スイッチがP1とP2への返り値（レジスタへロードする値）を作成してP1にm, P2にm+e1（またはP1にm+e2, P2にm）が返される．このように多段結合網と組み合わせることで，F&A命令は同期のためのメモリトランザクションを削減できる．また，F&A命令自身も使い道が広く，クリティカルセクションの確保の他にも，ループのイテレーションの分配や条件同期で述べるバリア同期のソフトウェアによる実現のために，Test & Set等の他の不可分命令よりも効果的に用いることができる．

4.2 条件同期

条件同期は他のプロセスに関する一定の条件が成立するまで処理を先に進めずに待ち合わせを行うものである．ソフトウェアで条件同期を行うには，共有メモリ上の同期変数を介して行うビジーウェイト（busy wait）方式とオペレーティングシステムを介して割り込みで行うシグナル（signal）方式が一般的である．ビジーウェイト方式は共有メモリ上に同期変数を取り，条件成立時に同期変数にある決められた値を書き込んでもらい，条件成立を確認する必要のあるプロセスはその同期変数を調べることで待ち合わせを行う．同期変数に決められた値が格納されていない場合は，その値が格納されるまで繰り返しその同期変数を調べ続ける．シグナル方式は同期変数の管理をオペレーティングシステムが行い，条件成立の通知や同期の問い合わせはプロセスからオペレーティングシステムへのソフトウェア割り込みで行われる．ビジーウェイトとの最大の差は同期待ちによるプロセッサ資源の浪費を避ける点にある．つまり，プロセスから同期の問い合わせがあり，その同期の条件がまだ成立していない場合，オペレーティングシステムは問い合わせを行ったプロセスを一時中断（サスペンド：suspend）してプロセッサの割当を解き，他のプロセスをそのプロセッサで実行させる．条件成立の割り込みが行われた時点で，その条件待ちで実行を中断しているプロセスを再起動する．ただし，シグナル方式はオペレーティングシステムへの処理の依頼や割り込みの利用を伴うため，同期処理のコストがかなり大きい．

共有メモリを介してのプロセス間通信では，プロセスは他のプロセスからのデータの送信の完了（共有変数への書き込みの終了）を自然に知ることができないため，必ず条件同期を伴う必要がある．粒度が荒い場合は，時間的にも共有変数の領域的にも大きな単位で通信を行うので，ビジーウェイト方式もしくはシグナル方式で受信側のプロセスに送信の完了を通知してもオーバヘッドは問題にならない．けれども，粒度が細かい場合は頻繁に小さな単位で共有変数を介した通信が行われるため，その通信のたびに条件同期が必要と

なり，オーバヘッドが問題になる．処理コストの比較的小さなビジーウェイト方式を利用しても，同期変数の操作はアクセスが主記憶や他のキャッシュにまで及び，メモリアクセスのレイテンシによるコストが問題になる．また，相互結合網のトラフィックも同期変数の操作で増加するため，結合網を飽和させてシステムの性能を著しく低下させる危険性もある．

前述のように，あるプロセス（生産者）が共有変数に計算結果を書き込み，他のプロセス（消費者）がその変数を読み出して利用する場合に必要になる条件同期を生産者—消費者型の同期と呼ぶ．この生産者—消費者型の同期を効率良く行うための機構として，Full/Empty ビット（F/E ビット）と呼ばれる1ビットの制御タグをメモリシステムのワードごとに設ける方法がある[22]．ワードへの書き込み前は F/E ビットが Empty に初期化されており，書き込みが行われると Full になる．プロセスが Empty 状態のワードを読もうとすると，その読み出しの実行は再試行または延期させられる．

生産者—消費者型の同期の他にバリア（barrier）同期と呼ばれる条件同期もよく用いられる．バリア同期は，並列処理で関連し合うプロセスのすべてがプログラム上のある地点に到達するまで互いに待ち合わせを行うものである．バリア同期は非常に強力な同期手段で，通常は必要以上に条件が厳しい．しかし，すべてのプロセスが待ち合わせ行い，バリア後はバリア前の処理がすべて終っていることが保証されるため，プログラミングは非常に楽になる．バリア同期はプロセスがバリア地点に到達するたびに，同期用の共有カウンタを排他的にカウントアップして，プロセス数に達することを調べることで容易に実現できる．しかし，この方法ではカウンタへの操作が逐次化され同期処理のコストが大きい．同期を取り合うすべてのプロセスが同時にプロセッサに割り当てられることが保証されていれば，バリア同期はハードウェアで簡単に実現でき，同期のための処理コストが大幅に削減できる[23]．さらに，プロセッサ間のバリア同期を2フェーズに拡張したり，カウンタの導入で拡張して，バリア同期の必要以上の厳しさを緩和しオーバヘッドを軽くする研究や，生産者—消費者型の同期への流用を可能にする研究も行われている[23)~25]．

5. 共有メモリ型並列計算機の高性能化

5.1 プロセッサスケジューリング

共有メモリ型並列計算機のメリットとして，その高い自由度と汎用性を挙げたが，プロセッサが完全に個別の資源として仮想化されていたのでは細粒度の並列性を活かした高速化は望めない．つまり，密に協調動作するプロセスは同時に実プロセッサに割り付けられていなければ，互いに無駄に同期待ちを行ったり，コンテクスト切替を繰り返したりしてオーバヘッドがかさむ．また，階層化された大規模並列計算機では，このようなプロセスは同一クラスタ内にプロセッサを割り当てられなければ性能が向上しない．このことから判るように，単一プロセッサシステムのときの1台のプロセッサを仮想化したプロセスを基準に実プロセッサをスケジューリングするのではなく，マルチプロセッサとしての仮想化つまり密に協調動作するプロセスをグループとしてまとめてスケジューリングする必要がある[23),26]．最近のオペレーティングシステム[26]ではこのようなプロセッサスケジューリングも徐々に採用され始めている．

5.2 静的コードスケジューリング

ノイマン型プロセッサの長所(短所でもあるが)は，制御の流れがコンパイル時に決定され，実行する命令が静的にスケジューリングされている点である．ノイマン型プロセッサはデータフロー計算機と比べると，多くの部分ではメモリ中の命令が順次に実行され，たとえデータ依存の条件分岐があっても過去の履歴に基づく分岐予測が的中する確率が高い．このことは1本の制御の流れしかないときでも，命令パイプラインで効率の良い実行が可能なことを示している．つまり，パイプライン中で命令を処理中に次の命令のフェッチや解釈を開始（命令の先行実行）できる．これに対してデータフロー計算機では処理が終了しないと次に実行すべき後続命令が決定しないため，パイプラインは直接依存関係のない処理で埋めるか他に実行すべきものがない場合はパイプラインのステージを空のままにせざるを得ない[27]．ノイマン型の並列計算機の場合は，コンパイル時に決められた本数の命令の流れを仮定してそこに命令を最適配置することで，非常に効率の良い実行コードを生成できる可能性がある(VLIW 計算機[注4)28]はこの実例と見做せる)．このレベルでは共有メモリを介するプロセッサ間通信はオーバヘッドが大きいので，VLIW 計算機のように共有マルチポートレジスタファイルを用意する必要がある．また，同期に関しては命令パイプラインのステージレベルで行われる軽い同期機構がサポートされなければならない．

5.3 キャッシュのコントロール

大規模な共有メモリ型並列計算機となると，メモリアクセスでキャッシュをミスした場合のペナルティ（レイテンシ）が大きな問題となる．できるだけキャッシュのヒット率を高めて高速化するためにキャッシュをコントロールする方法が考案されている．データをキャッシュへプリフェッチ注5)（prefetch）する方法[29]やデータのタイプによってスヌーププロトコールを動的に切替える方法[23),30)]やキャッシュを制御する命令を使う方法[31]等が考案されている．もちろん，プログラマがキャッシュの制御を指示することもできるが，使い易さを考えると，キャッシュの制御はコンパイラによる静的な最適化（メモリアクセスの最適化）の課題の1つである．

5.4 マルチスレデッドアーキテクチャ

メモリアクセスのレイテンシによる性能の低下を防ぐ方法として最近注目を浴びているものに，要素プロセッサにマルチスレデッド（multi-threaded）アーキテクチャを採用するという方法がある．複数のプロセッサコンテクスト（つまりレジスタセット）をプロセッサ内に用意してコンテクスト切替を高速化し，メモリアクセスでキャッシュミスが発生したり，プロセッサ間の同期待ちが発生した場合は，コンテクスト切替を行って他の命令流（スレッド：thread）を実行させる方式である[32),33)]．

6. むすび

共有メモリ型並列計算機の利点，共有メモリの構成法，同期機構，さらなる高性能化のための方向について解説を行った．今後も，共有メモリ型並列計算機はその自由度と汎用性のために広く利用されていくと思われる．高性能化への対応では，共有メモリモデルを保ちつつも，汎用性とトレードオフを取りながら，静的に最適化できる部分は最適化する方向に向かうと考えられる．本稿ではアーキテクチャのハードウェア的側面を中心に述べたため，あまり触れることができなかったが，この高性能化においては並列化コンパイラ（最適化コンパイラ）もしくは並列処理言語のコンパイラが静的にどこまで最適化できるかが鍵になると考えられる[34)~37)]．ハードウェアやオペレーティングシステムはこれらのコンパイラが容易に利用可能なオーバヘッドの少ない通信と同期の手段や機構を提供する必要がある．また，超並列計算機とか大規模並列計算機と呼ばれる範疇では，実装の階層構造を反映した階層構造をもったアーキテクチャの共有メモリ型計算機が実用化されるようになると考えられる．

参考文献

1) 富田：並列計算機構成論，昭晃堂（1986）
2) 富田，末吉：並列処理マシン，オーム社（1989）
3) 笠原：並列処理技術—マルチプロセッサシステムのハードウェアー，シミュレーション，8-2, 34/42 (1989)
4) T. Y. Feng: A Survey of Interconnection Network, IEEE Computer, 14-12, 12/27 (1981)
5) 黒川，相磯：並列処理の問題—結合方式—，情報処理，27-9, 1005/1021, (1986)
6) A. Agarwal, et al.: An Evaluation of Directory Schemes for Cache Coherence, Proc. 15th Int. Symp. on Computer Architecture, 280/289 (1988)
7) D. Chaiken, et al.: Directory-Based Cache Coherence in Large-Scale Multiprocessors, IEEE Computer, 23-6, 49/58 (1990)
8) D. V. James, et al.: New Directions in Scalable Shared Memory Multiprocessor Architectures: Scalable Coherent Interface, IEEE Computer, 23-6, 74/77 (1990)
9) K. Li: IVY: A Shared Virtual Memory System for Parallel Computing, Proc. 1988 Int. Conf. on Parallel Processing, 94/101 (1988)
10) U. Ramachandran, et al.: Coherence of Distributed Shared Memory: Unifying Synchronization and Data Transfer, Proc. 1989 Int. Conf. on Parallel Processing, 2, 160/169 (1989)
11) G. Delp: The Architecture and Implementation of MemNet: A High Speed-Shared Memory Computer Communication Network, Ph. D Thesis, Univ. Delaware (1988)
12) J. R. Goodman: Using Cache Memory to Reduce Processor-Memory Traffic, Proc. 10th Int. Symp. on Computer Architecture 124/131 (1983)
13) J. Archibald and J.-L. Baer: Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model, ACM Trans. Computer Systems, 4-4, 273/298 (1986)
14) P. Sweazey and A. J. Smith: A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus, Proc. 13th Int. Symp. on Computer Architecture, 414/423 (1986)
15) J. L. Hennessy and D. A. Patterson: Computer Architecture A Quantitative Approach, Morgan Kaufmann Pub., Inc. (1990)
16) D. Lenoski, et al.: The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor, Proc. 17th Int. Symp. on Computer Architecture, 148/159 (1990)
17) D. R. Cheriton, et al.: Paradigm: A Highly Scalable Shared-Memory Multicomputer Architecture, IEEE Computer, 24-2 33/46 (1991)
18) M. Raynal: Algorithms for Mutual Exclusion, The MIT Press (1986)
19) IBM Corp.: IBM System/360 Operating System Supervisor and Data Management Services, Form C28-6646, IBM Corp. (1967)
20) G. Graunke and S. Thakkar: Synchronization Algorithms for Shared-Memory Multiprocessors, IEEE Computer, 23-6, 60/69 (1990)
21) J. Edler, et al.: Issues Related to MIMD Shared-Memory

Computers: the NYU Ultracomputer Approach, Proc. 12th Int. Symp. on Computer Architecture, 126/135 (1985)
22) H. F. Jordan: Performance Measurement on HEP—A Pipelined MIMD Computer. Proc. 10th Int. Symp. on Computer Architecture, 207/212 (1983)
23) 松本：細粒度並列実行支援マルチプロセッサの検討，情報処理学会論文誌，**31**-12, 1840/1851 (1990)
24) R. Gupta: The Fuzzy Barrier: A Mechanism for High Speed Synchronization of Processors, Proc. 3rd Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 54/63 (1989)
25) 松本：Elastic Barrier：一般化されたバリア型同期機構，情報処理学会論文誌，**32**-7 886/896 (1991)
26) D. L. Black: Scheduling Support for Concurrency and Parallelism in the Mach Operating System, IEEE Computer, **23**-5, 35/43 (1990)
27) 坂井，平木，山口，児玉，弓場：データ駆動計算機のアーキテクチャ最適化に関する考察，情報処理学会論文誌，**30**-12, 1562/1572 (1989)
28) J. A. Fisher: Very Long Instruction Word Architectures and the ELI-512, Proc. 10th Int. Symp. on Computer Architecture, 140/150 (1983)
29) E. H. Gronish, et al: Compiler-Directed Data Prefetching in Multiprocessors with Memory Hierarchies, Proc. 1990 Int. Conf. on Supercomputing, 354/368 (1990)
30) T. Matsumoto, et al.: MISC: A Mechanism for Integrated Synchronization and Communication Using Snoop Caches, Proc. 1991 Int. Conf. on Parallel Processing, **1**, 161/170 (1991)
31) A. Goto, et al.: Processor Element Architecture for a Parallel Inference Machine, PIM/p, IPSJ Journal of Information Processing, **13**-2, 174/182 (1990)
32) R. A. Iannucci: Toward a Dataflow/von Neumann Hybrid Architecture, Proc. 15th Int. Symp. on Computer Architecture, 131/140 (1988)
33) W. Weber and A. Gupta: Exploring the Benefits of Multiple Hardware Contexts in a Multiprocessor Architecture: Preliminary Results, Proc. 16th Int. Symp. on Computer Architecture, 273/280 (1989)
34) F. Allen: An Overview of the PTRAN Analysis System for Multiprocessing, J. Parallel and Distributed Computing 5, Academic Press, 617/640 (1988)
35) M. Wolfe: Optimizing Supercompilers for Supercomputers, The MIT Press (1989)
36) C. D. Polychronopoulos: Parallel Programing and Compilers, Kluwer Academic Publishers (1988)
37) 本多，水野，笠原，成田：OSCAR 上での Fortran プログラム基本ブロックの並列処理手法，信学論，(D), J73-D-I **9**, 756/766 (1990)

脚 注

注1) （マルチプロセッサ）
日本語では小規模もしくは中規模の共有メモリ型並列計算機を指す用語として用いられることが多い．しかし，並列計算機全般を指す場合もある．本稿では，小規模の共有メモリ型並列計算機を指す．

注2) （プリエンプション）
時分割でプロセッサ資源を複数のプロセスで共用する場合，実行中のプロセスを一時中断させてプロセッサの使用権を他のプロセスに渡すことをプリエンプションという．

注3) （粒度）
複数のプロセスによる並列処理で，プロセス間通信（同期）から次のプロセス間通信（同期）までの平均処理量を粒度と呼ぶ．荒い（大きい）粒度の並列処理とはあまりプロセス間で通信や同期の行われない独立度の高い処理であり，細かい粒度の並列処理とはプロセス間で密に協調動作する処理である．

注4) （VLIW 計算機）
VLIW (Very Long Instruction Word) 計算機は複数の演算器を持ち，1つの長い（ビット数の多い）命令の中にそれぞれの演算器に対応する指令を格納するフィールドを持ち，各演算器にデータが共有レジスタファイルから供給され，この長命令を1ステップずつ実行する．各演算器とそれに対応する長命令のフィールドを独立したプロセッサと対応する命令と見做すことによって，VLIW マシンは1命令ごとにバリア同期を取る MIMD 型並列計算機と考えることができる．

注5) （プリフェッチ）
実際にデータが必要になる前に先読みして，必要になった時にすぐに参照できるように，データを手元に（この場合キャッシュに）置いておくこと．