

リング結合型並列計算機の 仮想共有メモリシステム

* ** *
中條 拓伯 和田 耕一 金田 悠紀夫

* 神戸大学 工学部 システム工学科
** 筑波大学 電子情報工学系

データベース処理を高速に行うためには、2次記憶装置の入出力バンド幅が重要な要素となる。ここでは、共有空間をセグメントに分割し、各セグメントを要素プロセッサ上のメモリに分散したような形態の仮想共有メモリシステムについて述べ、高いバンド幅を持つ2次記憶システムの実現について説明する。そして、それぞれのページ間において、無矛盾性を保証するためのプロトコルについて詳説する。

最後に、現在開発を行っているリング結合型並列計算機上での実現のためのメッセージパケットのフォーマットについて述べ、今後の課題について説明する。

A VIRTUAL SHARED MEMORY SYSTEM ON RING-CONNECTED PARALLEL COMPUTER

* ** *
Hironori NAKAJO Koichi WADA Yukio Kaneda

* Department of Systems Engineering,
Faculty of Engineering, Kobe University

** Institute of Information Science and
Electronics, University of Tsukuba

The input/output bandwidth of secondary storage device is one of the most important elements for fast database processing. In this paper, virtual shared memory constructed with distributed shared segments into each processing element are described, with the aim of the secondary storage with wide bandwidth. And we will explain the coherency protocol guaranteeing consistency among pages in this memory system. At last, we will describe the formats of message packets utilized in our RING-CONNECTED PARALLEL COMPUTER under development, and the future work will be discussed.

1. はじめに

近年、高性能なコンピュータへの要求がいろいろな分野において高まっている。その要求に応えるために、プロセッサ自身の性能を高める方向と並行して、多数のプロセッサを並列に接続し、システム全体の処理能力を高める並列処理マシンの実現がいろいろな機関で研究されている。データベースの分野でも、大量のデータを高速に処理するために高並列データベースマシンの研究がなされているが、H. Boral, D. DeWitt らは、「2次記憶装置の変革なしには高並列マシンの意味がなくなりつつある」と主張している^[1]。つまり、プロセッサ数がいくら増加しても、実際にデータが格納されている記憶装置との出入口が1つしかなければ、全体の処理能力の向上は望めないということを意味している。したがって、データベース処理を高速に行うためには、データが格納された記憶装置と処理を行うプロセッサとの関係を見直す必要がある。

従来、大容量のメモリシステムとしては、仮想メモリシステムが多くのマシンで採用されてきた。単一プロセッサの場合、多くのシステムで成功を納めてきたが、複数のプロセッサで大量のデータを共有するといったとき、磁気ディスク装置と各プロセッサとの間の通信路がボトルネックとなる。多数のヘッドを持ったディスク装置の開発も行われてはきたが、高並列マシンに対する効果は望めないであろう。そこで、システム全体として高いバンド幅を持った、大容量のメモリシステムを実現する必要が出てきた。

最近では、大容量の半導体メモリが安価に供給されるようになり、数百メガから数ギガにおよぶ半導体メモリを実装したマシンが出現するようになった。そこで、共有空間をセグメント単位に分割し、それぞれのセグメントを要素プロセッサ (PE) のメモリ上に分散させた形態の、高いバンド幅を持つ仮想共有メモリシステムを提案する。特にデータベースマシンでは、2次記憶の入出力バンド幅が性能を決定する。このことから、2次記憶装置と処理装置を分離した形態ではなく、システム全体にデータを供給する機能を、処理を行うPEに付加したような形態を持つ並列処理システムが有効であると考えられる。

ここでは、まず仮想共有メモリの基本的な概念について説明し、現在、開発を行っているリング結合型並列計算機^[2]上での実現について述べる。さらに、データの一貫性を保持するための無矛盾化プロトコルについて述べ、システムの可能性について考察する。

2. 仮想共有メモリシステム

並列処理システムにおいて、PE同士が協調して処理を進める場合、お互いにデータ交換を行う必要がなければ、それぞれのローカルメモリ内で処理を行い、台数に比例した処理能力の向上が望める。しかし、PE間で通信し合いながら処理を進める場合には処理能力の向上は通信の方法に依存する。一般的に通信方法はメッセージパッシングと共有メモリの2つの方法がある。メッセージパッシングは、おもに疎結合マシンで利用されるデータ通信方法で、CPU台数の拡張性は高いが、アプリケーション内でデータ通信操作を明示的に記述しなければならないことや、複雑なデータ構造を転送することが困難であるという問題点を持つ。

共有メモリは複雑なデータ構造もポインタを転送するだけで簡単にデータ通信を行うことができ、アプリケーションの記述性は高いが、共有バスや共有メモリ自身がボトルネックとなって、大規模なCPU台数に対しては実現は難しい。そこで、大規模なプ

ロセッサ数に対応するために疎結合でありながら、システム全体としては仮想的に共有メモリを有しているようなメモリシステムを考える。

疎結合ネットワークによる仮想共有空間の構成方法は、Yale大学のIVY^[3]などいくつか提案されている。そのほとんどが、仮想領域に対して、メインメモリとしての位置づけを考慮しており、CPU台数も小規模なものにとどまっている。しかし、大規模な並列処理システムのメモリシステムとして考えた場合、仮想共有領域も大容量なものとなり、メインメモリとしてよりも、むしろ2次記憶装置としての役割の方が有効であると考えられる。

IVYでは、仮想空間を構成する記憶素子は各PE上のみに分散されており、「読み出したデータは、最も最近書き込みが行われたページに存在する」というポリシーのもとに設計されている。したがって、すべてのPEは仮想空間の全ページに関する情報を保持する必要がある、どのページをアクセスしても、そのページを保持している所有者にたどりつけなければならない。例えば1Gバイトの仮想空間を1Kバイト単位のページに分割した場合、1Mページの情報を管理しなければならない。1ページあたりのステート情報にもよるが、全仮想空間を個々のPE上で管理することは困難である。

また、多数のページが1つのPEに集中し、メインメモリ容量を越えた場合は所有権を簡単に放棄することはできないため、そのPEの2次記憶装置に退避する必要がある。したがって、個々のPEが2次記憶を持たない場合には、他のPEに所有権を譲渡するといったことを行う必要があるため制御が複雑になる。特に高並列プロセッサシステムの場合、それぞれのPEに2次記憶装置を装備するのは、実装上の問題から困難であると考えられる。

そこで、仮想空間をセグメントに分割し、それぞれのセグメントは、システムを構成するPEのメモリ上に分散させる。セグメントはページ単位に分割し、ページ管理をそのセグメントを所有するPEに行わせるといった図1のような形態の仮想共有メモリシステムを考える。このように、セグメント内のページを管理するPE上のメモリ・コントローラ (MC) をセグメント管理者 (Segment Manager) と呼ぶ。

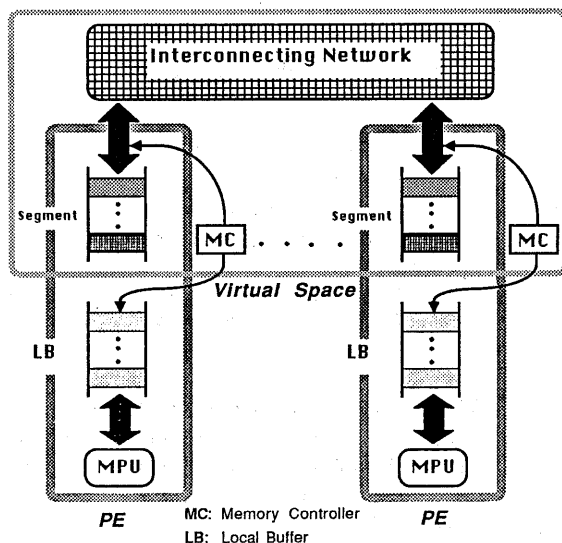


図1 仮想共有メモリシステム

本システムのように仮想空間を構成するセグメントを個々のPE上に分散させ、ローカル・バッファ内にそのコピーを転送し、処理を行うような場合、IVYなどのように共有空間が全く仮想的であるのに比べて、最悪時には倍のメモリを必要とすることや、セグメントとローカル・バッファ(LB)の内容の一致、不一致を考慮しなければならないなど、不利な点がある。しかし、実現性を考慮した場合、以下に示す利点を持つ。

(1) 管理テーブル

仮想空間のそれぞれのページの管理はそのページが含まれるセグメントの管理者が行う。したがって、仮想空間にアクセスしフォールトを起こした場合、所有者(Owner)を探索する必要は無く、セグメント管理者にアクセスを行えばよい。セグメント管理者は自分の所有するセグメント領域に関してのみ管理すればよく、管理テーブルは小容量で済む。

(2) ページング

ページフォールトを起こし、新たにページを読み込んだとき、ローカル・バッファメモリのページの中から最も不必要なものを選び、そのページを追いつ出した後に、ページを読み込む。したがって、1カ所に多数のページが集中するといった不均質な状態は存在しない。

(3) ハードウェア化

PE内において、セグメントとローカル・バッファメモリが分離していることによって、他のPEからそのセグメント上のページへのアクセスがあった場合に、PE上のMPUとは独立にページ転送を行えるようなハードウェアの実装が容易になる。すなわち、PE上のMPUから見た場合、提供したセグメントは2次記憶装置の一部とみなせることができるようになる。もし共有空間をIVYのような仮想空間にすると、PE上のMPUとアービトレーションを行う必要があるため、ハードウェアが複雑になる。

IVYなどの疎結合ネットワーク上の共有空間は、それを構成するページがどこかのPE上に存在し、空間自身は完全に仮想的であるのに対し、本システムには、実際に実メモリが存在することから、バス結合型マルチキャッシュシステムに類似している。しかし以下の点で異なる。

(1) ターゲット台数

共有バスシステムの場合、バス自身がボトルネックとなって、プロセッサ台数としては数十台までが限界である。本システムでは、それを越える台数でも性能を維持できることを目標としている。

(2) ページフォールトの場合のシステムの負荷

キャッシュの場合、ミスヒットしたときに、メインメモリからコピーを転送する必要がある。その負荷はヒットした場合の数倍で済むのに対し、本システムの場合、ネットワークを通じて、パケット通信によりページの読み込みを行う。したがって、ページフォールトを起こした場合の負荷は、ヒットした場合に比べて大きなものとなる。

(3) ページサイズ

本システムではフォールト時の負荷から、できるだけページフォールトが起こらないようにする必要があるため、ページサイズは、キャッシュのブロックサイズが数倍から数十バイトであるのに対し、数百バイトから数Kバイトとなる。しかし、ページサイズが大きくなるとパケットサイズも大きくなるため、ネットワー

ク上の転送時間が問題となる。最適なページサイズについては通信路のデータ転送速度、無矛盾化プロトコルなどに依存するため十分に検討する必要がある。

(4) 無矛盾化(コンシステンシー)プロトコル

マルチキャッシュシステムの場合、ライトスルー、コピーバック、ライトワンスなどのプロトコルが密結合マシンにおいて採用されているが、メインメモリとキャッシュ間の転送レートが高速であるという前提でのプロトコルとなっているので本システム上でもそのまま利用するには問題がある。無矛盾化プロトコルに関しては次章で詳述する。

3. メモリの無矛盾化プロトコル

仮想メモリ空間を共有する場合、同一ページのコピーが複数のPE上のローカル・バッファに存在する可能性が出てくる。そのページ上のデータに対して、リードアクセスに関しては問題無いが、ライトアクセスを行ったときに、データの統一性を保つ必要がある。つまり、あるPEが読み込んだデータは、最も最近変更があったページに含まれているということを保証しなければならない。このために、システム全体で制御のプロトコルを決める必要がある。

前述したように、各PE内にある共有セグメント領域とローカル・バッファ領域は、密結合マルチプロセッサのマルチキャッシュシステムにおける、メインメモリとキャッシュとの関係に類似している。そのため、無矛盾化プロトコルも多くのマルチキャッシュシステムで実用化されているものに類似したものとなる。本システムにおけるページ管理方法について述べる。

3.1 ページの状態

(1) ローカル・バッファメモリ上のページ

本システムにおける各ページの状態について考える前に、各ページの状態はマルチキャッシュシステムを参考⁽⁴⁾に、以下に示す4つの条件の組み合わせによって構成される。

1) 有効/無効(Valid/Invalid)

そのページが利用可能であるときValid、リードライトできないときInvalidとなる。有効なページへのアクセスはページヒット、アクセスページがローカル・バッファに無い場合をページフォールトと呼ぶ。フォールトを起こし、新たにページを読み込むことをページイン、最も必要度の低いページを共有セグメントに書き戻すことをページアウトという。

2) 一致/不一致(Clean/Dirty)

共有メモリとローカル・バッファの内容が一致している場合はCleanで、一致していない場合はDirtyとなる。共有空間が仮想的である場合には、これらの状態は存在しない。

3) 排他的/共有可能(Exclusive/Sharable)

他にそのページを持つPEが存在しないときExclusiveとなり、そのページのコピーを持つPEが存在する可能性がある場合はSharableとなる。特に、ライトアクセスの場合、Exclusiveなページにしか書き込みができないようにしておかないと無矛盾性が保てない。つまり、そのコピーが唯一のコピーであることを保証する必要がある。

4) ロック/アンロック(Lock/Unlock)

ページ内の連続した領域にバースト・アクセスを行う場合に、そのページをロックしておけば、途中で無効化されることは無く

なる。特に、共有データへのバースト・ライトアクセスを複数の P Eが行った場合、スラッシングが生じ、システムの性能を低下させる危険性がある。そのために、ページを Lock 状態にすることによってバースト・アクセスを保証する。

これらの組み合わせの内、Invalid と他の組み合わせを考慮する必要はない。また、共有可能なページに対する書き込みは Exclusive の状態にした後でないと許されないで、Dirty かつ Sharable という状態は存在しない。したがって、考慮すべき状態は、

Invalid	I
Valid / Clean / Exclusive / Lock	C E L
Valid / Clean / Exclusive / Unlock	C E U
Valid / Clean / Sharable / Lock	C S L
Valid / Clean / Sharable / Unlock	C S U
Valid / Dirty / Exclusive / Lock	D E L
Valid / Dirty / Exclusive / Unlock	D E U

の 7 つの状態である。

リードミスにより読み込んできたページは、他にコピーが存在しなければ C E の状態になるが、そのページに他の P E が読み込みを行った場合、直ちに C S に変更する必要がある。また、C E の状態のページへの書き込みは、そのページが唯一のコピーであることから、そのままライトアクセスを行い、その属性を D E に変更するだけでよい。そして、そのページをリードアクセスする P E があれば、その D E のページを共有セグメントに書き戻し、C S の状態にする。共有バスのマルチキャッシュシステムにおいて、それぞれのキャッシュがバスのアクセスを監視しているため、このような変更は一瞬で行われる。しかし、本システムでは、ネットワーク遅延を考慮して、C E と C S の状態をまとめて Read という状態とする。つまり、リードアクセスによりページフォールトをおこして、ページインを行ったときに、他にコピーがあっても無くても Read という状態に統一しておけば、C E から C S に変更するといったメッセージは不要となる。

ページフォールトを起こしたとき、リード、ライトのどちらのアクセスでフォールトを起こしたかは、制御信号で容易に判断できる。そこで、リードアクセスでフォールトを起こし、読み込んできたページには、前述した通り Read 属性とする。また、ライトアクセスによって取り込んできたページには Write 属性を与える。Write 属性は上記の D E の状態に相当する。以上をまとめると、本システムにおけるローカル・バッファページの属性は、

Invalid	I
Valid / Read / Lock	R L
Valid / Read / Unlock	R U
Valid / Write / Lock	W L
Valid / Write / Unlock	W U

の 5 つの状態を保持する。

(2) 共有セグメント上のページ

共有バスを持たないシステムにおいて、P E 台数が多数の場合、すべての P E のアクセスを監視することは困難である。したがって、キャッシュシステムにおけるメインメモリに相当する共有セグメントのそれぞれのページにも属性を付加し、管理する必要がある。これを利用して R U 属性のページへの書き込み時に、他のページの無効化を効率よく行える。共有セグメント内の各ページの属性は

Read
Write

の 2 つの状態を保持する。初期状態では、セグメント内のすべてのページは Read 状態にしておく。

3.2 ページ管理テーブル

共有セグメントの各ページはテーブルによって管理される。テーブル内には以下の情報が格納されており、P E からのアクセスを制御する。

(1) access フィールド

これは、そのページの Read/Write 属性を保持するフィールドである。

(2) peset フィールド

Read 属性を持つページは複数のコピーが存在する可能性がある。無効化などはページ管理者からそのページのコピーを持つ P E へ要求を発する必要がある。そこで、それぞれのページのコピーをどの P E が保持しているかを peset フィールドで管理しなければならない。

(3) lock フィールド

共有セグメント内の同一ページに同時にアクセスがあった場合、この lock フィールドを用いて排他制御を行う。

3.3 ページングの方法

各 P E はページに対して、リードヒット、ライトヒット、リードミス、ライトミス、ページアウトを行い、セグメント管理者はセグメント内のページを持つ P E に対して無効化を行う。それぞれのアクセスをまとめると以下ようになる。

(a) リードヒット

① ランダムリード

R U 属性ページに対してはそのまま読み込み、属性も変更する必要はない。W U 属性のページに対しては、そのページの内容は共有セグメント内のページと異なっているが、唯一のコピーであることから同様に読み込んでよい。

② バーストリード

R U 属性ページに対しては、Lock 命令により R L 属性に変更した後、バーストリードアクセスを行う。アクセス終了後は Unlock 命令により R U 属性に戻る。アクセスの途中に無効化要求があれば、アクセス終了後に I 属性に変更する。W U 属性ページに対しても同様に W L 属性に変更した後、バーストアクセスを行い、アクセス終了後に W U 属性に復帰する。また、アクセスの途中に後に述べるリードバック要求や、シフト要求があれば、アクセス終了後にそれらの要求に従う。

(b) ライトヒット

① ランダムライト

R U 属性ページに対しては、他のコピーをすべて無効化する必要がある。そのためにセグメント管理者に対して無効化要求メッセージ (Invalid Request Message) を送信する。セグメント管理者は、同一ページに対する同時アクセスの調停を行うために、要求を受け入れた P E に対してのみ、無効化要求アクノリッジ (Invalid Request Acknowledge) を返送する。

それから無効化メッセージ (Invalidate Message) を、peset フィールド内のすべての P E に送信した後、peset フィールドに

はライトアクセスを行った P E だけを登録し、共有セグメントのページは Write 属性に変更する。ライトアクセスを行った P E はローカル・バッファ内のそのページを WU 属性に変更し、そのページに書き込みを行うことができるようになる。WU 属性のページに対しては、そのページが唯一のコピーであるため、そのまま書き込みを行ってよい。共有セグメントとは内容が異なってくるが、ページアウトする必要があるまで属性を変更する必要はない。

② パーストライト

R U 属性ページに対しては、ランダムライトと同様に、セグメント管理者に対して、他のコピーの無効化を依頼する。ライトアクセスを行った P E はローカル・バッファ内のそのページを W L 属性に変更し、パーストライトアクセスを行う。アクセス終了後には Unlock 命令により WU 属性に復帰する。ロック中にセグメント管理者からリードバック要求、シフト要求があれば、アクセスを終了し、ロックを解除した後、それらの要求に答える。

WU 属性のページに対しては、そのページが唯一のコピーであるため、W L 属性に変更後、パースアクセスを行うことができる。R U 属性に対する場合と、無効化を行う以外は同様である。

(c) リードミス

ページフォールトを起こすために、セグメント管理者に対してリード要求メッセージ (Read Request Message) を送信し、ページインする必要がある。要求したセグメント内のページの属性によって以下のようにアクセスが異なる。

① Read 属性

この場合、セグメント内のページと、ローカル・バッファ内のコピーとは一致しているので、ページをそのまま転送すればよい。転送後、ローカル・バッファ内のそのページは R U 属性とし、以下リードヒットと同様の動作を行う。peset フィールドには要求した P E の I D が追加される。

② Write 属性

この場合、そのページはそのコピーを持つ P E によって書き換えられている。したがって、セグメント管理者はそのコピーの所有者に対してリードバック・メッセージ (Read Back Message) によって返送要求を行う。リードバック・メッセージを受け取った P E は、そのページをセグメント管理者とリード要求を発した P E に転送し、ページ属性を WU から R U に変更する。また、セグメント内のそのページ属性は Read に変更する。W L 属性の場合はパースアクセス終了後に R U に変更する。セグメント管理者は peset フィールドに、コピーを持つ 2 つの P E の I D をセットする。リード要求を行った P E はページを受け取った後、リードヒットアクセスを行う。

(d) ライトミス

多くのマルチキャッシュシステムではライトミスは、リードミスとライトヒットの組み合わせで実現されている。しかし、本システムでは、リードかライトのどちらのアクセスが行われたかを共有セグメント側でも管理しているため、直接セグメント管理者ヘライト要求メッセージ (Write Request Message) を発する。要求したページの属性によって、以下のようにアクセスが異なる。

① Read 属性

セグメント管理者は、そのページの peset フィールドに含まれるすべての P E に無効化メッセージを送信し、peset フィールドには、要求を発した P E の I D のみをセットし、その P E にページを転送する。ページ転送後、アクセス P E はそのページを WU 属性とし、以下ライトヒットアクセスを行う。

② Write 属性

セグメント管理者はコピーを持つ P E に対して、シフト要求メッセージ (Shift Request Message) を送信し、peset フィールドはライト要求した P E の I D をセットする。シフト要求を受け取った P E は、要求した P E にページを転送し、そのページを WU 属性から I 属性に変更する。ここでの特徴は、セグメント内のページとシフトするページは必ずしも一致させる必要がないという点にある。

(e) ページアウト

ページングを行っている内に、ローカル・バッファ内の未使用のページが無くなった場合、不必要なページをセグメント内にページアウト (Purge) する必要がある。どのページをページするかは L R U や F I F O などのアルゴリズムがある。このときページするページの属性によって動作が異なる。

R U 属性の場合、セグメント内のページと一致しているので、書き戻す必要はない。ただし、そのページの peset フィールドから、その P E の I D を削除する必要があるため、セグメント管理者にリードバージメッセージ (Read Purge Message) を転送して、その操作を依頼する。WU 属性の場合、セグメント内のページとは異なっているため、書き戻す必要がある。この場合、セグメント管理者にはライトバージメッセージ (Write Purge Message) を転送する。このメッセージは書き戻すべきページも含んでおり、このメッセージを受け取った管理者は peset フィールドを空にして、ページ属性は Read に戻す。

(f) 無効化

ページの無効化は、そのページが含まれるセグメントの管理者が行う。セグメント管理者は無効化メッセージ (Invalidate Message) を peset フィールドにある P E のうち、無効化要求を出した P E 以外のすべての P E に対して送信する。無効化メッセージを受け取った P E はページ属性を R U から I に変更するが、無効化に対するアクノリッジ信号は返送する必要はない。セグメント管理者が Write 属性のページを持つ P E に送信するシフト要求メッセージも一種の無効化メッセージであるが、要求があった P E へ転送要求する点で異なっている。以上の無矛盾化プロトコルを表 1 にまとめる。

表 1 無矛盾化プロトコル

Page Hit	Random Read	Burst Read	Random Write	Burst Write	Page Out
R U	→ R U	→ R L	(I R M) (I R A) → W U Invalid	(I R M) (I R A) → W L Invalid	(R P M)
W U	→ W U	→ W L	→ W U	→ W L	(W P M)
Fault					
I (Read)	(R R M) → R U	(R R M) → R L	(W R M) → W U → Write Invalid	(W R M) → W L → Write Invalid	-
I (Write)	(R B M) → R U → Read	(R B M) → R L → Read	(S R M) → W U Invalid	(S R M) → W L Invalid	-

IRM: Invalid Request Message RPM: Read Purge Message
RRM: Read Request Message WPM: Write Purge Message
WRM: Write Request Message SRM: Shift Request Message
RBM: Read Back Message Invalid: Invalidate other pages
IRA: Invalid Request Acknowledge

3.4 アクセスの調停

同一ページに対する同時アクセスの調停について考える。基本的にはセグメント管理者に到着したメッセージの順に処理が行われる。ページがヒットする場合の内、リードアクセスに対しては考慮する必要はないが、RU属性のページに対する同時書き込みに対しては調停が必要になる。ここでは、セグメント管理者が複数の無効化要求を受け取ったとき、受け入れたPEに対してのみ無効化要求アクトリッジを返送し、受け入れられなかったPEには、他の peset フィールドに含まれるPEと同様に無効化メッセージが転送される。したがって、無効化要求メッセージが転送したPEは、アクトリッジか無効化メッセージが返送されるまでライトアクセスを待つ必要がある。ページがフォールトした場合は、アクセスに従ってセグメント内のページ属性が変化するので、メッセージの到着が遅れたPEはその属性に従って処理される。

4. メッセージフォーマット

無矛盾化プロトコルで使用するメッセージをまとめると、以下のようになる。

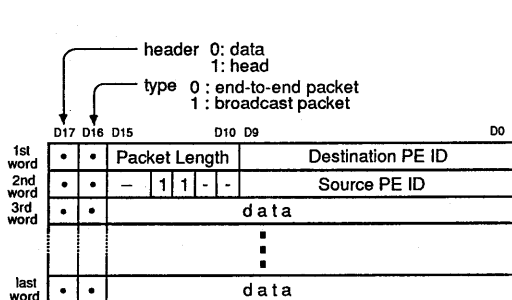
(1) 各PEからセグメント管理者へのメッセージ

- ① 無効化要求メッセージ (Invalid Request Message)
- ② リード要求メッセージ (Read Request Message)
- ③ ライト要求メッセージ (Write Request Message)
- ④ リードバージメッセージ (Read Purge Message)
- ⑤ ライトバージメッセージ (Write Purge Message)

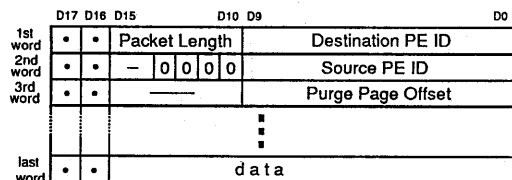
(2) セグメント管理者からPEへのメッセージ

- ⑥ 無効化要求アクトリッジ (Invalid Request Acknowledge)
- ⑦ リードバックメッセージ (Read Back Message)
- ⑧ シフト要求メッセージ (Shift Request Message)
- ⑨ 無効化メッセージ (Invalidate Message)

これらのフォーマットを図2に示す。



(a) ページデータメッセージ



(b) ライトバージメッセージ

5. おわりに

現在、我々はストリームコントローラと呼ばれる通信専用ハードウェアによってリング上に結合されたリング結合型並列計算機の開発を行っている。その上でのメモリシステムとして、メッセージ通信を行いながら、すべてのPEで仮想的に共有する2次記憶空間を構成するような仮想共有メモリについて述べ、データの無矛盾性を維持するためのプロトコルについて説明した。

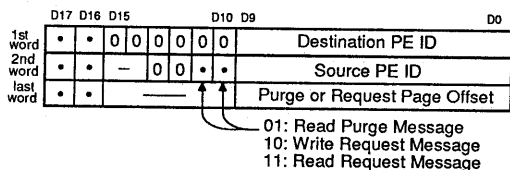
現在、ストリームコントローラにMMU (Memory Management Unit) としての役割を持たせるために、ハードウェアの追加、変更を行い、その詳細設計に入っている段階である。一刻も早く実機の完成を実現し、メモリ管理機構の実装を行い、さまざまな評価を行っていく予定である。

謝辞

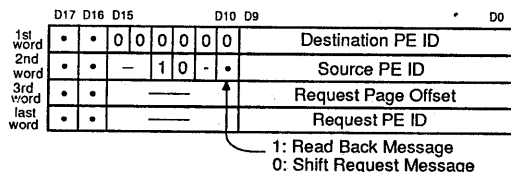
本研究の機会を与えていただいております神戸大学工学部システム工学科 前川禎男 教授に謹んで感謝致します。また、無矛盾化プロトコルなどに関して有益な助言をいただいた同学部、同学科 松田秀雄 講師に感謝致します。

[参考文献]

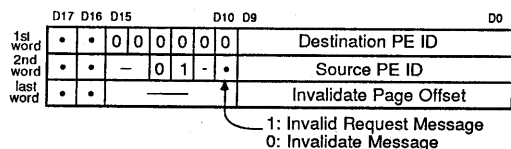
- [1] Boral, H. and Dewitt, D. J.: "Data Base Machines: An Idea Whose Time has Passed?", 3rd International Workshop on Database Machines, pp.166-187, Springer Verlag, (1983)
- [2] 中條, 和田, 金田: "並列知識ベースマシンのハードウェア構成", 並列処理シンポジウム JSPP'89, PP.147-154, (1989)
- [3] Li, K. and Hudak, P.: "Memory Coherence in Shared Virtual Memory Systems", ACM Transactions on Computer Systems, Vol.7, No.4, pp.321-359, (Nov. 1989)
- [4] 天野 英晴: "バス結合型マルチプロセッサ", 並列処理機構, pp.167-207, 丸善, (1989)



(c) リードバージ, リード/ライト要求メッセージ



(d) リードバック, シフト要求メッセージ



(e) 無効化, 無効化要求メッセージ

図2 メッセージフォーマット