ATMネットワークを用いた分散共有メモリ型分散処理システム

高橋雅史† 大庭信之‡ 小林広明† 中村維男†

†東北大学大学院 情報科学研究科
‡日本アイ・ビー・エム株式会社 東京基礎研究所

近年、情報処理速度向上の要求に対応するため、高性能化が著しいマイクロプロセッサを用いた分散処理の研究が広く行われている。本稿では、データ通信の分野で注目を集めている非同期通信モード (ATM) の技術を、分散処理システムの中でも既存の計算機技術との親和性の高い分散共有メモリ型のシステムに応用することを提案する。さらに、提案したシステムの有効性を検証するための試作ハードウェアの概要を示す。

1 はじめに

計算機の応用分野が広がるに従って、動画や音声の処理といったような従来と比較して格段にデータ量が多く、かつ、処理に許される時間が短い情報の処理が必要になった。このような状況下で、より高速に処理を行うことのできる計算機システムが求められるようになっている。これに答えるシステムの形態として、複数のプロセッサが協調して一つの処理を行うような、分散システム、あるいは、マルチプロセッサシステムの研究が広く行われている。

分散システムは複数の独立したプロセッサシステムがネットワークによって接続されたシステムであり、ネットワークの持つ拡張性に起因する高い拡張性を特長とする。しかしながら、ネットワークを通じての通信を意識する必要があるために、プロセッサの処理能力の有効利用に適した細粒度での並列処理プログラムの作成には困難が伴う。

一方、マルチプロセッサシステムは、一般に、プロセッサ間通信を容易にするような機構を備えており、細粒度の並列処理を行う上でのオーバーヘッドは小さくなる。しかしながら、バスをはじめとするプロセッサのインタフェースに特化した機構を要求するため、拡張性に乏しく、高価となる傾向がある。

そこで、本稿ではプロセッサバスをネットワークを用いて延長した分散システムを提案し、このシステムを実現するためにネットワークインタフェースが備えるべき機能と、ATM ネットワークを介して通信を行うためのプロトコルを示す。さらに、並列処理を行う場合に問題となるプロセッサ間の同期について、いくつかの

A Distributed Shared-Memory System Using ATM Networks
† Graduate School of Information Sciences, Tohoku University
‡ IBM Research, Tokyo Research Laboratory, IBM Japan Ltd.

実例を挙げて実現方法を示す。最後に、本稿で提案したシステムを評価するための試作システムの概要を示す。

2 システムの構成

2.1 概要

本システムでは、プロセッサから見たシステムアーキテクチャとして分散共有メモリアーキテクチャ[1]を、また、プロセッサ間を結ぶネットワークとしてATMネットワーク[2]を採用した。

分散共有メモリアーキテクチャは、物理的にはシステムに広く分散して配置されているメモリを一つのアドレス空間に配置したマルチプロセッサシステムのアーキテクチャである。このアーキテクチャはどの共有メモリに対してもプロセッサから同様な手順でアクセスすることを可能にするため、ソフトウェアの作成が他のアーキテクチャのシステムに比べて容易であるという特長をもつ。

ATMネットワークは、情報をセルと呼ばれる固定長の単位で伝送することを特徴とするネットワークであり、ハードウェアによる自己ルーティングを行うことで高スループットが実現できる。

2.2 モデル

本稿で示すシステムは、処理を行うプロセシングエレメント(PE)とデータを格納するメモリ(MEM)、それらの間を接続するATMネットワーク(ATMN)からなるモデルで表わせる(図1)。

PEには一つ以上のプロセッサと、これらが処理するデータの一時的な格納場所となるキャッシュ、および、そのPEにローカルなデータを格納するローカルメモリを含む。これらは共有アドレス空間へのアクセス要求を発生するクライアントとなる。一方、MEMはアクセス要求に答えてデータを供給するサーバである。このように、本システムはPEとMEMからなるサーバ・クライアント型のシステムである。

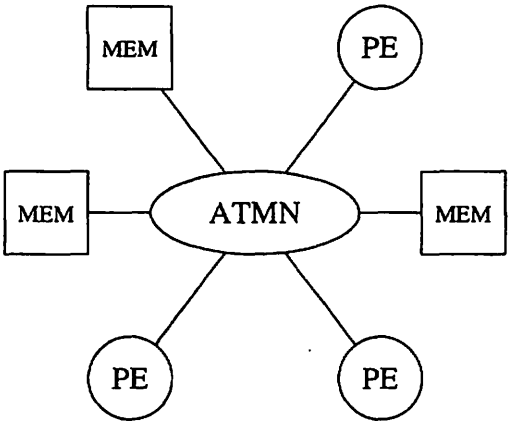

図1: System Model

PEとMEMにはそれぞれIDが割り当てられており、このIDを用いて要求・応答先の指定を行う。二つ以上のPEが同一のIDをもつことは許されておらず、これはMEMについても同様である。ただし、PEとMEM間のIDの重複は許される。同一IDを持つPEとMEMの組をノードと呼び、このときのIDをノードIDという。よって、ノードにはPEとMEMの両方、あるいは、どちらか一方が含まれることとなる(図2, 図3)。同一ノードに含まれるPEとMEMはバスで結ばれ、PEはMEMを高速に参照することができる。また、ノードにはバスとATMNの間の差違を吸収するためのネットワークインタフェース(NIF)も用意される。

ネットワークはノード・グループ・メタグループという三階層で管理される。物理的に距離が小さく、かつ、ネットワークだけでなく後述するキャッシュの一貫性を保つための補助ハードウェアによっても接続されているノード群をグループ呼ぶ。さらに、ある程度短いターンアラウンドタイム内に応答を得ることを期待できる

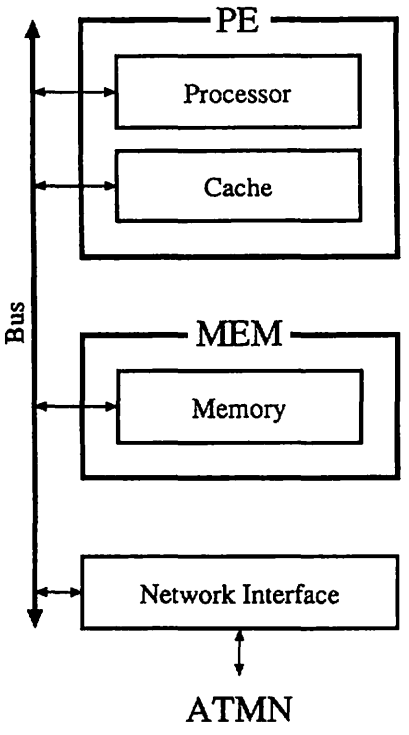

図 2: Single Processor Node

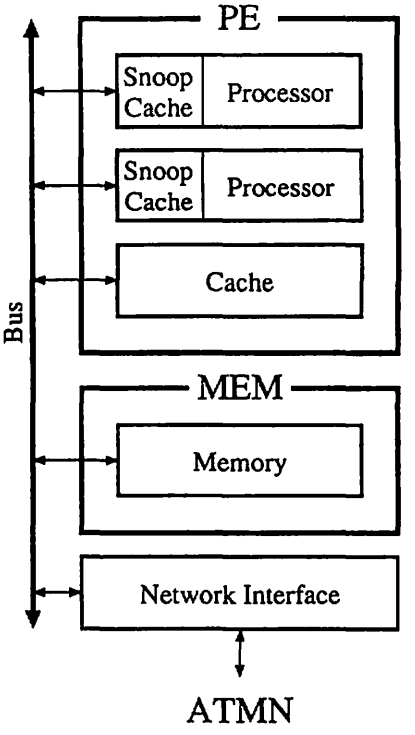

図 3: Multi Processor Node

ようなグループ群を一つのメタグループとする(図 4)。本稿で提案するプロトコルは、現時点では、同一グループ内に属するノード間の通信を対象にしているが、このように階層化されたネットワークへの拡張も考慮している。

3 ネットワークインタフェース

本稿で提案するシステムでは ATM ネットワークを用いてプロセッサバスの延長を行う。しかし、当然のことながら、バスと ATMN の間には差違が存在する。そのため、NIF はこの差違を吸収する機構を持たなければならない。本節では機能・性能の両面から、バスと ATMN を比較し、NIF に要求される機能を明らかにする。また、NIF 間のデータ転送のプロトコルの概略について述べる。

3.1 バスと ATMN の比較

機能上の差違 データの受け渡しは、バスでは制御線を使ったハンドシェークによって行われる。一方、ATMN ではセルという情報のかたまりを一方的に送り出すことで行われる。PE 内のキャッシュのラインサイズを適切な大きさに設定し、キャッシュのロード、フラッシュという操作により MEM へのアクセスを行うことで、通信データのセル化を実現する。また、ハンドシェークによる確実なデータ転送を行うために、要求セルと応答セルの組による転送を

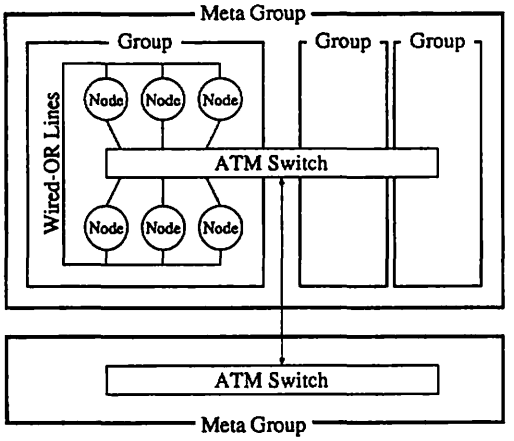

図 4: Network Hierarchy

行う。

ATM ではある程度のセル損失を認めることで、高速なネットワークの実現を容易にしている。しかし、セルが失われた結果、プロセッサにデータが届かなければ致命的な障害が引き起こされるため、何らかの方法で損失の検出と回復を行わなければならない。一般に、セルの損失が起こる確率は非常に小さいので、回復に要する時間が大きくとも、全体の性能に与える影響は小さい。そこで、本システムでは応答時間の監視による損失の検出という単純な方法を採用した。

以上の通信シーケンスについての詳細は性能上の差違への対応方法とともに 3.3で述べる。

性能上の差違 物理的に近距離にある要素を結ぶバスと比べると、ある程度の距離がある要素を結ぶネットワークはスループットやターンアラウンドタイム、エラーレートなどの性能で劣ることが多い。幸い、ATMN は特に高いスループットを持つネットワークであり、この点についてはバスと遜色がないが、ターンアラウンドタイムとエラーレートについては何らかの対応をとる必要がある。本システムではバースト転送、および、誤り検出情報の付加によってこれらの欠点の影響を軽減している。

3.2 セルの構造

ATM のセルは 53 バイトの大きさを持つ。このうち、先頭の 5 バイトをヘッダ、残りの 48 バイトをインフォメーションフィールド (IF) という。ヘッダは交換網が利用する部分であり、データ転送には利用できない。そこで、PE と MEM の間の通信に必要なデータはすべて IF に格納することになる。セルの構造を表 1に示す。各行の左欄の数値はセルの先頭バイトからのオフセットであり、バイトを単位に示している。

表 1: Cell Structure

0–4	Header
5–51	Information field
5	Cell type
6	Block size
7	PE meta-group ID
8	PE group ID
9, 10	PE node ID
11	MEM meta-group ID
12	MEM group ID
13, 14	MEM node ID
14–17	Address data
18–49	Memory data
50, 51	CRC

Cell type はセルの種類を示すフィールドであり、その値と種類の関係は表 2のようになる。ここで、request は PE から MEM への通信、acknowledge は MEM から PE への通信を意味する。Test and set request は MEM に対してテストアンドセットのアトミックオペレーションを要求する。また、interrupt request は PE から MEM ID で指定されたノードにある

PE への割り込み発生要求である。このフィールドの最上位ビットは通常 0 であるが、再送要求セルとそれに対応する応答セルに限り 1 となる。

表 2: Cell Type

0	read request
1	read acknowledge
2	write request
3	write acknowledge
4	test and set request
5	test and set acknowledge
16	interrupt request
17	interrupt acknowledge

Block size はメモリへのアクセス単位の大きさを示すフィールドであり、下位 4bit で連続して転送するセルの数を示す。とりうる値は 0 から 15 であり、この値を n として、2^n ラインを連続して転送する。上位 4bit については現時点では規定していない。このフィールドの存在により、連続領域に対する複数の read request を、一つの read request にまとめることが可能になり、MEM の連続する領域からキャッシュへのデータ転送の効率が高まる。

PE meta-group ID, PE group ID, PE node ID をまとめて PE ID、MEM meta-group ID, MEM group ID, MEM node ID をまとめて MEM ID と呼び、それぞれ、そのセルの通信に関係する PE と MEM を特定する。

Memory data には MEM から PE、あるいは、PE から MEM への転送データが格納される。この部分の情報が使われない場合は不定値になる。このフィールドの大きさは 32 バイトであり、PE が持つキャッシュのラインサイズと同一である。

Address data はそのセルによって操作される領域の、それぞれのノードにおけるオフセットアドレスを格納する。MEM へのアクセスをライン単位で行うため、このフィールドの下位 5 ビットは無視される。MEM ID の 32 ビットで MEM をもつノードを特定し、さらにオフセットアドレスを 32 ビット与えるため、本セルフォーマットが扱う共有アドレス空間の大きさは 64 ビットとなる。

CRC は誤り検出のためのフィールドであり、IF の CRC 情報が格納される。

3.3 転送シーケンスの概要

読み出し MEM の内容を読み出してキャッシュに取り込むシーケンスである。MEM は要求セルを受け取ると、応答セルの memory data に指定アドレスの内容を格納して送出する。

書き込み PE のキャッシュ内の内容を MEM に書き出すシーケンスである。MEM は要求セルの memory data を指定アドレスに書き出し、応答セルを送出する。

テストアンドセット 不可分なテストアンドセットの操作を行うシーケンスである。要求を受けた MEM は、操作対象となるラインの最初の 1 バイト (第 0 バイト) の最下位ビットが 0 であれば、第 0 バイトのすべてのビットを 1 にするとともに、要求を発生した PE ID を第 4 バイトから第 7 バイトの範囲に書き込む。第 0 バイトの最下位ビットが 0 以外の場合にはラインの内容の変更は行わない。以上の操作を MEM に対して行った後、応答セルに変更前のラインの内容を格納して送出する。

割り込み PE と PE との直接の通信を行うためのシーケンスであり、MEM の内容には影響を与えない。このシーケンスに限り、MEM ID を割り込みを受ける PE の指定に用いる。割り込み要求を受け取った PE は応答セルを送出するとともに、PE 内のプロセッサに割り込みを発生する。プロセッサへの割り込みが禁止されている場合でも応答セルは送出される。

再送処理　セルのヘッダ部分とIFにはそれぞれ、誤り検出のための情報が用意されおり、これを用いて誤りの検出を行う。ヘッダに誤りが検出された場合には誤り訂正が行われる。一方、IFに誤りが検出された場合にはそのセルは破棄され、再送処理に移行する。また、ATMN中でのセルの損失は応答時間を監視することで実現される。予想される応答時間を越えてセルが到着しなかった場合には再送処理が行われる。

要求セルは再送か否かにかかわらずMEM(割り込み要求の場合はPE)によってつねに受け付けられるが、通常の応答セルに続く再送応答セルは破棄される。これは、時間切れで再送処理に移った後に、応答セルが到着するという状況に対応するためである。

テストアンドセット再送要求セルを受けたMEMは指定アドレスからの内容をそのままmemory dataに格納して応答セルを送出する。ただし、要求を出したPEのPE IDが操作対象となるラインに格納されているPE IDと一致した場合に限り、memory dataの最初の1バイトを0にセットしてから応答セルを送出する。

4　キャッシュ

4.1　キャッシュの構造

キャッシュはラインとブロックという2つの単位で制御される。ラインはATMセルのmemory dataフィールドと同じ大きさであり、それぞれaddress tagとvalid bit, dirty bitを持つ。ブロックはいくつかのラインの集合であり、read requestへの応答としてバースト転送されるデータの大きさに一致する。ただし、バースト転送で送られるライン数は要求セルのblock sizeによって変化するため、ブロックの大きさも可変である。

4.2　一貫性の保証

PEのキャッシュはコピーバック型のプライベートキャッシュであり、ATMN上の通信量の減少と、プロセッサからみたMEMの平均アクセス時間の短縮を目的とする。しかし、本システムはマルチプロセッサシステムであるためキャッシュの内容と、MEMの内容が一致しない状況が生じる。キャッシュとMEMの不一致がある状況は様々な不都合を生じるため、何らかの方法で一致した状態に回復させる必要がある。本システムにはこの不一致を検出して、自動的に解消するためのハードウェア[3]は用意していない。

不一致が問題になるのはスピンロックやバリア、セマフォアといった「きわどい領域」の制御に関係する命令の実行時に参照される領域と、プロセッサ間のデータの受け渡しに使われる領域である。しかしながら、これらの領域の一貫性が常に保たれている必要はない。まず、前者の場合は特定の命令が実行される時刻に一貫性が保証されれば十分である。後者の場合は受け取り側のプロセッサによってデータの参照が行われている期間、継続して一貫性が保証されていなければならない。しかし、この期間に他のプロセッサがデータの更新を行うことは不自然であることを考えると、受け渡しを行うある時刻に一貫性が保たれていれば問題がないことになる。つまり、このどちらの場合においてもある特定の時刻に一貫性を保証すればよく、また、その時刻をソフトウェア側から指定することは容易である。よって、ある特定の時刻に一貫性を保証するようなしくみが用意され、かつ、ソフトウェアによって制御可能であれば何ら問題がないことになる。

そこで、本システムでは、不一致を解消するためのハードウェアの機能を用意し、それらをソフトウェアによって制御することで、これらの領域の一貫性を保証することにする。

ある時刻の一貫性を保証するための操作とし

て、read through, write through を用意する。プロセッサが read through を指定してメモリ読み出しを行うと、たとえ、キャッシュがヒットした場合でも MEM へのアクセス要求を発生し、最新の MEM の内容をプロセッサに渡す。また、write through を指定してメモリ書き込みを行うと、キャッシュの内容を更新したのち、直ちにキャッシュの内容を MEM に書き出す。しかし、スピンロックなどをこの機能を用いて実現すると、MEM への参照が頻繁におこるためネットワークトラヒックの増加を招く。そのため、このような操作には割り込み機能を用いるか、あるいは、4.3で述べる補助ハードウェアを利用することが望ましい。

キャッシュへの特殊な操作としては read through と write through の他に、将来的に必要になるデータをあらかじめキャッシュに取り込むための touch load と、MEM からの読み出しを行わずにキャッシュラインを有効にする force validation、不可分なテストアンドセット処理を行うための test and set を用意する。

4.3 補助ハードウェア

4.2に示した機能だけを用いた場合、ある PE が MEM の内容を更新したことを他の PE が知るには、その MEM の内容を read through によって取り出して確認する必要がある。しかし、スピンロックのように継続して監視を行うような処理を read through で実現すると、ネットワークトラフィックの増加をまねくことになる。そこで、これを回避するための補助的なハードウェアを用意し、ネットワークへの付加を軽減する。このハードウェアの構造は一本の信号線に各 PE の出力を wired-OR で接続し、同時に各 PE がその信号線のレベルを参照できるようにしたものである。

変更を行う PE は write through で MEM の内容を更新した後、信号線への出力を一定期間アクティブにする。参照を行う PE は read through で MEM を監視するかわりにこの信号線のレベルを監視しており、この信号線がアクティブになった時点で初めて MEM への read through を行う。このとき既に MEM の内容の変更は完了しているため、PE は変更後の MEM の内容を受け取る。参照側の PE は (割り込み処理などの理由によって) アクティブ期間中に信号線の参照が行えなかったことに備えて、適当な周期で MEM に対する read through を行う。この周期的な read through の導入は、信号線への信頼性の要求を著しく軽減する。なぜなら、アクティブ状態の検出に失敗したとしても、次の read through までの無駄な時間が大きくなるだけであり、また、誤ってアクティブ状態と認識しても、無駄な read through が発生するだけである。しかも、どちらの場合にも一貫性は保証され、また、処理対象となるデータは (エラー対策がなされた) ネットワークを経由して伝えられるために致命的な障害は発生しない。さらに、割り込み処理や CPU 時間待ちなどの検出失敗の原因となる事象が発生する状況であれば、無駄時間も問題にならないと考える。この信号線の構造は非常に簡単なものであり、信頼性も要求されないために、必要ならば多数用意することも容易である。この場合、複数の信号線を使って、例えば、ハンドシェイクのような制御を行うことも可能である。

5 プロセッサ間の同期

本節ではプロセッサ間の同期の基本操作となるスピンロック、バリアの実現方法について説明する。

5.1 スピンロック

スピンロックは共有領域を複数のプロセッサで排他的に利用するための機構であり、それぞれのプロセッサで実行される処理の流れはリスト 1 のようになる。本システムのような、自動

的な一貫性の保持能力を持たないキャッシュを持つ分散共有メモリ型のシステムにおいて、この処理を実行するには次の二つの問題を解決する必要がある。

- lock の開放 (10 行) が lock を参照 (2 行) しているプロセッサに伝わらなければならない
- lock の開放を確認 (2 行) から lock の変更 (3 行) まで他のプロセッサの lock の参照を許してはならない

これらの問題は lock の確認と変更、および、開放を、それぞれ、test and set, write through によって行えば解決される。しかし、この方法では lock が確保されている間、多数の test and set request セルが ATMN に流れ込むようになる可能性がある。そこで、本システムでは 4.3 で示したような補助線を用意することでリスト 2 のような処理を可能にしている。ここで、

- line(番号) の参照と代入は、それぞれ、番号で指定された補助線が接続される入出力ポートへの読み出しと書き込みを意味する
- hash(変数) は変数に対応する補助線の番号を返す多対一の写像関数である
- sleep は適当な時間、処理を休止する命令である
- count は各プロセッサに独立な変数である
- COUNT は適当な整数値である

となる。この処理で多数発生する処理は補助線への参照であり、この参照は各 PE 内で独立に処理できるため、ATMN のトラフィック増加は生じない。また、一本の補助線について複数のロック変数を割り当てられるため、ロック変数の数が補助線の数によって制限されることがない。このスピンロックアルゴリズムでは補助線が ACTIVE になった時刻に多数のトラヒックがロック変数を持つ MEM に集中する可能性がある。しかし、より改良されたアルゴリズム[4]を基本として、リスト 1 からリスト 2 への変更と同様な変更を加えることでトラヒックは分散できる。

リスト 1: Simple Spin-Lock

```
 1  wait:
 2    if lock = FREE then
 3      lock := LOCKED
 4      goto exit_wait
 5    else
 6      goto wait
 7    endif
 8  exit_wait:
 9    process
10    lock := FREE
```

リスト 2: Improved Spin-Lock

```
 1  wait:
 2    if lock = FREE then
 3      lock := LOCKED
 4      goto exit_wait
 5    else
 6      count := 0
 7  sense:
 8      if line(hash(lock)) = INACTIVE
              and count < COUNT then
 9        count := count + 1
10        goto sense
11      endif
12      goto wait
13    endif
14  exit_wait:
15    process
16    lock := FREE
17    line(hash(lock)) := ACTIVE
18    sleep
19    line(hash(lock)) := INACTIVE
```

5.2 バリア

バリアはすべてのプロセッサがある段階に達するまで、処理が終了したプロセッサを待機させる機構であり、それぞれのプロセッサで実行される処理の流れはリスト 3 のようになる。こ

こで、PROCESSORS はこのバリアに関係するプロセッサの数であり、local は各プロセッサに独立な変数である。

本システムでこの処理を実行するには次の二つの問題を解決する必要がある。

- barrier の変更 (1 行) が barrier を参照 (2 行, 8 行) しているプロセッサに伝わらなければならない
- barrier の変更 (1 行) から参照 (2 行) までの間、他のプロセッサの barrier の変更を許してはならない

これらの問題は test and set 命令で barrier の変更権を確保してから write through によって変更を行い、また、barrier の参照を read through によって行えば解決される。しかし、待ち状態のプロセッサが増えるに従って、多数の read through セルが ATMN に流れ込むようになる可能性がある。そこで、補助線を利用してリスト 4 のような処理を行う。補助線への参照は各 PE 内で独立に処理できるため、ATMN のトラフィック増加は生じない。また、補助線が ACTIVE になった時刻に多数のトラヒックがカウント変数を持つ MEM に集中することは、より改良されたアルゴリズム[4]を基本として、リスト 3 からリスト 4 への変更と同様な変更を加えることで解消できる。

6 試作システムの概要

本稿で提案したシステムの有効性を検証し、また、ソフトウェアのインプリメントを行うための試作システムの構築を行っている。第一段階のシステムは 4 つのノードを 4 入力 4 出力のクロスバ ATM スイッチで結んだ構成をもつ。ノードとスイッチの諸元を表 3 に示す。

ノードのベースにはパーソナルコンピュータのマザーボードを利用し、これに、MEM とキャッシュ、ネットワークインタフェースを搭載したカードを実装する形態をとっている。このような構成により、高性能な PE を確実に、また、安価に実現できる。ネットワークインタフェースカード上にはキャッシュと共有メモリのための 8Mbits の S-RAM を 4 個、ATMN との間のバッファとアドレスタグのための 256Kbits の高速 S-RAM を 4 個、キャッシュの制御を担当する大規模 PLD を 6 個実装する。大規模 PLD は基板上に実装したままでプログラムの変更が可能であり、キャッシュの制御方法をさまざまに変更して評価を行なうことが可能である。また、第二段階のシステムとして、光ファイバと VESA ローカルバスの組み合わせによるより高速なシステムの設計を行なっている。

リスト 3: Simple Barrier

```
 1   barrier := barrier - 1
 2   local := barrier
 3   if local = 0 then
 4     barrier := PROCESSORS
 5     goto exit_wait
 6   else
 7 wait:
 8     if barrier > local then
 9       goto exit_wait
10     else
11       goto wait
12     endif
13   endif
14 exit_wait:
```

表 3: Prototype Features

Node	
CPU	i486DX2-66MHz
Local Memory	16Mbyte
Shared Memory	1Mbyte
Cache Memory	1Mbyte
ATM Switch	
Throughput	8Mbyte/sec
Configuration	4 × 4 Crossbar

リスト 4: Improved Barrier

```
 1   barrier := barrier - 1
 2   local := barrier
 3   if barrier = 0 then
 4     barrier := PROCESSORS
 5     line(hash(barrier)) := ACTIVE
 6     sleep
 7     line(hash(barrier)) := INACTIVE
 8     goto exit_wait
 9   else
10 wait:
11     if barrier > local then
12       goto exit_wait
13     else
14       count := 0
15 sense:
16       if line(hash(barrier)) = INACTIVE
             and count < COUNT then
17         count := count + 1
18         goto sense
19       endif
20       goto wait
21     endif
22   endif
23 exit_wait:
```

7 おわりに

本稿ではプロセッサバスを ATM ネットワークを用いて延長した分散共有メモリ型分散処理システムを提案し、そのシステムの構成を示した。

分散共有メモリ型の計算機アーキテクチャを採用することにより、プログラムの作成が容易になる。分散共有メモリ型のシステムで大きな問題となるキャッシュの一貫性の保持方法については、一貫性を保つ必要がある時間が全体の処理に要する時間に対して小さいと仮定して、ソフトウェアによる解決を若干のハードウェアによって補助する方法を選択した。

また、バスの延長のために特別に設計したネットワークを用いず、汎用のネットワークである ATMN を用いることで、安価でかつ高性能なシステムの実現が期待できる。さらに、標準的な ATMN の規格にあわせてネットワークインタフェースを設計すれば、ネットワーク部分をブラックボックスとして扱うことができるようになる。よって、新たに開発された ATM の技術を ATMN の置き換えだけでシステムの高性能化に利用できるようになる。

今後の課題として、本稿で提案したシステムがより実用的なものにするために、ソフトウェアによる一貫性の保証をプログラム作成者に意識させないコンパイラの開発が不可欠である。また、より大規模なシステムへの対応、具体的には

- グループを越えた規模のシステムの制御方法
- 一部のノードが機能を果さなくる状況への対応

についても考える必要がある。

参考文献

[1] P. Stenström, "A Survey of Cache Coherence Schemes for Multiprocessors," *IEEE Comput.*, Vol. 23, No. 6, June 1990, pp. 12–24.

[2] S. E. Minzer, "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Commun. Mag.*, Vol. 27, No. 9, Sept. 1989, pp. 17–25.

[3] D. Lenoski, *et al.*, "The DASH Prototype: Logic Overhead and Performance," *IEEE Trans. Parallel Distributed Syst.*, Vol. 4, No. 1, Jan. 1993, pp. 41–61.

[4] J. M. Mellor-Crummey, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors," *ACM Trans. Comput. Syst.*, Vol. 9, No. 1, Feb. 1991, pp. 21–65.