計算機アーキテクチャ　128－19
ハイパフォーマンス
コンピューティング　70－19
（1998．3．6）

ディレクトリキャッシュを付加した通信機構による仮想共有メモリの支援

滝田　裕[†]　田中英彦[†]

WS/PC クラスター上での分散共有メモリは一般に仮想共有メモリによって実現されている。仮想共有メモリでは特にハードウェアを必要としないが、ソフトウェアによる共有ノードディレクトリの管理とノード間通信起動のオーバーヘッドにより性能の低下を招いている。そこで本稿では、マルチスレッド化された並列アプリケーションのメモリアクセスパターンの解析から共有ノードディレクトリのキャッシングの有効性を示し、それに基づく共有ノード管理機構を備えた共有メモリ専用通信インターフェースによって WS/PC クラスター上での仮想共有メモリの共有メモリアクセス性能の向上を図る手法について提案する。

Supporting Virtual Shared Memory using the Network Interface with Directory Caching

HIROSHI TAKITA[†] and HIDEHIKO TANAKA[†]

Many WS/PC clusters implement the distributed shared memory(DSM) based on the shared virtual memory(SVM). A SVM system doesn't need any additional hardware, but it suffer from bad performance becase of the software overhead of coherence management and inter-nodes communications.

We show that the shared nodes directory cache is effective, based on analysis of the multi-threaded applications' memory access pattern. In this paper, we suggest the improvement of SVM using the communication interface with the shared nodes directory cache system.

1. はじめに

単体の WS/PC の性能向上により、LAN によって WS/PC をつないだ WS/PC クラスターが安価な並列計算機として普及しつつある。またプログラムの並列化手法として、従来のメッセージ通信によるものよりも修得の容易な、共有メモリを前提とした multi thread による並列化も一般化している。そこで WS/PC クラスター上での共有メモリ実現に関する研究が盛んに行われている。

従来の WS/PC クラスター上の共有メモリは、主に OS の仮想記憶管理機構を利用した仮想共有メモリによって実現されてきた。この手法では、特にハードウェアを追加する必要がなくソフトウェアのみで実現できるが、

- 共有メモリアクセス検出から通信まで、全てソフトウェアで行うため共有メモリへのアクセスに時間がかかる。
- ページ単位共有による false sharing によって、不要な通信が増大する。
- WS/PC で使用されている LAN を用いるため、ノード間通信のスループットが低い。
- 他のノードの共有メモリアクセスによる共有ノード情報の更新のため、データを共有している全ての CPU に割込みがかかる。

という理由により、全体の性能を悪化させていた。

本稿では、マルチスレッド化されたアプリケーションの共有メモリアクセスに関する解析を基に、仮想共有メモリ支援機構としてディレクトリキャッシュ機能を持つ専用の通信インターフェースを設け、性能の向上を図る手法を提案する。

2. 仮想共有メモリ

2.1 構　成

仮想共有メモリはハードウェアには一切手をつけず、OS の仮想記憶機構を改変することで、仮想アドレスでの共有メモリを実現するものである。

一般の計算機で OS がおこなっている仮想記憶管理は、図 1 に示す通りである。仮想共有メモリでは、

† 東京大学大学院工学系研究科情報工学専攻
Information Engineering Course, Graduate school of Engineering, The University of Tokyo

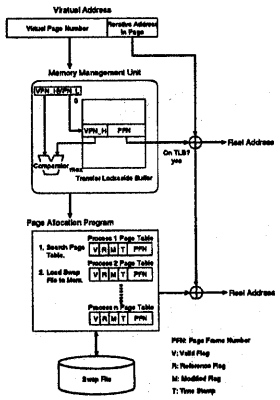


図 1　一般の仮想メモリ機構

ディスクへのスワップのかわりに、ネットワーク上の他のプロセッサのメモリへデータが保管されているものと考えることができる (図 2)。自分のノードのメモリにないデータへのアクセスは page fault によって検出され、OS によって他の共有ノードのデータの更新/無効化がおこなわれる。

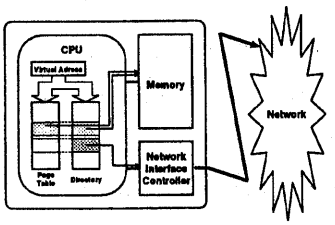


図 2　分散仮想共有メモリ

2.2 特　　徴

この方式の長所は、特別なハードウェアを必要としないため、低コストでアーキテクチャに依存しない移植性の高い共有メモリ機能を実現できるところにある。しかし、以下の事柄により性能の低下を招いている。

- 共有メモリの参照によりページアクセス例外を発生させて、ソフトウェアで通信を起動させるため、共有データ参照時のオーバーヘッドが大きい。このオーバーヘッドはプロセススイッチと通信起動によって発生している。(通信性能の低下)
- 仮想記憶機構により共有メモリアクセスを検出するため、共有データブロックはページサイズ (一般に 4 kbytes) になる。これにより一回の通信に含まれるデータはページサイズとなり、転送に時間がかかる。(通信性能の低下)
- データ共有の単位がページサイズなので、関係のないデータへのアクセスによる共有データの無効化 (False Sharing)(図 3) がおこりやすい。False Sharing は、非共有データへのアクセスによる不必要な通信を引き起こし性能低下させる。(不要な通信の増加) また、適切なデータ一貫性管理プロトコルを用いないと、二つのノードで共有データの内容が異なるといった状態を引き起こしやすい。

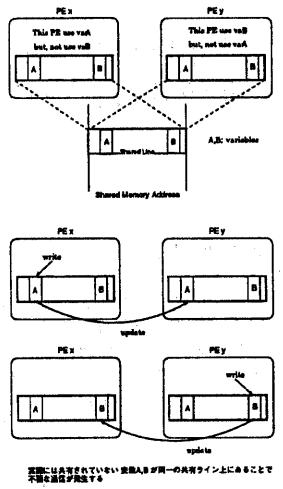


図 3　False Sharing

ノード間通信を Ethernet のような低速 LAN で行う場合は、データ転送のスループットが性能に影響を与える。

しかし、性能は低いが原理は計算機アーキテクチャに依存しないため、様々な実装 (IVY[1], TreadMark[2] 等) が存在する。

2.3 関連研究

上記の仮想共有メモリの性能を改善するための研究は、既に幾つも行われている。

より細粒度なメモリアクセスの検出による性能改善の試みとして、Wisconsin 大学で行われている Wind Tunnel Project[3] における Blizzard[4] がある。Blizzard には、多くの WS のメモリに備えられたエラー訂正用メモリ (ECC) を変更することで、キャッシュラインサイズ程度のメモリアクセス検出を可能とする Blizzard-E と、実行前にシミュレーションを行い共有アクセス命令を検出し、その前後にキャッシュラインサイズ毎の共有状態を示すテーブルを参照する命令を挿入する Blizzard-S がある。

また仮想共有メモリではないが、WS クラスタと専用コントローラによる共有メモリの実現も研究されている。JUMP-1/3[5] では、通信インターフェース上にハードウェアによる一貫性管理が行われる共有メモリを載せ、I/O バス経由でアクセスを行う方式が採用されている。この方式でハードウェアにより共有メモリを実現するため、共有メモリそのものの性能は高いが、I/O バスの能力が共有メモリ性能を下げている。

3. 並列アプリケーションの特徴

予備実験として、既存の並列アプリケーションの共有メモリアクセスについての測定をおこなった。

並列アプリケーションとしては、Stanford 大学で開発された並列ベンチマークである SPLASH2 から FFT, LU, RADIX, OCEAN(LU, OCEAN は不連続メモリ割り付けモデルを使用) と、一般の科学技術計算で使用される行列とベクトルの乗算部分 (MATVEC) をマルチスレッド化したものを用いた。これを分散共有メモリ計算機シミュレータ (表1参照) 上で実行し、共有メモリへのアクセス状況を測定した。各 Processor Element(PE) にはローカルメモリが存在し、実行バイナリとスタックはローカルメモリ上に、それ以外のデータは共有メモリに置かれる。

表1 シミュレータの構成

PE 数	16
共有メモリ構成	UMA
メモリアクセス遅延	0(ローカル、共有共に)
実行命令セット	SPARC V8
実行バイナリー形式	ELF(SunOS5)

共有メモリブロックを 4096bytes と仮定した場合の実験結果は以下の通りである。

図4は同一共有メモリブロックを共有するノード数の分布を表わしたものである。

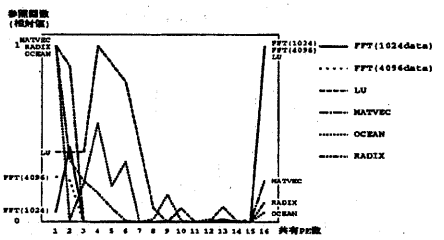


図4 共有ノード数の分布

表2は全ノードで共有される共有メモリブロックの数と、そのブロックへ書き込みを行ったノードの数の平均を表わしている。

図5はそれぞれのアプリケーションでの、PEごとの共有ブロック参照パターンを示している。アプリケーションによっては PE0 に特別の機能を割り振っているため、右側のグラフは PE0 の参照パターンを、左側のグラフはその他の PE の典型的な参照パターンを示している。

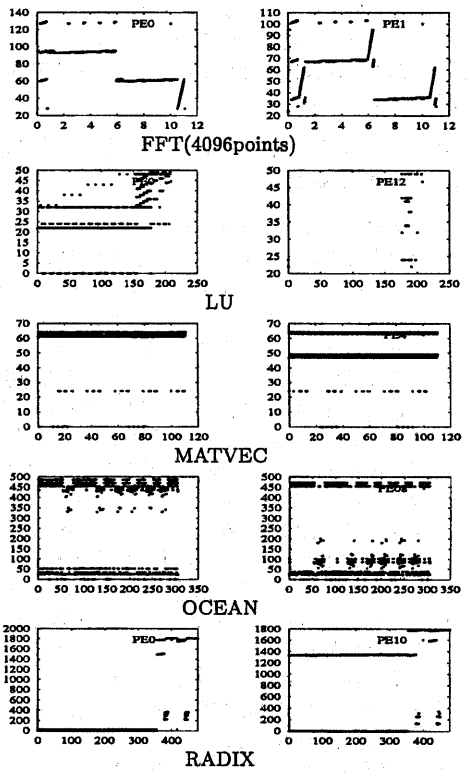
実験結果より

- 2つのノードで共有される共有メモリブロックの数が多いが、これはデータ配置がずれたためだと考えられる。
- 共有メモリブロックをアクセスする PE の数で分類すると、全ての PE からアクセスされるものと、少数の PE によってアクセスされるものに大別でき、多くのアプリケーションにおいて前者よりも後者が多い。
- 多くのアプリケーションでは、全ての PE からアクセスされる共有メモリブロックに対する書き込みは、少数に PE によって行われている。

表2 全ノードで共有されるブロックのアクセス状況

Apps	TB	AA	AAwW	AWCAA
FFT(1024points)	56	37	34	2.243
FFT(4096points)	104	69	66	1.609
LU	22	5	2	3.400
MATVEC	36	4	2	8.000
OCEAN	457	16	5	2.688
RADIX	698	39	34	13.949

TB(Total Block) = アクセスのあった共有メモリブロックの数
AA(All Access) = 全ノードで共有される共有メモリブロックの数
AAwW(AA with Write) =
上記ノードで書き込みのあったものの数
AWCAA(Average Write Counts of AA) =
AA での書き込みノード数の平均



縦軸: 共有ブロック番号　横軸: クロック (1/10000)

図5 共有ブロック参照頻度

- 共有メモリブロックのアクセス頻度は、
  - 他の全ての PE でも同じようなアクセス頻度を示している
  - 他の PE によってあまりアクセスされてない、つまりその PE のアクセスが支配的である

  のどちらかの傾向を示している。
- 少数の PE によって共有される共有メモリブロックでは、短時間に集中してデータがアクセスされる。

ということが判る。

仮想共有メモリにおいて、全ての PE が高頻度にアクセスするようなデータは各 PE にコピーされるため、性能に影響を与えないと思われる。上記の結果からディレクトリ情報は全ての共有ブロックに付加せず、一部の頻繁に使用される部分だけでも特に問題を生じないことが判った。

4. ディレクトリキャッシュを用いた仮想共有メモリ支援

本提案では、仮想共有メモリの問題点のうち通信性能の改善を図るために共有メモリ専用の通信インターフェースを用いることにする。通信インターフェース内には以下の機構を設け、通信の高速化を図る。

- 共有メモリブロック (ページ単位) 毎のホームノードの位置は通信インターフェース内で管理する。
- 頻繁に使用される共有メモリブロックに関しては、フルマップディレクトリを用意して一貫性管理通信に使用する。(ディレクトリキャッシュ機構)
- 通信インターフェース内でディレクトリを管理するために、通信の起動は各ノードにおけるプロセス ID と仮想アドレス、および物理アドレスを通信インターフェースに渡すことで行う。
- データは通信インターフェース上の DMA 機構を用いて転送する。

本提案では、ディレクトリの管理はホームノードで行うが、ディレクトリ情報をキャッシュしている場合についてはこれを用いる。

4.1 通信機構のハードウェア構成

通信インターフェースの内部は以下に示すように、大きく分けて 4 つの部分からなる。

- 仮想アドレス-ネットワークアドレス変換部 (図 6 の中央上部)
  CPU からプロセス ID と仮想アドレスから、ネットワーク内で一意に定まるネットワークアドレスへの変換/逆変換を行う。
- ディレクトリ保持部分 (図 6 の中央部分)
  ネットワークアドレスに対応するページのホームノードを検索する。
- ディレクトリキャッシュ(図 6 の右上部)
  頻繁に利用する共有データブロックのフルマップディレクトリを保持している。
- ネットワークインターフェース (図 6 の下部)
  共有メモリアクセスによる通信と PE 内メモリの読み込み/書き込みを行う。

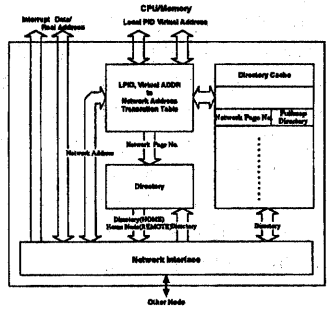


図 6 通信インターフェースの内部構成

各部分の構成は以下の通りである。

仮想アドレス-ネットワークアドレス変換部分

仮想アドレスとプロセス ID を基にネットワーク上で一意に定まる共有メモリアドレスを生成する (図 7)。ネットワーク上でのデータの読み込み/書き込みは、このアドレスを基に行う。ディレクトリ保持部分とディレクトリキャッシュ部分には、ネットワークアドレスのうちネットワークページナンバーの部分だけを供給する。また、他のノードから送られてきたパケットの

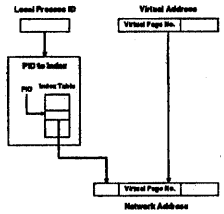


図 7 仮想アドレス-ネットワークアドレス変換部分

ネットワークアドレスからローカルなプロセス ID と仮想アドレスへの変換も行う。逆変換でプロセス ID は、インデックステーブルのデータの検索で算出する。

ディレクトリ保持部分

ネットワークページナンバーを基に、ホームノードの検索を行う。ここではネットワークページナンバーに、各 PE を静的に割当てる。内部構成は図 8 に示されるように、アクセスのあったネットワークページアドレスがどの PE のメモリ上に存在するのかを判別する機構と、自分がホームノードだった場合のディレクトリを保持するテーブル (フルマップディレクトリ) の二つから構成される。この機構は、自己ノードの共有メモリアクセスによる通信と他のノードからのディレクトリ情報要求への返答で使用される。

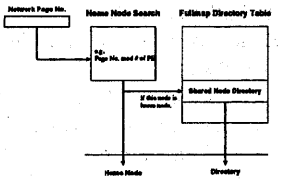


図 8 ディレクトリ保持部分

ディレクトリキャッシュ

頻繁に利用する共有メモリブロックのディレクトリ情報をキャッシュする。

キャッシュエントリーは、ネットワークページナンバー、共有ノードディレクトリからなり、エントリーの割当ては自分のノードにおけるデータ読み込み/書き込み要求と他のノードからのデータ要求により行われる。他のノードからの無効化要求によりエントリーは削除される。また、ディレクトリ情報の更新はネットワークインターフェースから送られるディレクトリ情報によって行われる。

ネットワークインターフェース

ネットワークインターフェースは通信におけるパケットの生成、送受信を処理する部分である。送られてきたパケットによる CPU への割込み、受信確認メッセージの送信、返答データの送信、ディレクトリキャッシュへのデータの受渡しはネットワークインターフェースの処理である。

ネットワークインターフェースは以下の機能を持つ。

- 自己ノードにおける共有メモリアクセス発生時のデータ要求/更新パケットの生成
- 他のノードのデータ読み込み要求によるメモリからデータ送信
- 他のノードのデータ無効化要求による CPU への割込み
- ホームノードだった場合、他のノードのデータアクセスによるディレクトリ情報の更新
- ディレクトリキャッシュ上にデータがあった場合、他のノードのデータアクセスによるディレクトリ情報の更新
- 受信したパケットの受信確認パケットの送信

4.2 ディレクトリキャッシュによる仮想共有メモリ

ここでは、上記の通信インターフェースを利用した仮想共有アドレス機構について説明する。

OS の仮想記憶管理機構に以下のような機能を付加する。

- **共有メモリアクセスの検出**
  ページフォールトにより共有メモリのアクセスを検出する。この時、共有メモリアクセスと本来のページフォールトと区別するためのテーブルを引く。
- **共有メモリアクセスによる通信インターフェースへのデータ要求**
  ページフォールトが共有メモリアクセスだった場合、通信インターフェースにプロセス ID と仮想アドレス、及びデータを書き込む実メモリアドレスを渡して共有データが送られてくるのを待つ。
- **アクセスコントロール**
  共有メモリの仮想アドレスに対応するページテーブルエントリーは、ホームノードで他に共有ノードがない場合のみ Read/Write に、それ以外は常に Read Only に設定する。

また、通信インターフェースからの割込みの処理として、

- 要求のあったプロセス ID と仮想アドレスからページテーブルを参照し、通信インターフェースに該当共有メモリブロックの実アドレスを渡す。(他のノードの読み込み)
- 要求のあったプロセス ID のページテーブルの該当する仮想アドレスの部分を無効化する。(他のノードの書き込み)

を行う。

4.3 一貫性管理プロトコル

一般的に共有メモリではライトバック動作より通信頻度を下げることができるが、ライトスルーでは常にオーナー = ホームとなり通信インターフェース上の共有ノード管理を簡略化できる。更新型に比べ無効化型では、書き込み時の通信に使用されるパケットが 1 つで済み、また、無効化されたノードはディレクトリから削除できるので、その後の書き込みにおける相手を減らす効果もある。そこで、この手法での一貫性管理プロトコルは、ライトスルー/無効化型を基本とする。通信に使用されるパケットは、読み込み、書き込み、書き込み (無効化済)、無効化、ディレクトリ更新のそれぞれに要求、返答を行うため、計 10 種類が用いられる。

ホームノード

本手法では、一貫性管理プロトコルとしてライトスルーを用いている。これにより、常に正しいデータがホームノード上に存在することになる。

共有メモリアクセスによるホームノードでの動作は以下の通り。

- **他のノードによる読み込み**
  読み込みを行ったノードをディレクトリに追加する。この時、全て共有しているノードに対してディレクトリ更新のパケットを送る。読み込みを行ったノードに対してデータとディレクトリ情報を返答する。
- **自己または他のノードによる書き込み**
  書き込んだノード以外をディレクトリから削除し、これらのノードに無効化のパケットを送る。他のノードによる書き込みの場合、受け取ったデータを用いて PE 上のメモリの更新を行う。
- **他のノードによる書き込み (無効化済)**

書き込んだノード以外をディレクトリから削除する。受け取ったデータを用いてPE上のメモリの更新を行う。

ホームノード以外

共有メモリアクセスによるホームノード以外のノードでの動作は以下の通り。

- 自己ノードによる読み込み
  ホームノードへデータを要求する。受け取ったデータをPE上のメモリに書き込み、CPUにデータが有効になったことを報せる。同時にディレクトリ情報をディレクトリキャッシュに格納する。
- 自己ノードによる書き込み
  ディレクトリキャッシュにディレクトリが存在した場合は、ホームノード以外に無効化パケットを、ホームノードには書き込み(無効化済)のパケットを送る。ディレクトリが存在しない場合はホームノードに直接書き込みパケットを送る。
- ホームノードからの無効化
  キャッシュのディレクトリエントリを削除し、CPUに割込みをかけ、該当するページテーブルを無効化する
- ホームノードからのディレクトリ更新
  キャッシュ上にディレクトリエントリが存在した場合は更新する。そうでない場合は無視する。

5. おわりに

本稿では、仮想共有メモリの支援機構として共有ノードディレクトリのキャッシング機能を備えた専用通信インターフェースの提案を行った。PE毎に共有メモリブロックアクセスのパターンは異なるため、ディレクトリ情報として共有メモリブロック毎のホームノードのみを記憶し、頻繁に使用する共有ブロックのディレクトリをキャッシュしても性能を悪化させることは無い。また通信インターフェース上にディレクトリ情報を持つことで、データの変更に関する通信でのCPU使用時間を削減することが可能である。

現在、本提案手法の性能評価のためのシミュレータを作成している。今後は、割込みによるプロセススイッチの時間や通信を行うために必要な時間を含め、ソフトウェアによる仮想共有メモリとの比較をシミュレーションによって行う。また、通信機構に共有メモリブロック毎のディレクトリを付加した場合との比較も行う。

参考文献

1) Kai Li. IVY: A Shared Virtual Memory System for Parallel Computing. In *International Conference on Parallel Processing*, pp. 94 – 101, 1988.
2) Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, and Willy Zwaenepoel. An Integrated Approach to Distributed Shared Memory. In *First International Workshop on Parallel Processing*, December 1994.
3) Mark D. Hill, James R. Larus, and David A. Wood. Parallel Computer Research in the Wisconsin Wind Tunnel Project. In *NSF Conference on Experimental Research in Computer Systems*. NSF, June 1996.
4) Mark D. Hill, James R. Larus, and David A. Wood. Tempest: A Substrate for Portable Parallel Programs. In *Compcon*, March 1995.
5) 安生健一朗, 西宏章, 董小社, 吉川晃, 工藤知宏, 中條拓伯, 天野英晴. 超並列計算機用結合網RDTのルーティング制御評価用システム: JUMP-1/3. 信学技報 (CPSY), pp. 39–46, August 1996.