WS/PC クラスタにおけるハードウェア支援を用いた仮想共有メモリの実現

滝田　裕[†]　田中英彦[†]

並列計算機上で比較的容易に修得できるマルチスレッドによる並列化を行うには、共有メモリを備えた実行環境が必要である。安価な並列計算機である WS/PC クラスタ上で共有メモリを実現するには仮想共有メモリを使用するが、ハードウェアによる共有メモリと比較すると共有メモリへのアクセス性能が非常に低い。そこで I/O バス上に仮想共有メモリの処理の一部である共有ノード管理とリモートデータのキャッシュを行う支援機構を用意して、仮想共有メモリの性能向上させる手法を提案する。シミュレーションによりいくつかのアプリケーションでの評価を行い、キャッシュの増加させることで性能向上することが示された。

Hardware accelerated shared virtual memory on WS/PC cluster

Hiroshi Takita[†] and Hidehiko Tanaka[†]

Multi thread programing is easier than message passing programing on parallel program, but it requires shared memory on parallel computer. Most popular shared memory implementation on WS/PC cluster is Shared Virtual Memory(SVM). SVM is inferior than Hardware Distributed Shared Memory on shard memory access time.

We propose a technique to improve shared memory access by the shared node management and remote data cache mechanisms on I/O bus. Our simulation results show that system performance is improved by increase of remote data cache capacity.

1. はじめに

従来の共有メモリを持たない並列計算機ではメッセージパッシングにより並列処理を記述する必要があった。これは互いのノードのデータの参照を行う時に、通信を直接記述する必要があるためプログラムの開発が難しく、一般に並列計算機が普及しない原因となった。しかし、ノード間のメモリを共有している並列計算機で用いられるマルチスレッドによる並列化は、並列に実行されるスレッド間の共有データアクセスで明示的な通信を記述する必要がないため、より容易な修得、開発が可能になる。

WS/PC クラスタは価格性能比の高い WS/PC を複数用意し LAN で接続した安価な並列計算機である。WS/PC クラスタは基本的に共有メモリを持たない並列計算機であるため、並列実行可能なアプリケーションを開発するのは非常に難しかった。現在、WS/PC クラスタ上でマルチスレッドによる並列化を可能にするために、WS/PC クラスタ上の共有メモリ実現に関する研究が数多く行われている。

従来の WS/PC クラスタ上の共有メモリの多くは、OS の仮想記憶管理機構を利用した仮想共有メモリ (Shared Virtual Memory: SVM) で実現されている。この手法は OS の改変だけで実装可能だがページ単位の共有を行うため、通信が遅い並列計算機では性能が極端に低下する。またデータを共有しているノードの管理や共有データの転送も CPU が行うため、共有メモリのアクセスで使用される CPU の処理時間は多大なものとなっている。

そこで本研究では、WS/PC に備えられている I/O バス上に共有ノード管理とリモートデータのキャッシュを行う支援機構を用意することで、仮想共有メモリの性能改善を図る。支援機構上のキャッシュによりデータ転送で使用される CPU の処理時間の削減とキャッシュライン単位でのデータ更新を実現し、共有ノード管理機構は CPU への割込みを低下させる。

本稿では、仮想共有メモリの概略と問題点を述べ、SVM 支援機構の概念、構成を説明する。またシミュレーションによる SVM 支援機構がある場合とない場合との比較を行い、最後に今後の課題について触れる。

† 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

2. 仮想共有メモリ

2.1 概　　略

IVY[1] で提案された SVM 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、仮想アドレスレベル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 に示すようにメモリ上にないデータへのアクセスを page fault で検出し、ページテーブルの情報を利用してディスク上にスワップされたデータをメモリに読み込む機構である。

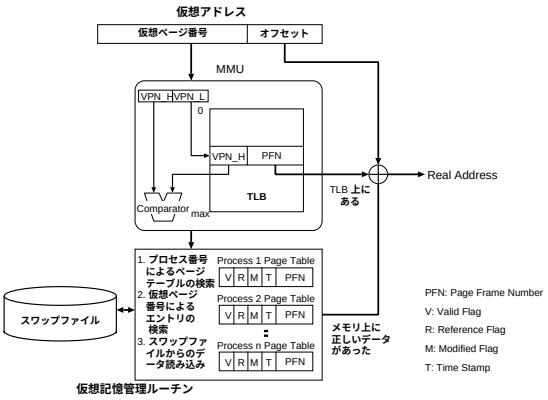


図 1　一般の仮想メモリ機構

SVM では、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仮想記憶同様 page fault によって検出されるため、共有データの転送はページ (約 4KBytes) 単位で行われる。各ノード上にあるデータは、Berkeley プロトコルに似たライトバック無効化型のデータ一致プロトコルによって一貫性が保持される。共有メモリの配置は、ある 1 つのノードに集中しているものと各ノードに分散して配置されるものの 2 種類が実装されている。共有メモリが分散して配置される場合は、共有ブロックのオーナーの情報を 1 つのノードだけが持つものと全てのノードに配布されるものが実装された。

SVM は、各ノードに仮想記憶機構を構築できる並列計算機ならばどのようなものでも実装できるため、WS/PC クラスタのような分散メモリ型の並列計算機での共有メモリ機構の実装によく使用されている。

2.2 問題点と関連研究

仮想共有メモリはソフトウェアのみで実装しているため汎用性が高い反面、以下のような問題点を持っている。

- 共有データアクセスにより引き起こされる通信一回で、ページサイズ (おおよそ 4 KBytes) のデータの転送を行う必要があり、実際に CPU が必要としているデータはキャッシュラインサイズ (64 ∼ 256bytes) でしかないことを考えると多くの無駄なデータが転送されることになる。これはノード間通信が CPU-Memory 間の転送に比べて大幅に遅い WS/PC クラスタでは、性能に大きな影響をあたえる。
- データ共有の単位がページサイズなので、本来ならば共有されてない変数へのアクセスでデータの一貫性管理機構が働いてしまう false sharing 3 がおこりやすい。false sharing は不必要な通信を引き起こし性能低下させる。また、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。

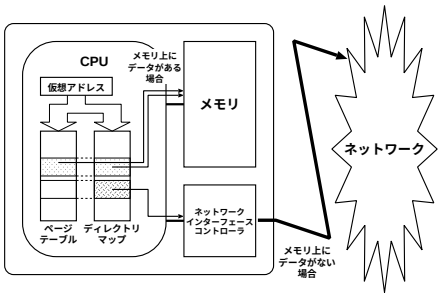


図 2　仮想共有メモリ

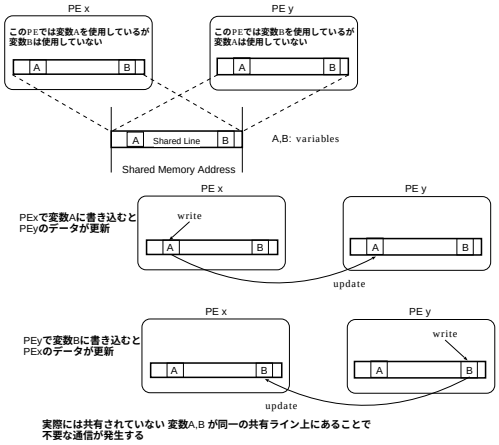


図 3　False Sharing

- データを持つノードの検索とノード間通信に関する処理は全て CPU によって行われるため、専用のハードウェアを用いてこれらの処理を行うものと比較した場合、通信が実際に行われるまでの時間が長くなっている。
- 他のノードの共有メモリ参照により、正しい共有データを持つノードの CPU がデータの転送に使用されてしまい、他の処理を行うことができなくなる。

これらの問題点を改善し SVM の共有メモリアクセ

ス性能を高める研究としては、高速なノード間通信機構によりデータ転送にかかる時間を削減したものや、共有メモリアクセスの前後に高速なタグチェック命令を挿入することでページサイズよりも細かなブロックを単位として一貫性管理を行うものがある。高速ノード間通信を用いた研究では汎用のメモリベース通信インターフェースが使用されることが多く、Myrinet[2] を用いた Shrimp[3] や Memory Channel[4] を使用した CASHMERE[5] 等が有名である。タグチェック命令挿入による研究は、コンパイル時に共有メモリアクセスを検出し適切な共有ブロックサイズを確定させる Shasta[6] や実行前にシミュレータにより共有メモリアクセスを検出しバイナリの改変を行う Blizzard-S[7] がよく知られている。

3. SVM 支援機構

本研究では図 4 のような構成の WS/PC をノードとし、各ノード間が高速なネットワークによって接続されている WS/PC クラスタを対象とする。このような WS/PC クラスタでの仮想共有メモリ実装で、主にデータ転送時に CPU が行う処理の低減を目的としている。

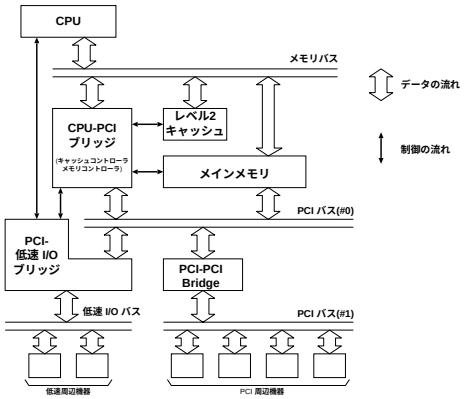


図 4 対象とする WS/PC の構成

マルチスレッドで並列化されたアプリケーションの共有データアクセスをトレースした結果、各ノードにおける共有データアクセスの局所性は高いことが示されている。そこで、他のノードからアクセスのあったデータを I/O バス上の SVM 支援機構にキャッシュすることで、CPU を介さずにデータの転送、更新を行うことが可能になる。データの転送や更新には共有ノードのディレクトリ情報が不可欠なので、これも CPU から SVM 支援機構に移すことにする。これにより、共有メモリに CPU が関わっているクロック数が減少し、またデータ更新が単時間で行われるようになるので共有メモリのアクセス性能が向上する。

SVM 支援機構の内部構成

SVM 支援機構の内部は図 5 に示すように、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**
  CPU からプロセス ID と仮想アドレスを受取り、ネットワーク内で一意に定まるネットワークアドレスに変換するテーブルで構成される。
- **共有ノード ディレクトリ**
  ネットワークアドレスから、そのアドレスが属する共有ブロックのホームとなるノードの番号を参照するためのテーブルと、自分のノードがホームである共有ブロックのコピーがどこにあるのかを示すディレクトリから構成される。
- **データキャッシュ**
  他のノードからの読み出し要求や、他のノードへの読み出し要求で受け取った共有ブロックをキャッシュする。同時にその共有ブロックのフルマップディレクトリも保持している。
- **ネットワークコントローラ**
  共有メモリへのアクセスでパケットを生成し、通信を行なう。また、他のノードによるデータ書き込みによって CPU に割込みをかけ、メモリのページ無効化を行う。

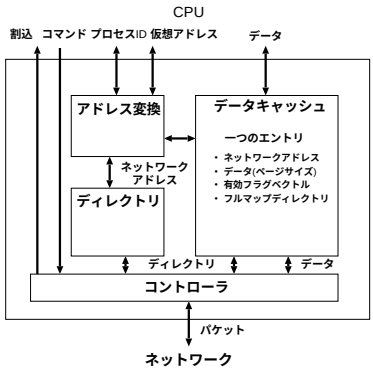


図 5 SVM 支援機構の構成

一貫性管理プロトコル

ここでは各ノードの状態を簡単にするために、各ノードの書き込みによって共有ブロックのホームのメモリにあるデータは更新、それ以外は無効化される write through/無効化型の一貫性管理プロトコルを基本に考える。この場合、各ノードのメインメモリ上にキャッシュされたデータは、他のノードの書き込みによりつねに無効化される。しかし、これではページ単位で共有ブロックを管理しているために頻繁にデータの無効化がおきてしまい、性能が大きく低下する。そこで、支援機構上のデータキャッシュ上にデータがキャッシュされていた場合は更新を行えるように、共有データ書き込みによって送られる無効化メッセージにキャッシュラインサイズの更新されたデータを入れることにする。

一般の共有メモリでの通信では無効化に必要なパケットは 1 つで済むが、更新のためには数パケット送る必要があり、更新のコストを引き上げていた。一般的な SVM ではもっぱら無効化により一貫性を管理していたが、この手法を用いることでデータキャッシュ上にあるデータだけは更新が無効化と同一のコストで可能となる。キャッシュにのっているデータの読み込みに関してはキャッシュライン単位での、その他のデータに対してはページ単位のコヒーレント管理が SVM 支援機構により提供される。

各ノードでの動作

共有ノードの共有ディレクトリ管理は SVM 支援機構によりページ単位で行われ、ホームノードとキャッシュ上のデータがあるノードに関しては全ての共有ノードの情報を、その他のノードはホームノードがどこにあるのかの情報を持っている。この情報を基にネットワークコントローラは共有データアクセスを他のノードへの通信に変換する。共有メモリのノード管理は SVM 支援機構で行うため、共有メモリアクセスがおこった後でも、CPU は他のプロセスの実行を行うことが可能になる。

各ノードの共有メモリアクセス時の動作は以下の通りである。

- **共有メモリの読み込み**
  - **読み込み要求ノード**
    (1) メモリアクセスにより page fault が発生する。発生しなかった場合は、そのノードのメインメモリ上に共有データのコピーが存在している。
    (2) SVM 支援機構にプロセス ID と仮想アドレスを渡し、データが送られてくるのを待つ。
    (3) SVM 支援機構がホームノードから送られてきたデータを直接メモリに書き込む。
    (4) 転送完了を CPU に通知。
  - **ホームノード**
    (1) データキャッシュ上にデータがある場合はそこから転送する。
    (2) 共有ノードのディレクトリ情報を更新する。
    (3) データキャッシュにない場合、CPU から該当ページの物理アドレスを受け取る。
    (4) SVM 支援機は、メインメモリ上のデータを要求のあったノードに転送すると同時にデータキャッシュにキャッシュする。
- **共有メモリへの書き込み**
  - **書き込み要求ノード**
    (1) 共有メモリへの書き込みは必ず page fault が発生する。
    (2) SVM 支援機構にプロセス ID, 仮想アドレス、書き込むデータを渡して SVM 支援機構の応答を待つ。
  - **書き込み要求を受け取ったノード**
    (1) ホームノードでない場合はデータキャッシュの該当エントリを更新し、CPU に対して該当ブロックの無効化を要求する。
    (2) ホームノードの場合、データキャッシュの該当エントリの更新または新規エントリの作成を行い、CPU に該当ブロックの一部のデータが変更されたことを通知する。

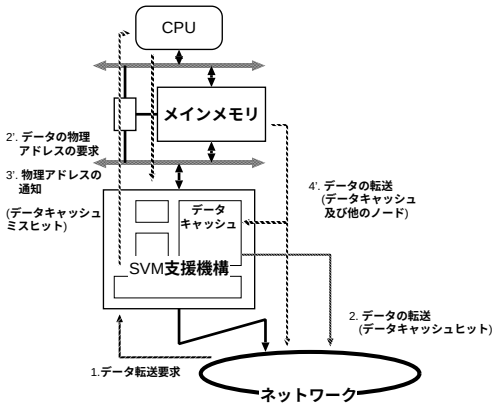


図 6 ホームノードでの動作

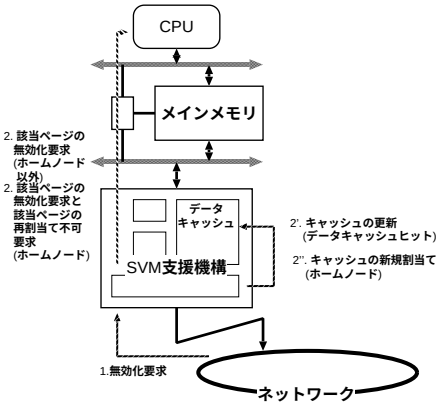
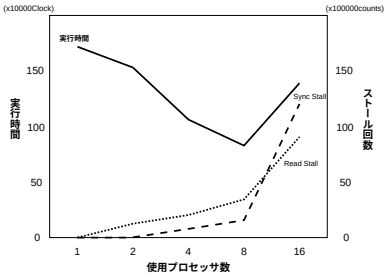


図 7 書き込み要求を受け取ったノードの動作

4. SVM 支援機構の評価

本研究の有効性を示すために、シミュレーションによる評価を行った。シミュレータの仕様は表 1 の通りである。



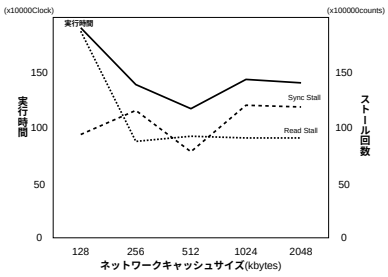


図 8 実行 CPU 数と実行時間 (FFT)

このシミュレータは Solaris 2.x 上で動作し、これ自身が POSIX スレッドを利用したマルチスレッドプログラムになっている。

表 1 シミュレータの構成

ノード数	1〜16
CPU	SPARC V8 相当 ELF 形式マルチスレッド バイナリーの実行が可能
キャッシュライン	64bytes
ページサイズ	4kbytes
メモリ	ローカル 2Mbytes 共有 32Mbytes
I/O 速度	CPU 速度の 1/8
データキャッシュ	128k〜2M bytes
接続形態	スター
ルータティング	ハードウェアによる ダイレクトルーティング
パケット	固定長

このシミュレータ上での測定には、共有メモリ型並列計算機向け並列ベンチマークである SPLASH2 の FFT, LU, RADIX, OCEAN および行列とベクトルの積 (MATVEC) を使用し、SVM 支援機構上のキャッシュの容量と CPU 数を変化させて実行終了までにかかったクロック数を測った。

4.1 実行 CPU 数と実行時間

データキャッシュを 1Mbyte として、CPU 数を変化させて実行クロックの測定を行った。

LU と FFT の CPU 数 16 の場合を除くと概ね CPU 数を増やすことにより、実行時間が短縮している。(図 8, 10, 12, 14, 16) FFT の CPU 数 16 の場合で速度が低下するのは、多数のノードが一つのノードに対してデータ転送要求を行うことがおこり、共有ノードの読み込みでのストールが増大するからである。LU の場合は、行列を正方形のブロックに分割して各ノードに配置しているため共有データアクセスが頻繁に起ってしまい、計算とデータ転送が逐次化されてしまったために速度の向上が見られない。

4.2 データキャッシュ容量と実行時間

実行 CPU 数を 16 として、データキャッシュ容量を 128, 256, 512, 1024, 2048 kbytes と変化させて実行クロックの測定を行った。

OCEAN, RADIX はキャッシュの容量を増加させることにより実行クロックが減少する。FFT はキャッシュ容量が 128 ～ 512 kbytes の場合は減少しているが、1024, 2048 kbytes では逆に増加してる。LU, MATVEC ではまったく変化がみられないが、これは共有データが少なかったために 128 kbytes のキャッシュで全ての共有データをキャッシュできたためである。

5. 結　　論

簡単なプログラムではあるが SPLASH2 の kernels を実行させたところ、いくつかのプログラムでキャッシュ容量の増加により実行クロックが減少した。

今後の課題として、ネットワークキャッシュ上のデータを write back 動作に基づいて更新する一貫性管理プロトコルを用いて、共有メモリ書き込みによる通信の削減が挙げられる。ネットワークキャッシュの write back 動作は、 JUMP-1 の MBP[8] 等で使用されている共有ブロック毎の owner 属性の導入を予定している。

参 考 文 献

1) Li, K.: IVY: A Shared Virtual Memory System for Parallel Computing, *International Conference on Parallel Processing*, pp. 94–101 (1988).
2) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W.-K.: Myrinet – A Gigabit-per-Second Local-Area Network, *IEEE MICRO* (1995).
3) Blumrich, M. A., Li, K., Alpert, R., Dubnicki, C. and Felten, E. W.: Virtual Memory Mapped Network Interface for Shrimp Multicomputer, *The International Symposium on Computer Architecture*, IEEE (1994).
4) Fillo, M. and Gillett, R. B.: Architecture and Implementation of MEMORY CHANNEL2, *http://www.digital.com/info/DTJP03/\ DTJP03HM.HTM* (1997).
5) Kontothanass, L. I. and Scott, M. L.: Using Memory-Mapped Network Interfaces to Improve the Performance of Distributed Shared Memory, *2nd HPCA* (1996).
6) Scales, D.J., Gharachorloo, K. and Thekkath, C.A.: Shasta: A Low Overhead, Software-Only Approach for Supporting Fine-Grain Shared Memory, *the seventh International Conference on Architectural Support for Programing Languages and Operation Systems (ASPLOS-VII)*, Vol. 7, pp. 174–185 (1996).
7) Hill, M. D., Larus, J. R. and Wood, D. A.: Parallel Computer Research in the Wisconsin Wind Tunnel Project, *NSF Conference on Experimental Research in Computer Systems*, NSF (1996).
8) 佐藤充, 三吉貴史, 松本尚, 平木敬, 田中英彦: シミュレーションを用いた擬似フルマップ方式の定量的評価, 情報処理学会計算機アーキテクチャ研究会報告, pp. 201–208 (1994).