計算機アーキテクチャ　104－9
システムソフトウェアと
オペレーティング・システム　62－9
（1994．1．28）

ネットワーク環境における分散共有メモリの実現と評価

藏前 健治　　中條 拓伯　　前川 禎男

神戸大学工学部情報知能工学科

概要

本研究では，OS のカーネルに手を加えずユーザレベルによるソフトウェアのみによって，ネットワーク環境でメッセージ通信による仮想的な分散共有メモリ (DSM, Distributed Shared-Memory) の構築を試みた．共有メモリへのアクセスを高速化するために，ソフトウェア制御のコヒーレントキャッシュを実装する．本稿では，本システムの構成及び無矛盾化プロトコルについて述べ，次に，基本性能評価としてキャッシュへのミス／ヒットなどのアクセスタイプによる遅延時間を測定した結果を示す．さらに，実際の並列プログラムによりシステムを評価するために，行列の乗算と，SPLASH ベンチマークプログラムを実行した場合の性能評価も示す．

The implementation and analysis of Software Distributed Shared-Memory for Workstation Clusters

Kenji Kuramae, Hironori Nakajo and Sadao Maekawa

Department of Computer and Systems Engineering
Kobe University
Nada, Kobe 657, Japan

Abstract

In this paper, we describe the implementation and evaluations of Distributed Shared-Memory (DSM) for workstation clusters by using only standard UNIX user level processes. In order to hide a large amount of the access latency under environment local area network, our DSM system provides software controlled coherent caches. Since the cache-coherent is maintained by distributed-directory-based consistency protocol, the system can distribute load of memory management. We have also measured access latencies in executing some application programs, such as matrix multiplication, SPLASH benchmark sets (mp3d, water) in our system. For these results, we confirm software DSM can be an expectable process communication mechanism for parallel processing on workstation clusters.

1 研究目的

ネットワーク環境は，個々の独立性の高い計算機ノードを多数接続してシステムを構成するため，既存の資源を有効に活用しながら拡張が可能であるという利点を持つ．さらに，複数のノード間で並列に処理を行なうことにより，台数に見合った処理能力の向上が期待されている．

並列処理を行なううえで通信方式の選択は重要な要素である．一般に，並列処理における通信方法としては，メッセージ通信と共有メモリによる方法がある．後者は前者に比べ，プログラマにとってデータ分割やメッセージ通信を明確に定める煩わしさから解放されるなどの利点がある．しかしネットワーク環境において物理的な共有メモリは存在しない．

本研究では，OS のカーネルに手を加えずユーザレベルによるソフトウェアのみによって，ネットワーク上でメッセージ通信による仮想的な分散共有メモリ (DSM, Distributed Shared-Memory) を構築した．共有メモリを分散して配置することで，共有データへのアクセス及び管理負荷を分散させる．また，アクセスを高速化するため，データの一貫性制御をソフトウェアにより行なうコヒーレントキャッシュシステムを設ける．

本稿では，将来の高速ネットワーク環境における並列処理システムのノードに渡るプロセス間通信方法としてのソフトウェア DSM を構築して評価を行ない，システムの有効性・可能性を示す．また，現状での問題点，及び，さらに処理能力向上を図るための課題・方向性を検討する．

2 システム構成

2.1 ソフトウェア DSM の実現

ソフトウェア DSM の概念図を図 1 に示す [1]．共有メモリに対するアクセスを分散させるため，システム全体の共有メモリを，あるサーバノードが一括して管理するのではなく，各ノードに共有メモリを分散して配置する．これによって，共有データへのアクセス及び管理負荷を分散させる．

また，計算機の CPU 能力に対してネットワークを介したアクセス遅延は著しく大きい．アクセスプロセス（システムを利用するユーザプロセス）はアクセスを減らすため，共有メモリに直接アクセスせず，自ノード内のキャッシュに取り込まれた共有メモリのコピーに対してアクセスを行なう．コピーは，共有メモリを分割したページ単位で行なわれる．

共有メモリへのリードアクセスに関しては，同一ページの複数コピーによる同時アクセスを認める．このため，複数のノード間での同一データへのライトアクセス時に対してデータの統一性を維持するために，一貫性制御を 3節で述べる無矛盾化プロトコルを用いて行なう．

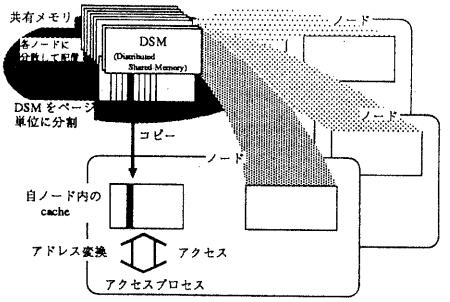

図 1: ソフトウェア DSM の概念図

2.2 ページ管理技法

ネットワーク環境では，あるノードのメモリアクセスを他のノードが監視することは困難であり，ノードからの全アクセスをブロードキャストを行なうにはコストが大き過ぎるため，スヌープ法は現実的でない．そこでページ管理の方法として，どのキャッシュがどのデータを持ってるかという情報をテーブルにより共有メモリの管理側で保持するディレクトリ法を用いる．

本システムでは特に，共有メモリが分散配置されたノード内において管理される分散フルマップディレクトリによってページ管理を行なう．各ノードは，自ノード内の DSM のみを管理すれば良いので，ページ管理の制御負荷を分散することができる．アクセスプロセスから共有空間へのアクセスには，キャッシュの状態も同時に保持するアドレス変換テーブルを用いる．ディレクトリの管理内容，キャッシュの状態の詳細は 3節で述べる．

本システムのディレクトリとアドレス変換機構を図 2 に示す．

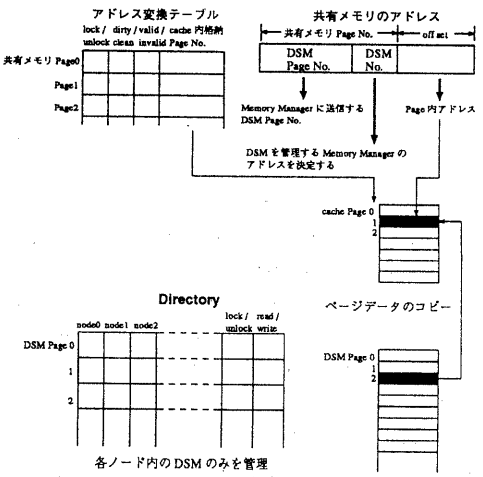

図 2: 分散ディレクトリ法とアドレス変換

2.3 システムを構成するプロセス

ノード内のシステム構成を図3に示す．各ノード毎に，ディレクトリの変更やページデータの無効化を行ないノード内のDSMを管理する **Memory Manager** と，ページ転送やアドレス変換テーブルの管理を行なう **Control Process** と呼ばれる2つのプロセスが存在する．両プロセスによる一貫性制御の詳細は3節で述べる．

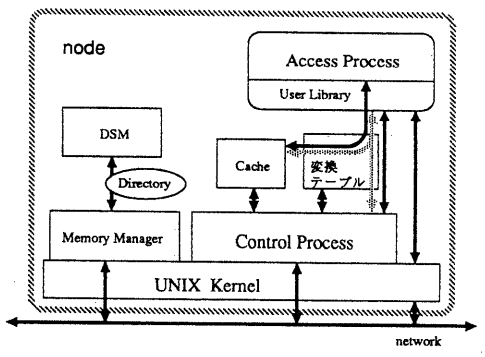

図 3: ノード内のシステム構成

3 DSM における無矛盾化プロトコル

仮想空間を共有する場合，同一ページのコピーが，複数のノード上のキャッシュ内に存在する可能性があり，キャッシュへの書き込み動作によって，内容の統一性が損なわれる可能性がある．

本システムでは，書き込み無効化／キャッシュ間転送型のライトバック方式の無矛盾化プロトコル [3] を用いて一貫制御を行なう．

3.1 DSM のページ状態

システムはプロセスからのアクセスを制御するため，共有メモリのページを，access (read/write), nodeset, lock (lock/unlock) の各フィールドを持つディレクトリによって管理する．

access フィールドが read の場合，共有メモリ内のページの内容が最新であり，write の場合はあるノードのキャッシュ上に書き込みが行なわれたが，まだ書き戻しが行なわれておらず，そのページの内容は最新ではない．nodeset フィールドでは，どのノードのキャッシュ上に最新のページのコピーが存在するかを管理する．lock フィールドは，共有メモリの同一ページに同時にアクセスがあった場合に，各ノードからのアクセスの直列化を行なうのに用いる．

3.2 キャッシュ上のページ属性

キャッシュ上のページ状態は valid/invalid（有効／無効），clean/dirty, lock/unlock で区分する．

ページ状態が clean の場合，共有メモリとページ内容が一致し，他のノードに同一ページのコピーが存在する可能性がある．よって，ライトアクセスを行なう際，他ノードのキャッシュデータの無効化を要求する必要がある．dirty の場合，すでにページに対して書き込み無効化が行なわれ，そのキャッシュ内のコピーのみが最新のデータなので，そのままアクセスが可能である．他ノードが同一ページに対してリードアクセスを行なうためページ転送を要求した時，同時に共有メモリへページを書き戻す．この時，状態は再び clean に変わる．また，アクセスの途中で無効化されることを防止する時，ページ状態を lock にする．

3つの状態を組み合わせて，キャッシュのページ属性は *I* 属性 (invalid)，*CU* 属性 (valid/clean/unlock)，*CL* 属性 (valid/clean/lock)，*DU* 属性 (valid/dirty/unlock)，*DL* 属性 (valid/dirty/lock) の5つに分類される．

3.3 アクセスの分類とシステムの動作

システム上のアクセスプロセスによるアクセスを，アクセス内容（リード／ライト），キャッシュのページ状態（ヒット／ミス，clean / dirty）で区分し分類すると，アクセスプロセスからのアクセスはリードヒット，ダーティライト（dirty page へのライト），クリーンライト（clean page へのライト），リードミス，ライトミスの5つに分類される．

以下にそれぞれのアクセス時に伴うシステムの動作について簡単に述べる．

リードヒット，ダーティライト　そのままアクセスが可能である．キャッシュの属性を変更する必要もない．

クリーンライト　他に同一コピーをキャッシュ内に持つノードが存在する可能性があるので，ページを管理する **Memory Manager** に無効化要求を行ない，他のコピーが全て無効化されてからアクセスを行なう．**Memory Manager** は同一ページを保持する他の全てのノードに無効化を発行し，要求先のノードに対して無効化応答を行なう．その後，access フィールドを write に変更し，nodeset フィールドを要求したノードの ID のみ登録する．要求先のノードは無効化応答が送信されるまでアクセスを行なえない．

リードミス　**Memory Manager** にリード要求を行なう．ページの access フィールドが read の場合，共有メモリに最新のページが存在するので，要求を行なったノードに対してページデータを送信した後，nodeset フィールドにリード要求を行なったノードの ID を追加する．access フィールドが write の場合，他のノードのキャッシュ内に，唯一最新のページが存在する．そこで **Memory Manager** は nodeset フィールドより最新のページを保持するノードを知り，リード要求を行なったノードへのページの転送及び書き戻しを要求する．転送要求を受け取ったノードは，ページデータを転送した後，キャッシュ内のページ属性を *DU* から *CU* に変更する．**Memory Manager** は，書き戻しが行われるまで lock フィールドを lock のままにする．書き戻しが行なわれた後，access フィー

ルドを read に変更し，nodeset フィールドにリード要求を行なったノードの ID を追加する．

ライトミス　**Memory Manager** にライト要求を行なう．ページの access フィールドが read の場合，同一ページを保持する他の全てのノードに無効化を発行し，ライト要求を行なったノードに対してページデータを送信する．その後，access フィールドを write に変更し，nodeset フィールドには，ライト要求を行なったノードの ID のみ登録する．access フィールドが write の場合，**Memory Manager** は最新のページを保持するノードにページデータの転送を要求する．その後，nodeset フィールドをライト要求を行なったノードの ID のみを登録し，受信応答を受けとるまで lock フィールドを lock のままにする．転送要求を受け取ったノードは，ライト要求を行なったノードに対しページデータを転送し，自ノードのキャッシュのページ属性を *DU* から *I* に変更する．

ページデータを受け取ったノードは，そのデータの送信元を調べる．もし，送信元がライト要求を行なった **Memory Manager** でない場合，**Memory Manager** に対して受信応答を行なう．

3.4　マルチキャストによる無効化

ネットワーク環境において他のノードのアクセスを知ることは困難であり，ライトアクセスを行なう際，キャッシュ内の情報だけで他ノードのページの無効化を発行するノードを知ることはできない．また，本稿で述べた無効化型の無矛盾化プロトコルでは，無効化が必要なライトアクセスの多くは 1 ～ 3 台のノードの無効化を行なうに過ぎず，ブロードキャストによって全てのノードへ無効化を発行する方法は，コストが大き過ぎる．

本システムでは，ディレクトリにより無効化を発行すべきノードを知ることができる **Memory Manager** へ無効化要求を行なうことにより無効化を行なう．無効化を行なうノードが複数存在する場合，**Memory Manager** は各ノード内のページを順に無効化するメッセージを一つ作成する．メッセージはノードを巡回しながら無効化を行ない，最後に要求元のノードへ無効化応答のメッセージとして到着する．この方法はシーケンシャルに無効化を行なうため，無効化を行なうノード数が多くなると無効化に要する時間が増大する半面，ネットワークを行き交うメッセージが少なくて済む利点がある．

3.5　ページ置換

データのコピーをキャッシュに書き込む際，どのキャッシュページに書き込みを行なうかを決定する必要がある．ページ番号を決定するには，共有メモリのページ番号から情報を得るアドレス変換テーブルとは別に，キャッシュページの番号から情報を得る機構が必要である．本システムではキャッシュページの管理リストを，UNIX system call による共有メモリを用いた双方向リストで実現する．ページを書き込む際に，リストの先頭を参照し，データを書き込むキャッシュのページ番号を決定する．

もし，キャッシュページに，すでに共有メモリのデータが書き込まれている場合，現在書き込まれているデータのページの書き戻しを行なう必要がある．ページアウトするデータが共有メモリの内容と一致している場合，ページを管理する **Memory Manager** へノード内のキャッシュからデータが削除させたことを通知するだけでよい．ライトアクセスが行なわれ，共有メモリに書き戻す必要がある場合，**Memory Manager** へページデータを転送する．アクセスプロセスがアクセスを行なう際，アクセスを行なったキャッシュページのリストを後方へ移動させることにより，LRU アルゴリズムを実現できる．

3.6　同期機構について

本システムでは共有メモリを分割したページ単位でロック行なうことで同期をとることが可能である．ロックは，キャッシュ側のアドレス変換テーブルへロックを行なう．同じページに対する他のノードからのアクセス要求は，最初の要求については，**Control Process** 側でアクセス可能になるまでキューに退避される．以降の要求は，**Memory Manager** 側でキューに入れられる．

4　基本性能評価

本システムを，Sun SPARCstationELC (21.0MIPS, 33MHz, 8MRAM) を Ethernet (10Mbps) で接続された動作環境で構築し，評価を行なった．また，プロセス間通信には UDP ソケットを用いた．

3.3節で述べたアクセスの分類に基づき，ページデータへの各アクセス時間を計測する．ページデータを管理する **Memory Manager** の存在する位置がアクセスプロセスと同じノード（ローカル）か，ネットワークを介した他のノード（リモート）かによってアクセス時間が異なるため，別々に区分して計測した．また，ライトアクセスの際に他のノードのキャッシュデータの無効化が必要な場合，無効化されるノード数によって遅延の変化が異なる．アクセスを行なう際に **Control Process** や **Memory Manager** に対して要求が必要な場合（リードミス，ライトミス，クリーンライト）で，かつ他のノードが共有メモリを利用していない場合の結果を図 4 に示す．

図 4 より，リードミス，ライトミスの場合，アクセス時間はページサイズにほぼ比例し，リモートの DSM へのアクセスの方がローカルの DSM へのアクセスより遅いことがわかる．また，クリーンライトは，キャッシュ内にデータがあるので，遅延の変化がページサイズに依存しない．

なお，リードヒット，ダーティライトの時間は 1 アクセス 40～50μsec であった．本システムのキャッシュはアクセスプロセスと **Control Process** で共有されるため，UNIX system call の共有メモリの形式で確保されているが，キャッシュにアクセスするにはセマフォを用いて排他制御を行なわなければいけない．システムコールによるオーバヘッドがキャッシュにヒットした場合のアクセス時間の高速化を妨げていると思われる．

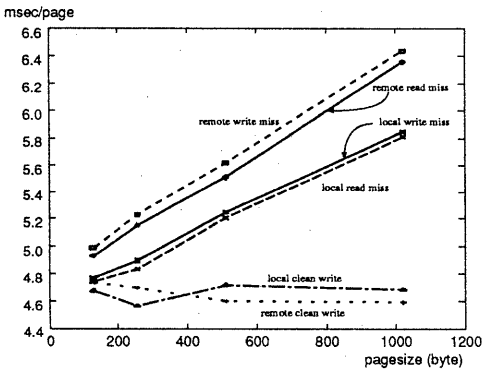

図 4: DSM の存在位置とアクセスの種類によるアクセス時間

アクセス時間の遅延は，データサイズ，アクセスの種類の他にネットワークトラフィックによっても影響される．従って，同時に共有メモリへアクセスするノード数の変化によってもアクセス時間は変化する．各ノード上で，同時にリードアクセスを行ない，1回当りの平均アクセス時間を計測した結果を表 1 に示す．アクセスするノード数が 1 台の場合に比べ，ノード数が 5, 8 台の場合は，それぞれ，1.5，2 倍以上のアクセス時間を要する．台数効果を評価する際，ノード数が増えシステム全体の通信量が増加するに従い，ミスペナルティが大きくなることを考慮しなくてはいけない．

表 1: ノード数によるアクセス時間の変動

node	128	256	512	1024
1	5.92200	6.74600	7.31400	8.90405
2	7.22600	7.84500	8.33000	9.68200
3	8.18333	8.99600	9.30519	11.62867
4	9.63650	9.29700	10.20600	13.08000
5	10.61120	10.53280	12.26160	14.29120
8	13.93325	14.53150	14.48625	19.33175

(単位 $msec$)

ライトアクセスの際に他のノードのキャッシュデータの無効化が必要な場合，無効化されるデータを保持するノード数によって遅延の変化が異なる．無効化されるデータを保持するノード数に対するライトアクセスの遅延の変化を図 5に示す．

本システムの無効化の方法では無効化される各ノード間をメッセージが巡回する．よって，無効化されるノード数が増えるに従い，遅延は大きくなる．しかし，無効化型の無矛盾化プロトコルでは，無効化されるノード数はほとんどが 1 ～ 3 台なので，大幅な遅延が生じる割合は少ない．また，クリーンライトの場合，ネットワークを経由するメッセージ長は，ページサイズに関係なく無効化されるノード数にのみ依存するが，ライトミスの場合，メッセージ中にページデータを含むため，データサイズによってもメッセージ長が異なる．

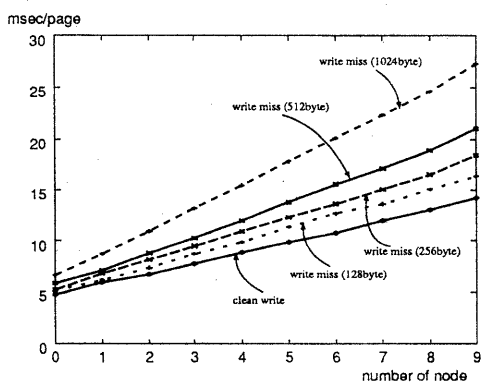

図 5: 無効化されるノード数によるアクセス時間の変動

5 並列プログラムによる評価

実際の並列プログラムを実行し，台数効果の計測を行なった．各ノードのキャッシュのサイズは 256Kbyte に固定し，アプリケーションを実行するノード数の影響のみを単純に比較するため，DSM を保持するノードは特に明記しない限り並列プログラムを実行していない特定の 1 台のノードに固定し評価を行なった．

5.1 行列演算における台数効果

100 × 100 行列 A, B, C を共有メモリ上に確保し，$AB = C$ の乗算を行なった結果を図 6 に示す．台数にほぼ比例した良好な結果が得られている．これは行列 A, B へのアクセスはリードアクセスのみであるうえ，行列の乗算が粒度の大きい演算だからである．また，台数が増えるに従い，台数効果が減少しているのは，ページサイズが大きいため，演算結果を格納する行列 C へのライトアクセス時に，フォールスシェアリングによる競合が起こるためと思われる．

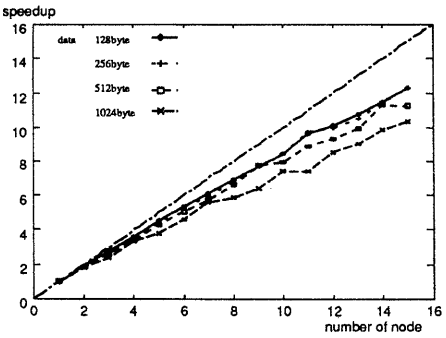

図 6: 100 × 100 の行列の乗算

5.2 SPLASH ベンチマークプログラムによる評価

共有メモリ型並列計算機のベンチマークプログラムセットである SPLASH (Stanford Parallel Applications for Shared-Memory) [4] の中から mp3d (3000mol, 5-step), water (343mol, 2-step) を実行し，評価を行なった．mp3d は，モンテカルロ法を用いて大気中の粒子の状態を摸式化する流体力学シミュレーションである．water は立方体内の液体状態の水分子の動きのシミュレーションをニュートン方程式によって求めるアプリケーションである．図 7, 8 に実行結果を示す．

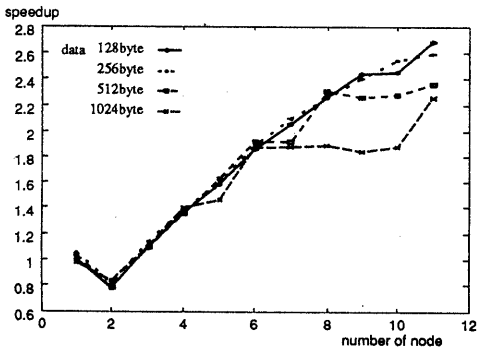

図 7: mp3d (3000mol 5-step)

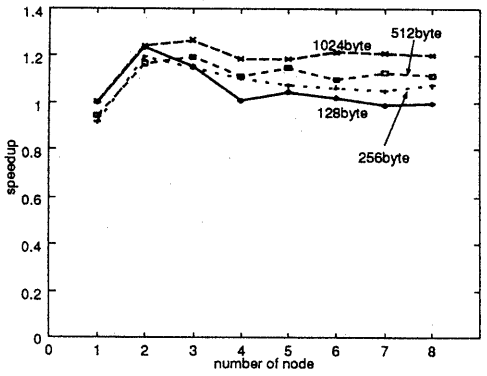

図 8: water (343mol 2-step)

図 7, 8 の実行時間には，実際にアクセスプロセスが CPU を占有し処理を行なっている時間（含バリア同期におけるスピンロックによる待ち時間），**Control Process** がメッセージの処理を行なっている時間，通信時間（含 **Memory Manager** の処理時間）等が含まれる．それぞれの内訳を図 9, 10 に示す．

図 9 では，アクセスプロセスの処理時間自体は，台数に応じて減少していることがわかる．これに伴い，各ノードに到着する通信量も減少しているので，**Control Process** の処理時間も減少していると思われる．しかし，各ノード毎の通信時間は，システム全体の通信量の増大による Ethernet のバンド幅がボトルネックとなり，台数が増えると通信時間はあまり変化しなくなると考えられる．また，バリア同期時における通信待時間の増大も通信時間増加の原因として考慮する必要がある．

図 10 では台数が増えるに従い，通信時間も増加している．通信時間の増加原因としては，water プログラム中においてロック変数による排他的な同期が必要であるホットスポットへのアクセスの待ち時間の増加や，ノード間で同一ページに対する同時アクセスが mp3d に比べ多いことが原因と思われる．

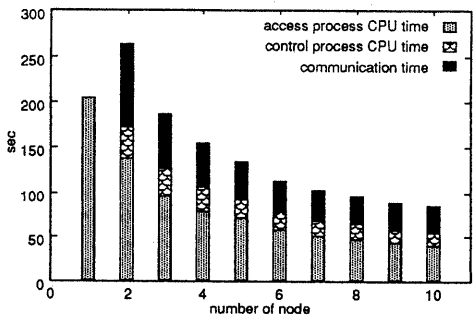

図 9: 実行時間の内訳 (mp3d, page 256byte)

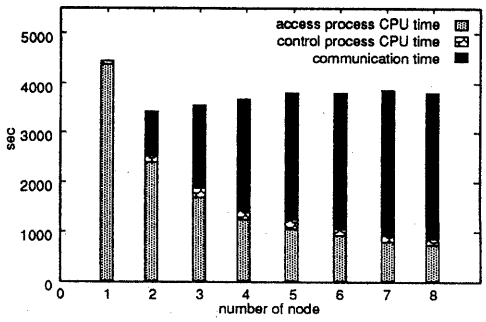

図 10: 実行時間の内訳 (water, page 256byte)

共有メモリに対してどのようなアクセスが行なわれたか，3.3節のアクセスの分類に従い，アクセス回数の統計をとった．各種アクセスの割合を表 2, 3 に示す．

表 2, 3 より mp3d, water におけるヒット率は非常に高い．しかし，実際にはヒット率が高いにも関わらず台数効果があまり得られないのは，ミスした場合のアクセス時間の遅延がヒットした場合のアクセス時間に比べ非常に大きいので，僅かなミスアクセスが実行時間に大きな割合を占める．さらなる台数効果を期待するには，ネットワークの高速化が望まれる．

表 2: 共有メモリに対するアクセスの種類の分類 (mp3d, page 256byte)

ノード数	リードヒット	ダーティライト	クリーンライト	リードミス	ライトミス
1	57.56	42.44	0.00	0.00	0.00
2	55.62	40.50	1.94	1.94	0.00
3	54.89	39.74	2.68	2.67	0.02
4	54.53	39.22	3.08	3.08	0.09
5	54.31	39.08	3.27	3.28	0.05
6	54.16	38.89	3.43	3.45	0.07
7	54.05	38.76	3.54	3.56	0.08
8	53.99	38.65	3.61	3.64	0.11
9	53.94	38.64	3.64	3.68	0.10
10	53.87	38.53	3.73	3.76	0.11

(単位 %)

表 3: 共有メモリに対するアクセスの種類の分類 (water, page 256byte)

ノード数	リードヒット	ダーティライト	クリーンライト	リードミス	ライトミス
1	84.38	15.61	0.00	0.00	0.00
2	84.28	15.52	0.10	0.10	0.00
3	83.90	15.14	0.48	0.49	0.00
4	83.52	14.80	0.83	0.85	0.00
5	83.34	14.71	0.90	1.04	0.00
6	83.16	14.62	0.99	1.22	0.00
7	83.09	14.58	1.04	1.29	0.00
8	83.02	14.58	1.05	1.35	0.00

実行時間に占めるバリア同期の割合を複数回計測した (スピンロックによる待ち時間を含む). 結果を表 4, 5 に示す. アプリケーションを実行する全ノードのバリア同期に要する時間を計測したので, 計測結果の最大・最小値も併せて示す. 表 4 より mp3d のプログラム実行中において, バリア同期に要する時間が大きいことがわかる. 現時点では, バリア同期には, 共有メモリのページに割り当てられたロック変数を 2 個用いることで実現しており, システムの管理が簡素である半面, 1 回のバリア同期でページデータがネットワーク間でやりとりされる回数が多く, ネットワーク環境ではコストが大きい. また, 同じアプリケーションを実行するノード中で同期ポイントに達するまでのノード間のバラツキが大きいのは, 通信時間が CPU の処理時間に比べて大きいので, ノード間におけるキャッシュのヒット率の僅かな差が大きく影響すると思われる. 表 5 より water は mp3d に比べバリア同期に要する時間が少ない. water では同期機構には主にロックを用いており, バリアを使用する回数が少ないからである. しかし, ホットスポットへの各ノードからのアクセスが直列化されるため, バリア同期のポイントに達するまでの時間がノード間でバラツキが大きいようである.

表 4: 実行時間に占めるバリア同期の割合 (mp3d, page 256byte)

node	max (sec)	min (sec)	average (sec)	rate (%)
1	–	–	0.016	0.08
2	14.362	6.922	9.859	3.77
3	10.857	6.883	8.764	4.67
4	17.422	4.471	9.951	6.43
5	25.813	5.183	14.920	11.05
6	16.348	5.617	10.071	8.90
7	19.782	5.106	11.773	11.56
8	19.026	6.653	12.363	13.05
9	22.399	8.588	17.117	19.24
10	27.257	11.307	16.384	19.49

表 5: 実行時間に占めるバリア同期の割合 (water, page 256byte)

node	max (sec)	min (sec)	average (sec)	rate (%)
1	—	—	0.071	0.00
2	330.225	3.819	167.022	4.89
3	221.323	6.049	124.935	3.52
4	96.609	19.870	63.468	1.72
5	175.813	35.017	85.309	2.25
6	176.863	16.666	78.761	2.06
7	163.615	10.701	71.924	1.86
8	209.530	3.619	126.678	3.34

Memory Manager に, あるノードから書き込み無効化要求が届いた時, 他のノードの同一ページコピーの無効化を行なってから書き込みアクセスを行なう必要がある. ライトヒット時に他のノードの無効化を行なった回数を表 6, 7 に示す.

表 6: クリーンライト時における他ノードの無効化回数 (mp3d, ノード数 10)

node	128	256	512	1024
0	0	0	0	0
1	19660	17325	15541	13383
2	351	422	441	629
3	45	46	39	40
4	29	13	9	5
5	7	11	4	3
6	1	10	6	1
7	1	5	8	3
8	2	1	1	4
9	45	45	49	48

(単位 回)

mp3d はノード間でアクセス競合の少ないアルゴリズムのため, 全体の約 95% が 1 台のノードの無効化である. 実際には, 特定のノードで共有メモリの初期化を行なっているため, 無効化されるデータのほとんどは, 初期化を行なったノード内のページデータであると思われる. また, 無効化台数の多いページデータは, バリア変数として用いられているページデータで

ある．waterはページサイズに対して共有メモリに使用するデータのサイズが小さいので，mp3dに比べ多くのノードを無効化してから書き込みを行なうアクセスの割合が高いと思われる．ヒット率が高い結果も考慮すると，表6, 7の結果は，同一ページに対するアクセスの競合は少ないことを示している．本システムのように無効化型の無矛盾化プロトコルを用いている場合，無効化するノードのみを選択してメッセージを送るマルチキャストは有効であると思われる．

表7: クリーンライト時における他ノードの無効化回数（water, ノード数8）

node	128	256	512	1024
0	0	0	0	0
1	96995	73703	59190	41431
2	4470	11339	16205	10939
3	1819	4989	8173	4414
4	1073	812	727	363
5	3	2	2	4
6	2	2	3	2
7	18	18	19	18

DSMを（アプリケーションを実行していない）複数のノードに分散し配置して実行した．DSMを1台のノードのみに配置した場合と比べ，どの程度の速度向上が得られたか，結果を表8に示す．共有メモリへのアクセスの分散によってサーバノードのボトルネックが回避され，わずかではあるが速度向上が見られる．現時点では，通信コスト自体の影響が大きく，**Memory Manager**の処理時間が全体に占める割合は小さいので，大幅な速度向上が得られないようである．

表8: DSMを分割した場合の速度向上

(page 1024byte)

DSM node	速度向上（対1）
1	1.000000
2	1.008524
4	1.023995
8	1.029715

（8台のノードでmp3dを実行）

6 まとめ

これまでの研究結果から，本システムを用いた並列プログラムの実行では，ある程度の台数効果による速度向上が確認された．また，mp3d, waterによるシステムの評価を行なった結果，性能向上の妨げとなっている一番の原因は，ネットワーク自体のバンド幅や速度であると思われる．今後のATM, FDDI, CDDI等の高速ネットワークの技術によって，この問題が改善されることにより，一層の性能向上が期待される．これより，ソフトウェアDSMは，高速ネットワーク時代において有望な技術であると思われる．

問題点としては，管理する単位がページ単位と比較的サイズの大きいものであることから，プログラムの共有データを共有メモリのアドレス空間へどのようにマッピングするかによって，フォールスシェアリングの影響で通信のコストを増加させたり，ノード間で同期地点までの処理時間のバラツキが大きくなるなど，性能に大きな影響を与えるということが考えられる．また，現在はアプリケーションを実行するノードと，DSMの存在するノードは別ノードであるが，実際には同一ノードにDSMとアクセスプロセスが存在するので，共有データの存在位置によって，ネットワークを介したアクセス時間が異なる．これより，ネットワーク環境においてソフトウェアDSMを用いて，ノード間通信の比較的少ない効率の良い並列処理を行なうには，コンパイラ等の支援が不可欠であると思われる．

他の問題点としては，同期機構として用いているロックやバリアが，現時点では共有メモリを用いた単純な機構で実現しているので，システムの管理が簡素であるという半面，ネットワークを介する通信が多くなり効率が良くないようである．

今後は，SPLASHに含まれるmp3d, water以外の並列プログラムにより本システムの評価を行ない，本システムで用いた無矛盾化プロトコル以外のプロトコルとの比較や，アプリケーションの粒度やシステムのページサイズ，共有データのマッピングの違いによってフォールスシェアリングが台数効果へ及ぼす影響の考察などを行なっていく．また，ヒットした場合のアクセス時間の高速化も重要な課題である．OSの軽量化や，システム自身のマルチスレッド化によって，コンテキスト切り替えやシステムコールのオーバヘッドの改善にも対応していく必要がある．以上のような点を考慮し，処理能力向上のための本システムの改善点や，分散共有メモリの有効性・問題点などをより明確にしたい．

参考文献

[1] 藏前他：“ネットワーク環境下における分散共有空間へのアクセス方式”，電気関係学会，G316, 1992.

[2] Bill Nitzberg and Virginia Lo: “Distributed Shared Memory: A Survey of Issues and Algorithms”, COMPUTER, pp.52-60, August 1991.

[3] 中條他：“ネットワーク結合型並列計算機上の仮想共有メモリシステムにおける無矛盾化プロトコルの性能評価とハードウェアによる実現”，並列処理シンポジウム，pp.45-52, 1991.

[4] Jaswinder Pal Singh, et al.: “SPLASH: Stanford Parallel Applications for Shared-Memory”, Computer Systems Laboratory, Stanford University, CA 94305.

[5] CEAIG C. DOUGLAS, et al.: “PARALLEL PROGRAMMING SYSTEMS FOR WORKSTATION CLUSTERS”, Yale University Department of Computer Science Research Report YALEU/DCS/TR-975, August, 1993.

[6] Pete Keleher, et al.: “TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems”, Rice COMP TR93-214, November 1993.