計算機アーキテクチャ　128－17
ハイパフォーマンス　70－17
コンピューティング
（１９９８．３．６）

計算機クラスタにおけるユーザレベルソフトウエア分散共有メモリの性能評価

緑川　博子、飯塚　肇
成蹊大学工学部
〒１８０－８６３３　東京都武蔵野市吉祥寺北町３－３－１

ＣＰＵ、通信性能の異なる２つの計算機クラスタにおいて、ユーザレベルの分散共有メモリシステムTreadMarksを用い、数種のベンチマークプログラムに対し、ＰＶＭと性能、プログラムの記述性を比較した。計算通信比、通信回数、通信量による違い、キャッシュ影響などがみられた。TreadMarksは、lazy release consistency、diff転送など、性能向上のための工夫がされているが、ページベースであるため、false sharingによってメモリ一貫性維持のためのメッセージが莫大になる場合があり、一貫性のための粒度には問題がある。オブジェクトベースの小さい粒度をもつAdsmithについても、TreadMarks,PVMと比較した。プログラマによる通信性能改善のための細かな指定が可能になる反面、プログラマへの負担増加、プログラムによっては記述が困難な場合がある、などの問題点がある。

The Evaluation of User level Software based Distributed Shared Memory System on computer cluster

Hiroko Midorikawa, Hajime Iizuka
Department of Information Sciences, Seikei University
3-3-1 Kichijoji-kitamachi, Musashino-shi, Tokyo 180-8633, Japan

User-level software-based distributed shared memory systems, TreadMarks and Adsmith, were evaluated on two computer clusters which have different computing and communication performances. Both DSM systems achieved comparable performance to PVM in several programs. In TreadMarks, which is a page-based system, consistency maintenance message traffic increases significantly because of false sharing in some benchmark programs. Adsmith, object-based system, can reduce maintenance message traffic under programmer's control, but it increases programmer's load, and it has a difficulty to describe some programs.

１．はじめに

様々な分散共有メモリシステムが注目をあつめているが、本報告では、計算機クラスタ上で容易にインストールできるユーザレベルDSMソフトウエア２種を、CPU、ネットワーク性能の違う２種の計算機クラスタ上にインストールし、数種のベンチマークプログラムを実行することにより、性能、プログラム記述性について評価した。

２．ソフトウエア分散共有メモリシステム

今回評価したのは、TreadMarks[1]とAdsmith[2]である。TreadMarks(以下Tmkと記す)は、Rice大で開発されたページベースのシステムで、lazy release consistencyを採用している。共有ページへの最初の書き込みで、そのページのコピー(twin)を作成し、一連の書き込み後、書き込み後のページとの比較(diff)に用いる。lazy releaseであるため、各release時にdiff作成を行わず、次のacquire時まで処理は延期され、より高い性能を得る工夫をしている。ここで用いたのは、version0.10.1.2(PCクラスタ)とversion0.10.1(WSクラスタ)で、通信にはUDPを用いている。

Adsmith(以下Admと記す)は、国立台湾大学で作成されたオブジェクトベースのシステムである。PVMの上にC＋＋のライブラリとして実装している。Adsmithは、release consistencyを用い、一貫性粒度はユーザの指定したオブジェクトで、通常ページよりは小さい。データアクセスには、load/store方式の命令を使用する。loadは最初の読み込みに、storeは最後の書き込みに用いる。それ以外のアクセスは、キャッシュされたローカルコピーに対して行われる。storeには、write through 方式を用い、それ以外は、write back方式を用いる。ここで用いたシステムは、version1.8.0dである。

３.計算機クラスタシステムとプログラム

表１に、今回用いたシステム環境を示す。ＷＳクラスタは１８台接続されているが、Tmkのシステム制限で８台までの計算機しか用いることができないため、多くの測定は８台までの台数で行なった。

表２に評価に用いたベンチマークプログラムのサイズを示す。EP, IS, 3D-FFTは、NAS並列ベンチマーク、Waterは、SPLASHのものである。その他

	PCクラスタ	WSクラスタ
CPU	Cyrix 6X86-P120-GP	SPARC 50MHz
メモリ	64 MB	64MB
ネットワーク	100Mbps Ether Hub	10Mbps Ether
OS	FreeBSD 2.2.2	SunOS 4.1.3
台数	8	18

表1 システム環境

Programs	Problem Size
EP	2**28
IS.medium	N=2**23, max=2**10,R=10
IS.large	N=2**23, max=2**15,R=10
3D-FFT	64 x 64 x 64, R=6
Water-288	288 moledules,R=5
Water-1728	1728 moledules,R=5
SOR-z, SOR-nz	1024 x 3072,R=50
TSP	19 Cities
Qsort.qsort, bubble	N/minblock=512K/1024(SUN), N/minblock=1000K/1000(PC)
Jacobi	1536 x 1536,R=20

表2 プログラム

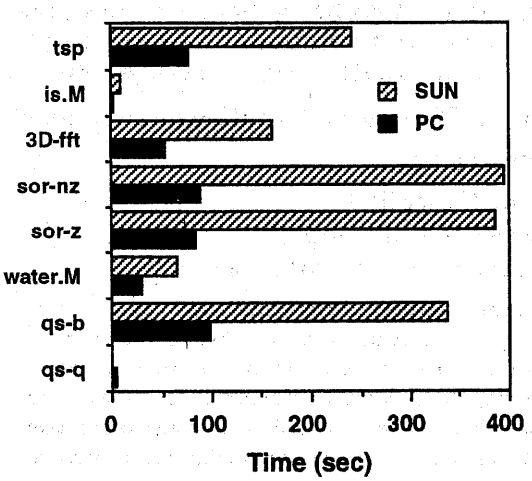

図1a 逐次実行時間

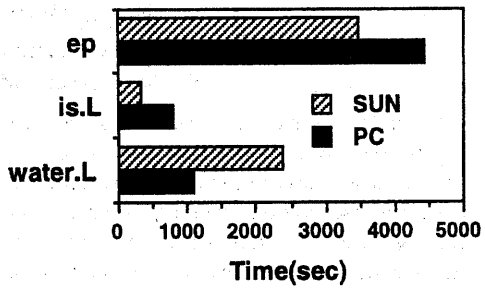

図1b 逐次実行時間

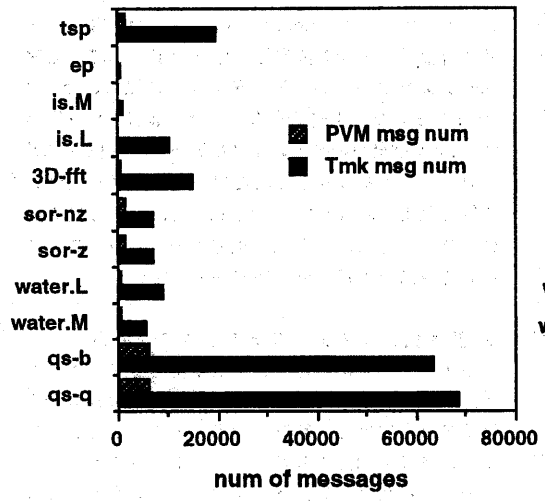

図2 メッセージ数（8プロセッサ）

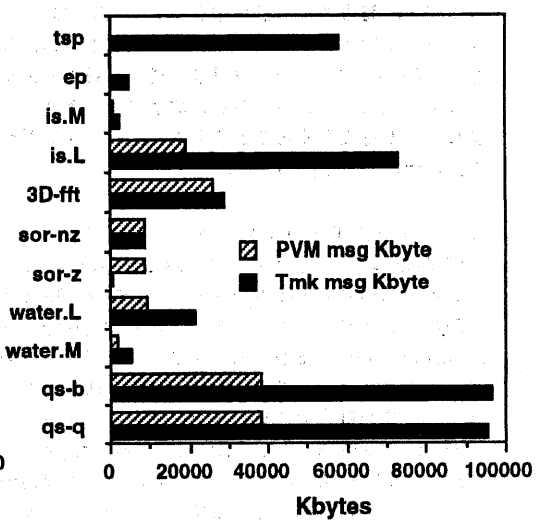

図3 通信量（KByte、8プロセッサ）

に、SOR(Successive over relaxation), TSP, Quick Sort, Jacobi などについても行なった。

4.TreadMarksとPVMとの性能比較

4．1 プログラムの特徴

それぞれのプログラムを2種のクラスタの計算機1台で逐次アルゴリズムで実行した場合の実行時間を図1a、図1bに示す。また、8台の計算機で、TmkとPVMを実行した時の、それぞれの通信メッセージ数と、通信データ量の比較を図2, 図3に示す。

4．2 並列処理速度向上比

計算機1台～8台で、TmkとPVMの各プログラムの実行時間から、図1の逐次プログラムの実行時間をもとに並列処理速度向上比を計算した。図4.1～図4.10は、PCクラスタの速度性能比、図5.1～図5.6は、WSクラスタの速度性能比を示す。

4．2．1 EP(Embarrassingly Parallel)

浮動小数計算、乱数発生などが多く、各プロセッサ処理の独立性の高いEPは、計算に対し殆んど通信が無視できる程度であるため、図4.3のように、PVM、Tmk共計算機8台で7.6、7.2の性能が得られ、PVMとの差が少ない。

4．2．2 SOR (Successive Over Relaxation)

SOR-Zは、2次元配列のエッジ要素を1、その他を0に初期化し、SOR-NZは、全ての要素を乱数で設定する。SOR-Zは、SOR-NZに比べ、データ更新が少ないため、diffの転送が少なく、メッセージ数は、SOR-NZと同じ(図2)であるが、データ転送量(図3)が少ない。SOR-NZのデータ転送量はPVMとほぼ同じで、どちらもメッセージ数はPVMの5倍となる。2つのプログラムの差は、図4.1、4.2のようにSOR-Zが若干性能がよいものの、PC、WSクラスタいずれにおいても差は小さい。したがって、全体の計算量に対し、通信データ量の違いによる性能の劣化は、大きく影響しなかったと思われる。

一方、2つのクラスタで比較すると、PCクラスタでは、TmkとPVMは計算機8台で5.2、5.9で同程度の性能であるのに対し、WSクラスタでは、PVMとTmkとの差が大きい。この原因の1つは、WSクラスタがPCクラスタに比べ、CPUが遅いため、PVMとTmkのメッセージ回数の違い(ソフトウエアオーバヘッド)によるものと考えられる。

4．2．3 Water

各プロセッサに割り当てられた分子同士の力を計算するために、データ交換が多く、メッセージ数、通信量ともにTmkはPVMに比べ多い。ただし分子数が多いwater.largeでは、それ以上に計算量が増えるため、並列向上がよい。water.mediumは、PCクラスタでは、water.largeに近い処理性能が得られているのに対し、WSクラスタでは、メッセージ数が多いためか、Tmkの性能が4.5程度と飽和している。

4．2．4 IS (Integer Sort, Ranking)

各プロセッサが割り当てられたkeyに対し、全体のデータをみて順位づけを行うため、広範囲のデータリードとkeyの出現範囲に応じたライトが発生する。したがって、アクセス局所性が少ないため多量なdiffが発生し、計算／通信比が低い。このため図4.7に示すように並列性能は最高2.8と低い。Tmkは、PVMにくらべ、IS.mediumで、メッセージ数6.7倍、データ量4倍、IS.largeでは、データ量3.9倍、メッセージ数72倍に達する。

さらに、PCクラスタでは、IS.largeを1台の計算機で実行すると、キャッシュ範囲をこえるため、ある一定以上のサイズ(データ数とkeyの範囲)を実行すると極端に処理時間が遅くなることが分かった。そのため、図4.8に示すように並列性能が極端に高く計算されてしまう。本来の計算／通信比以外の要素による性能劣化の例である。言い替えれば、並列化により処理データを分割して、データアクセス局所性を高めればキャッシュの効果が引き出され、台数効果以上の性能が出せる場合もある。

4．2．5 3D-FFT

3次元FFTの処理では、2次元面を各プロセッサに割り当てているため、最初2つの次元でのFFTでは通信は発生せず、第3番目の次元の計算前にデータ移動が必要となり通信が発生する。この移動は各プロセッサ、全領域に対する割り当て変更にも匹敵するため、各プロセッサ間でdiffの送り合いになり、また全体のデータ転送量も多い。したがって、並列性能は、WS、PCクラスタにおいて、Tmkは、最高2.5、4.1、PVMでは3.1、4.0になっている。

プログラム記述性の点では、FFTのようにデータアクセスパターンが各プロセッサで、処理の段階に応じて変化するような場合、PVMでは、非常にプログラム作成が大変になる。したがって、この種の応用には、共有メモリ型のプログラミングモデルが非常に優れている。

4．2．6 Quick Sort

2つに分割された部分列の1つを自分のプロセッサで処理し、残りをキューに登録する。各プロセッサは自分の処理部分列がなくなり次第、キューから処理する部分列を取り出す。ただし、部分列サイズが閾値以下の時は、登録せずにbubble sortまたは、quick sortを適用する。キューアクセス時の排他制

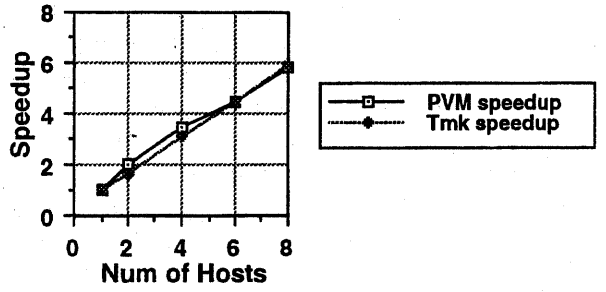

図4.1 SOR-Z PC 速度向上比

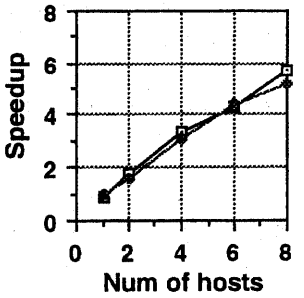

図4.2 SOR-NZ PC 速度向上比

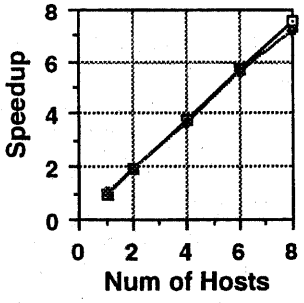

図4.3 EP PC 速度向上比

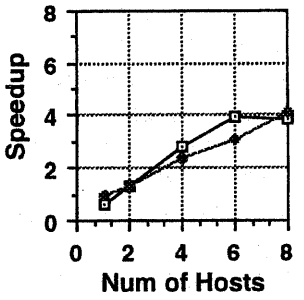

図4.4 3D-FFT PC 速度向上比

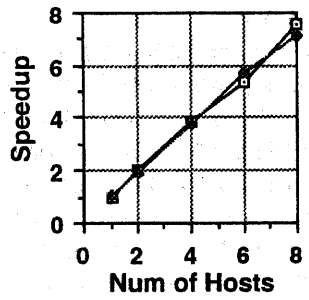

図4.5 Water.Large PC 速度向上比

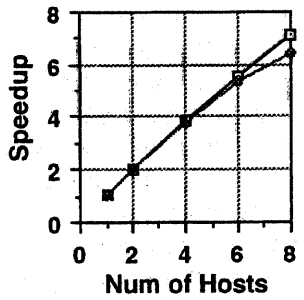

図4.6 Water.Medium PC 速度向上比

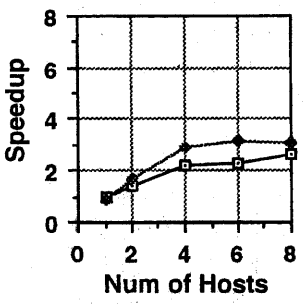

図4.7 IS.Medium PC 速度向上比

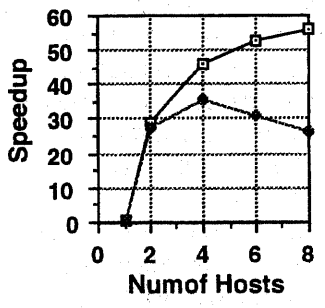

図4.8 IS.Large PC 速度向上比

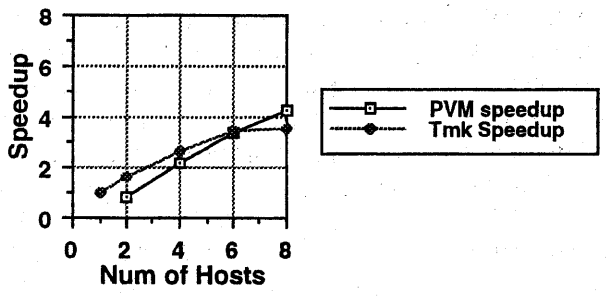

図4.9 QSort.bubble PC 速度向上比

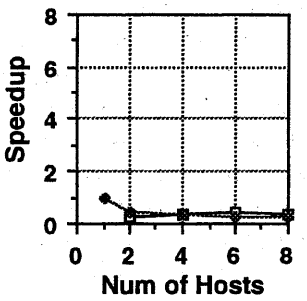

図4.10 QSort.qsort PC 速度向上比

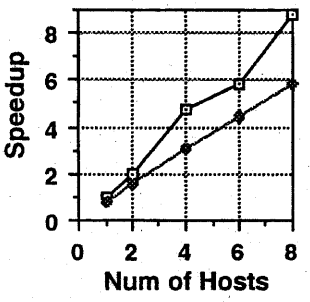

図5.1 SOR-Z SUN 速度向上比

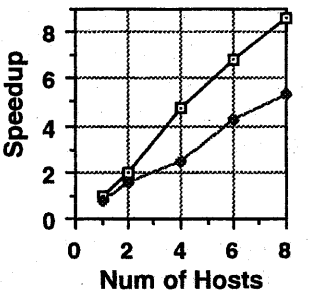

図5.2 SOR-NZ SUN 速度向上比

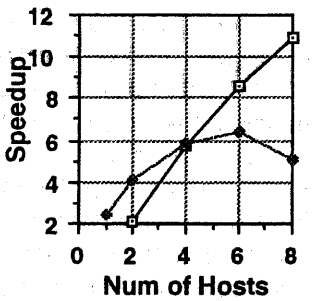

図5.3 Qsort.bubbleSUN 速度向上比

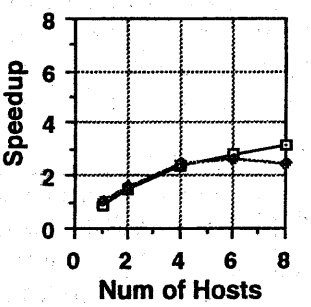

図5.4 3D-FFT SUN 速度向上比

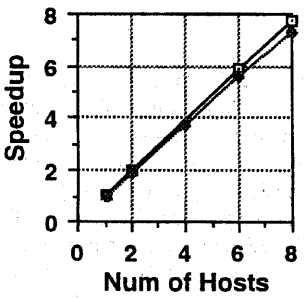

図5.5 Water.Large SUN 速度向上比

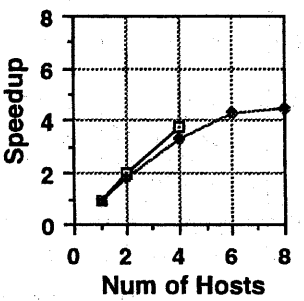

図5.6 Water.Medium SUN 速度向上比

御がいる上、各プロセッサのアクセス要素は全く独立であるにもかかわらず、false sharingのために同じページに割りあてられた部分列アクセスのたびにdiffが発生するため、メッセージ数、量とも非常に多い。その反面、比較処理は簡単なため計算／通信比が極端に小さい。図5.3に示すようにCPU速度の遅いWSクラスタでは、通信が少ないPVMは、計算量の多いbubble sortを用いると８以上の並列性能が得られ、Tmkでも6.4ほどの性能が得られている。しかし、図4.9のようにCPU速度の速いPCクラスタでは、PVM、Tmkの性能も、4.2、3.5どまりである。また、計算量の少ないquick sortをを末端処理に使用すると、図4.10のように並列性能はでない。この種の計算／通信比の低い応用は、CPU速度と通信速度の差が益々拡大する傾向にある最近のシステムでは、並列化そのものに意味がなくなる可能性もある。

5.TreadMarks、Adsmith、PVMとの比較

メモリー貫性の方式と粒度などが違うAdmとTmk、PVMで同じ応用を記述してみた。

図6はJacobiプログラムの例である。図4、5同様、逐次プログラムの実行時間で速度性能比を計算している。これよると、PVMの上に実装されたAdmの性能がPVMよりも高い。これは、Admが転送の必要な所だけにload/store命令を使用することができるためである。この応用では、各プロセッサで隣接するデータ行のみを通信し合うという記述ができるため、PVM、Tmkなどにくらべ、データ転送量が少なくなり、性能が向上する。PVMと同様のデータ量を転送するよう記述を変更すると(Adm db_allの線)、AdmはTmkと同程度の性能になる。したがって、Tmkなどの粒度の大きいページベースのシステムで起こりがちな、不要なデータ転送による性能劣化は起こりにくく、多くの場合性能向上が見込める。

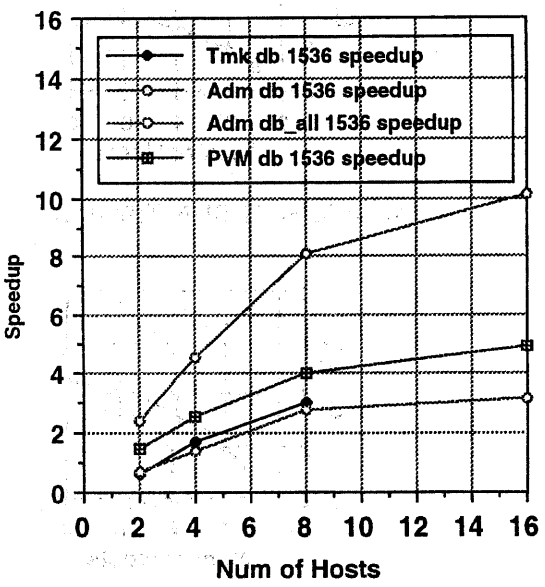

図6 Jacobi SUN 速度向上比

しかし、Adsmithでは、いつ、どこで、何に、load/store命令を使用するかはプログラマに任されているため、性能向上のためのチューニングができる反面、プログラマの負担が増す。

また、最初に定義したオブジェクトの単位で更新処理がされるため、アクセスするデータ範囲が静的に固定化している場合には、その範囲をオブジェクトとして定義すれば、無駄な通信を無くし、性能向上が達成できる。しかし、4.2.5のFFTの例のように、処理の途中で各プロセッサのデータアクセスパターンが変更されたり、初期オブジェクト宣言時とは異なるサイズで更新がしたい場合などには、Adsmithには記述能力がない。したがって、使用できる応用の種類が制限される。

6．おわりに

多くのプログラムで、Tmk、Admは、PVMに近い性能が得られた。しかし、共有メモリプログラミングにおいても、PVMによる並列プログラム作成時と同様に、通信に関する特性を考慮して作成することが重要で、応用、アルゴリズムによってはPVMとの性能差がでてしまったり、並列性が十分得られない場合もある。しかし、プログラム記述の観点では、多くの場合、Tmkは非常に記述が楽で、特にどの計算機がいつどのデータをアクセスするかが把握しにくいような応用の場合は、PVMよりも優れている。Adsmithは、一般的に性能ではTmkに優るものの、ある種の応用では、PVMよりもかえって記述しにくくなることがある。

[1]P.Keleher, S.Dwarkadas, A.L.Cox and W.Zwaenepoel "TreadMarks: Distributed shared memory on standard workstations and operating system", Proceedings of the 1994 Winter Usenix Conference, pp.115-131, January 1994.

[2]Wen-Yew Liang, Chun-Ta King and Feipei Lai "Adsmith: An Efficient Object-Based Distributed Shared Memory System on PVM", Proceedings of the 1996 International Symposium on Parallel Architecture, Algorithms and Networks, pp.173-179, June 1996.