ソフトウエア分散共有メモリシステム SMS

緑川　博子†　　大橋　祐介†　　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
E-mail: †midori@is.seikei.ac.jp

あらまし
分散共有メモリシステム SMS は，PC などの計算機とそれを結ぶネットワークから成る計算機クラスタにおいて，分散共有メモリ環境を提供するユーザレベルのソフトウエアシステムである．SMS は，汎用のハードウエアとソフトウエアのみを前提としていながら，汎用計算機クラスタ上での共有メモリプログラミングを可能にする．メモリコンシステンシモデルには，新しく提案した“暗黙にロックと共有データの関連付けを行う”改良型のエントリコンシステンシモデル IBEC を用いている．簡単なマイクロベンチマークでの評価では，メッセージパッシングプログラミング用の代表的なシステムである PVM，MPI や，ソフトウエア分散共有メモリシステムの代表格である TreadMarks と比較しても，同等以上の性能を得ることができた．

キーワード　分散共有メモリ，並列処理，クラスタ，メモリコンシステンシーモデル

The Software Distributed Shared Memory System: SMS

Hiroko MIDORIKAWA†,　Yusuke OHASHI†,　and Hajime IIZUKA†

†Faculty of Engineering, Seikei University
Kichijoji　Kitamachi3-3-1, Musashino-shi, Tokyo, 180-8633 Japan

E-mail: †midori@is.seikei.ac.jp

Abstract
A distributed shared memory system, named SMS, is a user-level software system. It　provides a virtual shared memory environment on a cluster consisting of computers connected by a communication network.　Although the SMS requires only commodity hardware/software, it enables users to write parallel programs with a shared memory programming model.　The employed memory consistency model is a newly proposed IBEC (Implicitly Binding Entry Consistency). It is a variant of the entry consistency model, but binds shared data with a lock variable implicitly.　The performance of SMS is comparable or better than typical message passing programming libraries, such as PVM and MPI, and the representative software shared memory system, TreadMarks.

Key words　distributed shared memory, parallel processing, computer cluster, memory consistency model

1. はじめに

本システムは，ＰＣもしくはワークステーションとそれを結ぶネットワークからなる汎用の計算機クラスタにおいて，分散共有メモリを実現する，ユーザレベルソフトウエアである．汎用のハードウエアとソフトウエアのみを前提としていながら，記述性のよい共有メモリプログラミング環境が実現でき，並列プログラムの学習や実行を容易にし，一般ユーザが並列処理をより身近に行える．

2. システムの特徴

本システムは以下のような特徴を持つ．

- **ユーザレベルソフトウエアによる実装**
  UNIX 系 OS の稼働を前提にしているが，OS に手を入れることなく，一般ユーザが容易にインストールして使用できる．
- **ネットワークを選ばない TCP/UDP 通信**
  プロセッサ間通信にはTCPもしくはUDPのいずれかをシステム時に選択できる．特殊なネットワークの機能を使うことを前提としていないため，ギガビットイーサ，ATM，あるいはMyrinetなど，TCPもしくはUDPをサポートする通信媒体であればどのようなものでも使用することができる．
- **緩和型メモリコンシステンシモデル IBEC**
  IBEC（Implicit Binding Entry Consistency）と呼ぶモデルを提案し，ロック使用時のメモリ更新情報の通信量を少なくし，従来のECにあった煩雑な指定をなくし，使いやすさと性能とのバランスをとっている．
- **柔軟なプログラム記述機構**
  コンディション変数を備え，クリテイカルセクションの実行制御などで，ポーリングを不要とした．共有データ変数割付時や実行中にデータを管理するページマネージャを指定／変更できるほか，バリア呼び出しの際にバリアマネージャの指定なども可能で，データアクセスの局所性や，１プロセスへの処理の集中を避け，より実行性能のよいプログラムの作成を可能にした．
- **ハードウエア独立な柔軟な仮想プロセス実行**
  単一CPUのPCクラスタからSMPクラスタまで，ハードウエア構成上のPC数，CPU数に対し独立に仮想プロセスを作成することが可能であり，前提とするハードウエア構成とは独立に並列プログラムを作成できる．

3. メモリコンシステンシモデル

本システムでは，分散メモリの一貫性をとる方式として，IBEC(Implicit Binding Entry Consistency)[1]というモデルを採用している．エントリコンシステンシモデル(EC)[2]と同様にロック同期変数に関連のある共有データの更新情報のみを，次のロック獲得者に渡すが，従来の EC のように共有データと同期変数の明示的なバインデイングは行わない．プログラム記述上は，リリースコンシステンシモデル(RC)[3]やレイジーリリースコンシステンシモデル(LRC)[4]と何ら変わりがない．ただしその更新のタイミングと，実際の更新箇所，転送更新データが異なる．

共有メモリシステムのプログラミングで使われるｐスレッドプログラミングなどでは，ある共有データに競合するアクセスがあるとき，このデータに関連するロック変数を定義し，プログラマは暗黙にこのロック変数と該当共有データを関連づけて使用する．IBEC でのプログラム記述はこれと同じで，プログラムで同期変数と共有データの関連付けを図１(b)のように明示的に記述する必要はない．従来の EC でプログラマに強制している共有データとロック変数の明示的な関連付けは，プログラマにとってはむしろ煩雑であるからである．一般の逐次プログラムでどの変数とどの変数を関連づけるかは，プログラマの責任であるのと同様に，IBEC では，どのデータ変数とロック変数を関連づけるかはプログラマが管理する．

RC や LRC との大きな違いは，グローバルタイムを元に，ロック解放時のすべての共有データ更新情報をロック獲得者に見せるわけではないということである．すなわち，IBEC では，あるロック変数で暗黙に関連づけられた共有変数の更新のみがそのロック変数を獲得したものにだけ見えるようにする．したがって，異なるロック変数の獲得，解放者には，それらの更新情報は必ずしも見えないし，伝達される更新情報はロック毎に独立となる．タイムスタンプは，ロック変数毎に独立で，バリア時にすべてのデータの更新とすべてのタイムスタンプのリセットが行われる．図２にその違いを示す．図中 Tn(Lx)はロック Lx に関するクロックタイムである．

同じプログラムを記述した場合，どのデータ更新が保証されるのか，IBEC と LRC との違いを図３に示す．IBEC における暗黙の関連付けとは，ロックが獲得されてから解放されるまでに変更された共有デー

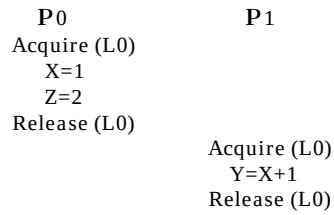


図 1(a)　RC の記述（暗黙の関連付け）

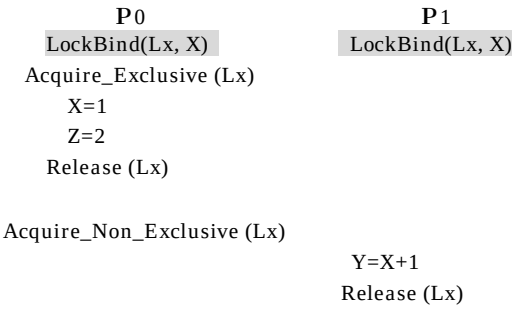


図 1 (b) EC の記述（明示的な関連付け）

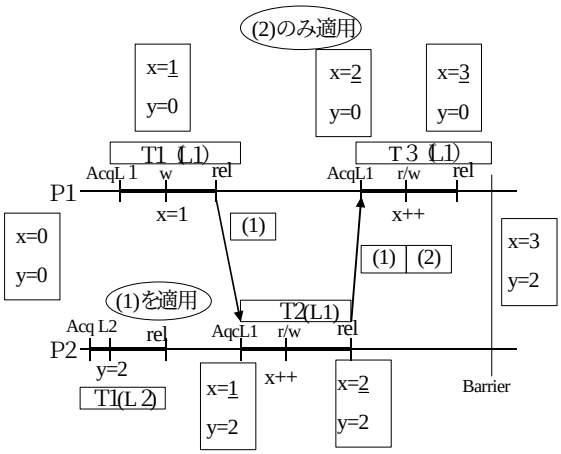


図 2 (a)　IBEC ロック毎に独立なインターバルタイム

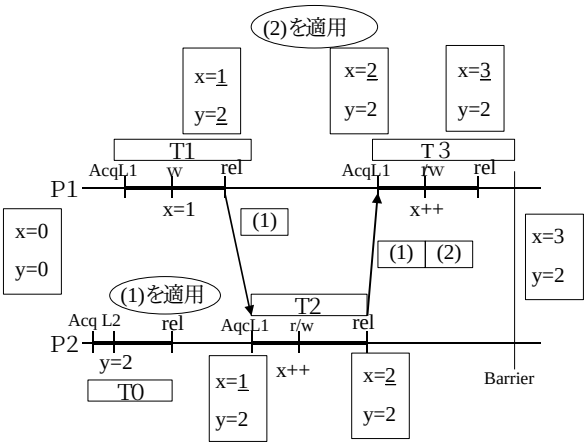


図 2 (b)　LRC グローバルインターバルタイム

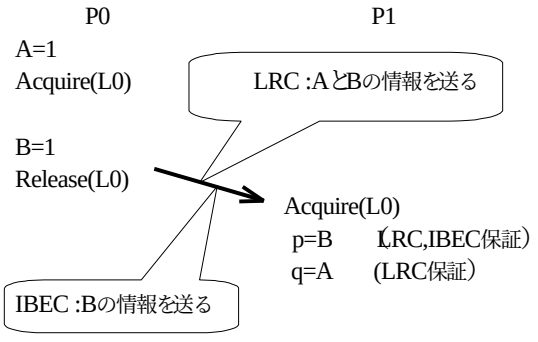


図 3　LRC と IBEC の更新保証の違い

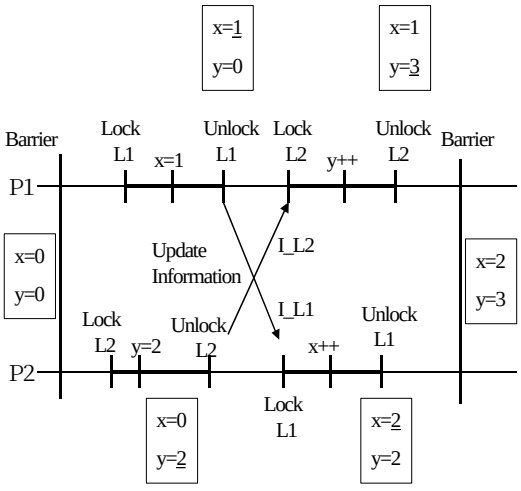


図 4　IEBC の共有データ更新方式

タと解釈しているので図中の B の値しか次のロック獲得者に渡さない．LRC は，基本的にリリース時点でのデータのすべての更新を次のロック獲得者へわたす．また，図１(a)のような記述では，X と Z の両方が L0 に関連づけられる．したがって，従来の EC では明示的に記述された変数 X のみが更新情報として次のロック獲得者へ渡されるが，IBEC では X と Z の両方の更新情報が渡される．

図４は，IBEC において，２つのプロセッサが２つのロック変数 L1, L2 をそれぞれ共有変数 X と Y に関連づけて記述したときのそれぞれのプロセッサにおける共有データ更新の様子である．下線の値が変更部分である．２つのバリアの間，P1 と P2 の x と y は同じではない．しかし，共有変数 x,y の更新は，暗黙に関連づけられたロック変数 L1,L2 を使用することにより，矛盾なくプロセッサ間で受け継がれる．最後のバリア後，P1,P2 のメモリは完全に同一になる．

4. SMS プログラムインタフェース

SMS システムのプログラムインタフェースとして，表１に示すような SMS 定数と，C 言語用 SMS 関数を提供する．sms.h をインクルードし SMS ライブラリ（libsms）とリンクすることで，共有メモリプログラミングスタイルの実行プログラムが作成できる．図５に簡単な SMS によるプログラム例を示す．

5. 実装

5.1 メモリアクセス検知とページの管理

SMS は共有データアクセス検知には OS のメモリ保護機構を用いているので，メモリ管理単位であるページを基本にしたシステムとなっている．ページの状態遷移図を図６に示す．

5.2 メモリコンシステンシのための更新情報

IBEC を用いた SMS ではページベースで，更新情報にページ差分（diff）を用いているが，EC の本来の性質である，関連づけられたデータ以外は，たとえ同一ページ内の変数あっても，更新情報を渡さないという点で大きく異なる．TreadMarks[5]では大域的にコンシステンシをとるという RC の性質を引き継いでいるため，あるページの特定データのみの更新で十分な場合にも，同一ページ内のその他のデータに関する更新もすべて反映されるようなモデルである．実際の更新処理は，該当ページがアクセスされるまで遅らされるが，無効化プロトコルの場合には，ページフォルト時にそのページに関する更新情報を複数のプロセッサから集めてこなければならない．IBEC の場合には，単に前回のロック獲得者からの情報を得るとういう方式を現在とっている．たとえ同一ページ内のデータであっても，ロック変数との暗黙に関連する変数の更新情報のみなので，本質的に偽共有の弊害をほとんど受けない．

また，その共有変数に関する最新情報は常に，その変数を処理するプロセッサに存在することになり，プログラム実行中にその役割が変化するようなことがあっても，自動的にその共有変数に関する情報に責任をもつプロセッサがロック変数の受け渡しと共に，移動していく．SMS では，共有変数の初期保有プロセッサ（ページマネージャ）をプログラムで静的に指定できる．

5.3 バリア，ロック同期

SMS にはバリアとロック獲得／解放の同期操作がある．現在のところ，バリアを管理するバリアマネージャ（プロセス）はユーザがバリア同期関数で指定できる．現在のところ，ロックに関しては，単一のロックマネージャがすべてのロックを管理しているが，バリア同期毎に前回のバリアマネージャとは異なるプロセスに割り当てるようにしている．それぞれの更新差分情報はそれぞれ diffB と diffL に分けて管理している．

5.4 条件変数制御

ロック変数を使用して共有データの排他制御する場合，CPU 時間を無駄に浪費するポーリング処理を避けて，共有データの値の変化などを知らせる機構として，条件待ち関数と条件成立通知関数を用意し，共有データの排他制御をより円滑にしている．すべての条件変数は，単一のコンディションマネージャがロックマネージャと連携して行う．デフォルトではロックマネージャと同一のプロセッサになる．

5.5 データ更新差分の作成と適用における粒度

更新情報 diff は，ページコピー（twin）と変更後のページをバイト単位で比較して作成するのが基本であるが，共有データオブジェクトのデータ型単位に比較して作成したほうが，diff の数，総量とも低減でき，全体としての更新情報は減少する．また diff 適用の際も不連続なバイト単位の diff を多数適用するよりも，数少ない diff を連続バイトで適用したほ

表 1　SMS の定数及び関数

sms_nproc　：並列稼働プロセス数
sms_proc_id　：プロセス ID(0,1,2,...)
void sms_startup(int argc,char *argv)　初期化関数
void sms_shutdown()　正常終了関数
void sms_error(char *errmsg)　エラー終了関数
void *sms_alloc(int size,int PageM_pid) 共有データ割付関数
void *sms_calloc(int num, int itemsize,int PageM_pid)
void sms_change_page_manager(void *adr,int size,int PageM_pid)ページマネージャ変更関数
void sms_barrier(int BarrierM_pid)　バリア同期関数
void sms_lock(int Lock_id)　ロック獲得関数
void sms_unlock(int Lock_id)　ロック解放関数
void sms_cond_wait(int cond_id, int Lock_id) 条件待ち関数
void sms_cond_signal(int cond_id)　条件成立通知関数
void sms_cond_broadcast(int cond_id) 条件成立放送関数

```
#include<stdio.h>
#include<sms.h>   /* sms 用ヘッダーファイル */
#define NUM 1024
void main(int argc,   char *argv[ ])
{
   int *data, *max, n, i;
   FILE *frp;
   frp=fopen(argv[1],"r"); /*データファイル*/
   sms_startup(argc, argv);     /* sms 起動関数 */
   n=NUM/sms_nproc;
   /* 共有データ領域確保 */
   data=(int *)sms_calloc(NUM, sizeof(int),0);
   max =(int *)sms_alloc(sizeof(int),0);
   /*マスタープロセスによるデータ初期化 */
   if(sms_proc_id==0){
      for(i=0;i<NUM;i++) fscanf(frp,"%d",&data[i]);
     *max=0;
   }
   /* 全プロセスによる最大値探索 */
   sms_barrier(0);          /* バリア同期 */
   for(i=sms_proc_id*n;i<(sms_proc_id+1)*n;i++)
   { sms_lock(0);         /* ロック獲得関数 */
        if(*max < data[i])   *max = data[i];
     sms_unlock(0);     /* ロック解放関数 */
   }
   sms_barrier(0);          /* バリア同期関数 */
   /*マスタープロセスによる結果出力 */
   if(sms_proc_id==0) printf("max =%d¥n",*max);
   sms_shutdown();         /* sms 終了関数 */
}
```

図５　SMS 関数を使用した簡単なプログラム例

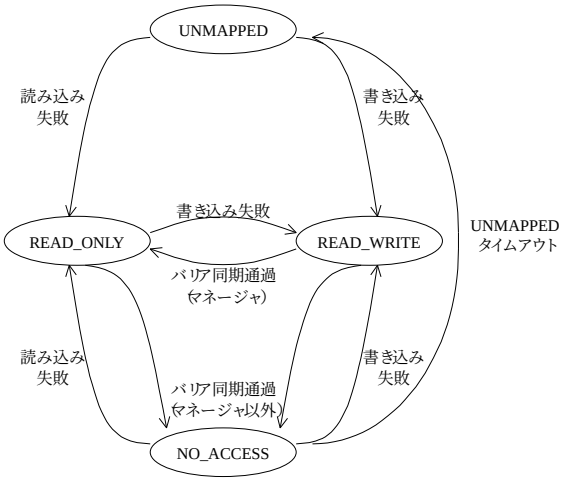


図６　ページ状態遷移図

環境１　8PC (8CPUs) ■CPU:　Intel MMX Pentium 166MHz ■メモリ：64MB ■OS:　FreeBSD2.2.2R ■ネットワーク：　100Mbps Ether Hub／switch Hub
環境２　8PC (8CPUs) ■CPU:　Intel Celeron 400MHz ■メモリ: 128MB ■OS:　RedHatLINUX6.0(kernel 2.2.5-15) ■ネットワーク：100Mbps Ether switch Hub／Myrinet
環境３　4SMP (8CPUs) ■CPU:　Intel PentiumIII　700MHz　DualCPU ■メモリ: 256MB ■OS:　RedHatLINUX6.2(kernel　2.2.14-5.0) ■ネットワーク：　100Mbps Ether switch Hub

表２　評価環境

うが，処理を高速化できる．このような観点から，ユーザが共有データ割付時に，sms_calloc 関数を使用して，diff の比較粒度を指定することができる．引数で指定された要素サイズで，diff の作成，適用が行われる．したがって，たとえ同じページにある共有データであっても，diff 作成／適用の粒度は，共有データ変数毎に異なっている．

5.6 プロセス間通信

プロセッサ間は，TCP または UDP によるシグナル駆動入出力を用いている．いずれのプロトコルの場合にも，続けざまに多くのメッセージが到着しないように SMS の通信方式を制御している．全プロセッサ間に，独立のソケットを使用している．

6. SMS の基本関数の評価

6.1 評価システム

基本性能評価には表２の３種の環境を用いた．８台の PC クラスタ，もしくは４台の２CPU の PC クラスタで，OS は Linux と FreeBSD である．ネットワークは 100Mbps イーサネットもしくは Myrinet を使用できる．本報告では 100Mbps イーサを使用した．

6.2 基本関数性能

8PC 使用時の PC クラスタ（環境２）と 4PC 使用時の PC クラスタ（環境３）における SMS の基本関数の性能時間を表 3 に示す．プログラムの起動には，rexec／rsh を用いており，SMS では PVM のようなデーモンを起動しないため，他の関数に比べ時間がかかる．rexec/rsh の起動順によって，各 PC の startup 時間が異なるのでそれぞれの環境で１台目と４台目の PC の値を示した．

7. 他システムとの比較

7.1 実験プログラム

マイクロベンチマークとして多体問題とマンデルブロー計算[1]について，広く使われているメッセージパッシングソフトウエアである PVM，MPICH と比較した．さらに代表的なソフトウエア分散共有メモリシステムである TreadMarks との比較も行った．

7.2 メッセージパッシングライブラリとの比較

PVM と MPICH，SMS の UDP 版と TCP 版とを環境２で比較したのが図７，８である．これは，SMS-UDP の１プロセス時の処理時間を１とした時の性能向上比である．図７は(a) (b)は繰り返し数が異なる．繰り返し数，物体数が増加すると，計算量が通信量に比して増加するため，各種方式の差はなくなる．PVM はマスタ・スレーブ通信型と分散通信型の２種を計測したところ，前者に比べ SMS-UDP の性能が高かった．図９は各方式におけるプロセッサ 0 の UDP パケットの状況を調べたものであるが，PVM のマスタ・スレーブ通信型ではユーザレベルでメッセージ数を少なくするようにプログラムしているにもかかわらず，システムレベルで多くの小さいメッセージに分割されており，さらに，SMS-UDP に比べ通信時期が一時期に集中することにより，性能が落ちていると考えられる．MPICH の性能が SMS-UDP に比べよくないのは，TCP を使用していることや，MPI_Allgather などの関数を使用しているためかもしれない．

図８(a)(b)にはデータ割付方式の違いがあり，(a)はバリアのみ，(b)はロックも使用している．(b)はプログラムの構成上 MPICH が他と違い１プロセス多く使っているので，性能がよいが，ほぼ SMS-UDP と同程度である．

7.3 ソフトウエア共有メモリシステム

筆者らがバイナリライセンスを保有する FreeBSD 版 TreadMarks と，環境１において SMSUDP 版と比較したのが図 10,11 である．扱った２種の問題では，TreadMarks と同等以上の性能が得られていることがわかる．これは，８プロセス使用時で TreadMarks の初期プロセス起動が SMS の 5〜6 倍も時間がかかっているためであるとわかった．したがって，起動時間に比して計算処理時間が長い処理であれば，TreadMarks の性能は相対に良くなると考えられる．また図９からわかるように，TreadMarks にくらべ，メッセージ数，量とも SMS は多くなっているが，これはコンシステンシモデルの差というより，ソフトウエア実装上の問題が影響していると考えられる．

8. おわりに

この種のソフトウエアの性能には CPU・ネットワーク性能，メモリ・キャッシュ量，OS の種類やバージョンなどが影響するため，他システムとの比較がむずかしい．さらに，更新データの構成法，通信方式などのソフトウエア実装の影響も大きく，純粋なメモリコンシステンシモデルの違いによる性能への影響を抽出するのは，かなり難しい．実装の最適化とコンシステンシの評価が今後の課題である．

文　献

[1] 緑川博子，飯塚肇，“ユーザレベル・ソフトウエア分散

共有メモリ SMS の設計と実装，” 情処論文誌，HPC，Vol.42，No.SIG9(HPC3)，pp.170－190, Aug. 2001.

[2] B.N. Bershad and Matthew J. Zekauskas, “Midway: Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors,” Technical Report CMU-CS-91-170, Carnegie Mellon University, Sep. 1991.

[3] B. Cater, J.K. Bennett, and W. Zwaenepoel, “Implementation and Performance of Munin,” Proc. of the 13th Symposium on Operating Systems Principles, pp.152-164, Oct. 1991.

[4] P. Keleher, A.L. Cox, and W. Zwaenepoel, ：Lazy Consistency for software Distributed Shared Memory, Proc. of the 19th Annual Symposium on Computer Architecture, pp.13-21, May 1992.

[5] P. Keleher, A.L. Cox, S.Dwarkadas, and W. Zwaenepoel, ：TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, Proc. of the Winter USENIX Conference, pp.115-132, Jan. 94.

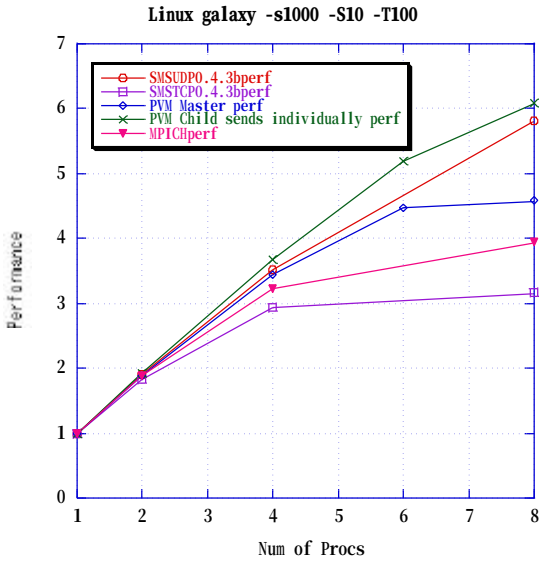
表３　基本関数性能

(a)　4SMPs　(4CPUs, pid１)

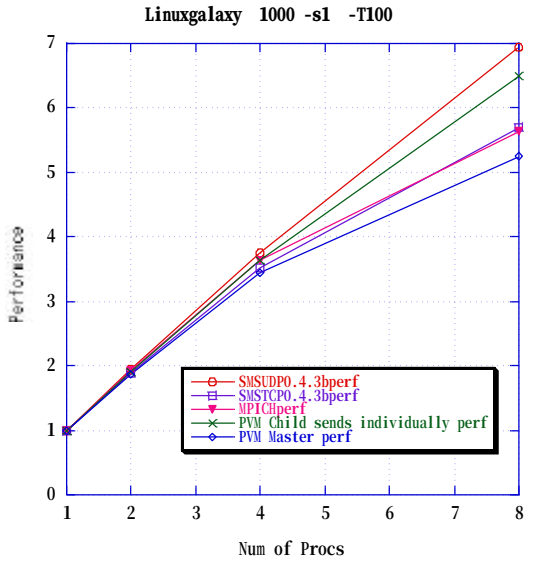
関数名(環境 3)	UDP 版	TCP 版
startup	0.076(sec)	0.205(sec)
alloc	0.395(msec)	0.648(msec)
barrier	0.923(msec)	1.340(msec)
lock/unlock	1.474(msec)	2.0932(msec)

(b)　8PCs　(8CPUs, pid４)

関数名（環境 2）	UDP 版	TCP 版
startup	0.143(sec)	1.537(sec)
alloc	0.812(msec)	3.564(msec)
barrier	1.794(msec)	11.718(msec)
lock/unlock	8.733(msec)	24.318(msec)

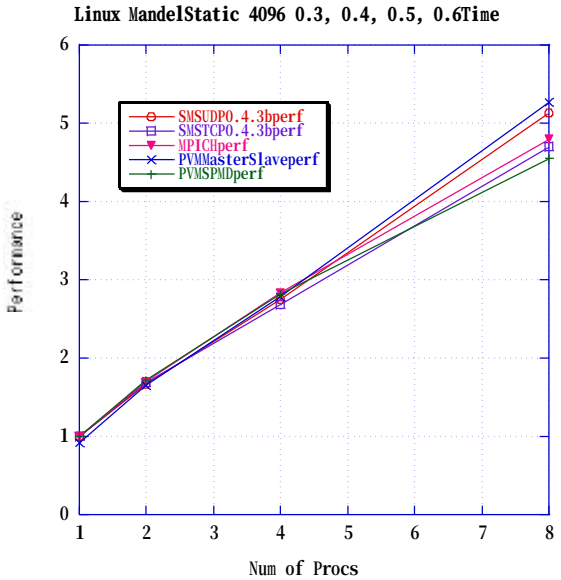


(a)　物体数 1000 繰り返し 10 回（環境２）

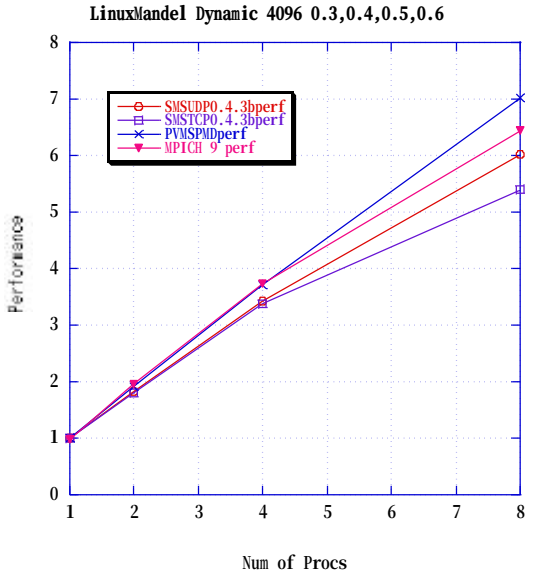


(b)　物体数 1000 繰り返し 100 回（環境２）

図７　多体問題　メッセージパッシングとの比較

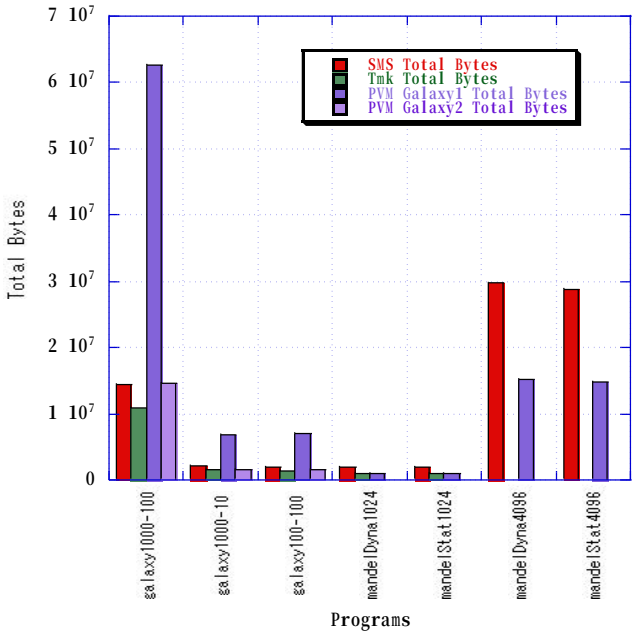


(a)　静的割付方式（環境２）

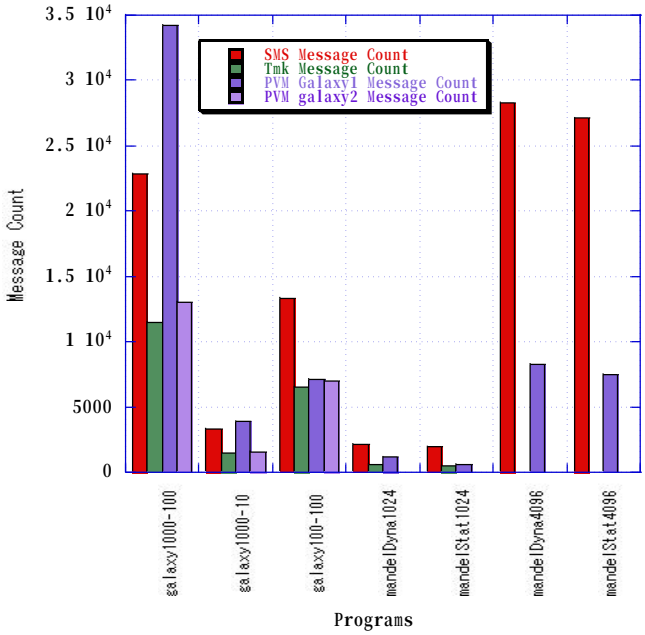


(b)　動的割付方式,64 ブロック（環境２）

図８　マンデルブロー集合計算　メッセージパッシングとの比較

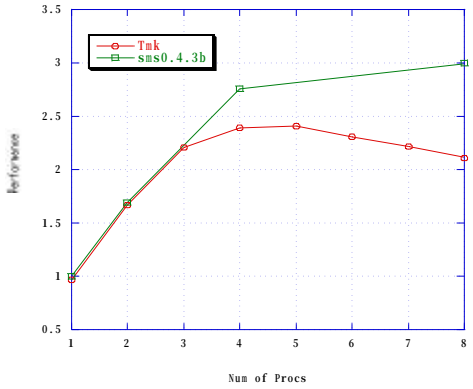


（a）メッセージ総バイト量

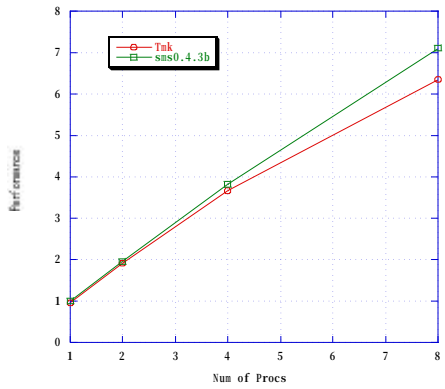


（b）メッセージ数

図９　各プログラム実行時のプロセス０におけるメッセージ比較

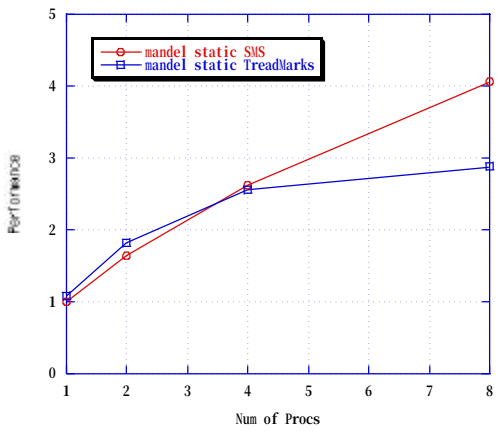


（a）　物体数 100 繰り返し 100 回（環境１）

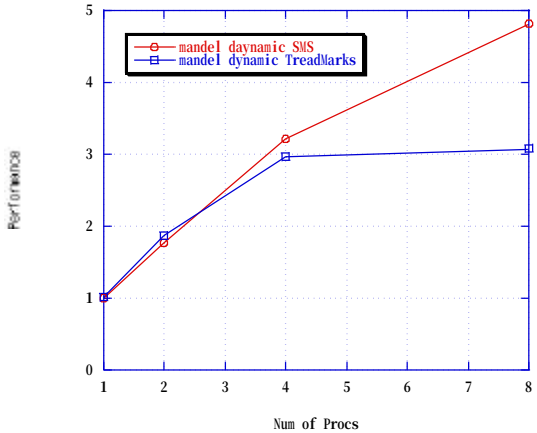


（b）物体数 1000　繰り返し 10 回（環境１）

図１０　多体問題　TreadMarks との比較



（a）静的領域割り当て並列処理 1024X1024（環境１）



(b) 動的領域割り当て並列処理　1024X1024,64 ブロック（環境 1)

図１１　マンデルブロー集合計算　TreadMarks との比較