ハイパフォーマンス
コンピューティング 62－4
（1996．8．28）

並列アルゴリズムにおけるCollective通信の性能比較

田中良夫† 久保田和人†
佐藤三久† 関口智嗣††

Broadcastやreductionなどのcollective通信はデータ並列プログラムの基本的な構成要素である．collective通信の性能を正確に知ることは，並列プログラムの性能予測など様々な点において重要である．我々はいくつかの並列計算機上でcollective通信の性能を測定した．性能を測定するにあたっては測定条件や測定方法を慎重に考慮する必要がある．本稿では測定方法およびその結果に関して報告する．本測定で得られた結果は，並列プログラムの性能予測，改善に用いることができる．

Comparison of Collective Communication Performance on Parallel Algorithms

YOSHIO TANAKA ,† KAZUTO KUBOTA ,† MITSUHISA SATO †
and SATOSHI SEKIGUCHI††

Collective communications such as *broadcast* and *reduction* are frequently used in data parallel programs. It is important to know the performance of such primitive communications to estimate the performance of parallel applications running on parallel systems. We measured the performance of the collective communications on multi-processor systems. In this paper, we present experimental results for collective communication performance. Results of our experiments are able to apply to performance estimation and improvement.

1. はじめに

Broadcast や Reduction などの全てのプロセッサ間で通信を行なう collective 通信はデータ並列プログラムにおいて基本並列演算として用いられる通信方法である．本稿の目的はさまざまな並列計算機の collective 通信の基礎的なデータを明らかにすることである．このようなデータは並列計算機を特徴づけるデータを与える．それらのデータによりその並列計算機のデータ並列プログラムの性能が予測でき，計算機とプログラムの適合性をある程度事前に知ることができる．我々は分散メモリアーキテクチャの並列計算機である Thinking Machines 社の **CM-5**[1),2)]，Intel 社の **Paragon**[3)] および Sun Microsystems 社のワークステーションをネットワークで繋いで構成したワークステーションクラスタ **(WSC)** 上でこれらの collective 通信の性能測定を行なった．また，いくつかの基本演算に関しては共有メモリアーキテクチャの並列計算機である Sun Microsystems 社の **SPARCcenter 2000(SC2000)** および Cray Research Superservers 社の **Cray Superserver 6400(CS6400)** 上でも性能測定を行い，分散メモリと共有メモリの並列計算機の比較を行なった．本実験で使用した計算機の仕様を表1および表2に示す．CM-5 はプロセッサ間の通信をサポートするネットワークとしてデータネットワーク **(DN)** とコントロールネットワーク **(CN)** の2つの内部ネットワークを持っている．DN は point-to-point の通信に用いられる．CN は broadcast やバリア同期などのグローバル通信の際に用いられる．WSC は 100Base-T と Myrinet のどちらかにより接続される．100Base-T で接続した場合 (WSC/100Base-T) は，9台の WS が HUB を介して接続され1つのクラスタを形成し，4つのクラスタが ether switch を介して接続されて合計 36 台のクラスタを形成する (図1)．Myrinet で接続した場合 (WSC/Myrinet) は，4から5台の WS がクロスバスイッチで接続されて1つのクラスタを形成し，それらのクラスタがさらにクロスバスイッチで接続されて合計 36 台のクラスタを形成する (図2)．WSC/100Base-T では **PVM**[6)] でプログラミングを行ない (WSC/PVM)，WSC/Myrinet では **PM**[4)] を用いてプログラミングを行ない (WSC/PM) データをとった．PM は WSC/Myrinet 上の MPC++ 実行環境のための通信ライブラリである．メッセージの配送の保証，メッセージ順序の保存，ネットワークコンテキストスイッチなどの機能を特徴とし，低レイ

† 新情報処理開発機構
Real World Computing Partnership
†† 電子技術総合研究所
Electrotechnical Laboratory

表 1　分散メモリ並列計算機の仕様

マシン	CM-5	Paragon	WSC/100Base-T	WSC/Myrinet
プロセッサ	SPARC	i860 XP	SuperSPARC	SuperSPARC
周波数	32MHz	50MHz	75MHz	75MHz
バンド幅	40MByte/sec(最大)	200MByte/sec	10MByte/sec	80MByte/sec
ネットワーク	木構造 (Fat Tree)	2 次元メッシュ	本文参照	本文参照
ノード数/CPU 数	32	64	36	36

表 2　共有メモリ並列計算機の仕様

マシン	SC2000	CS6400
プロセッサ	SuperSPARC TI	SuperSPARC TI
周波数	40MHz	60MHz
1 次キャッシュ	20(I)/16(D)KByte	20(I)/16(D)KByte
2 次キャッシュ	1MByte	2MByte
XDBus Freq.	40MHz	55MHz
バンド幅	640MByte/sec	1760MByte/sec
CPU 数	16	32

テンシ，高スループットを実現している．　次節以

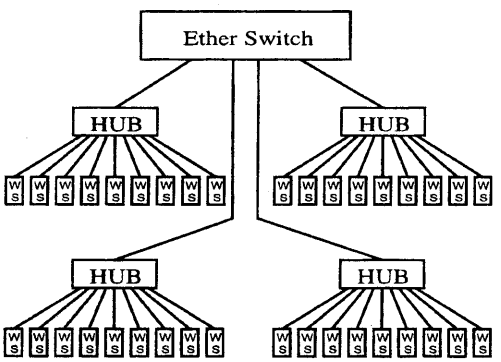

図 1　WSC/100Base-T の構成図

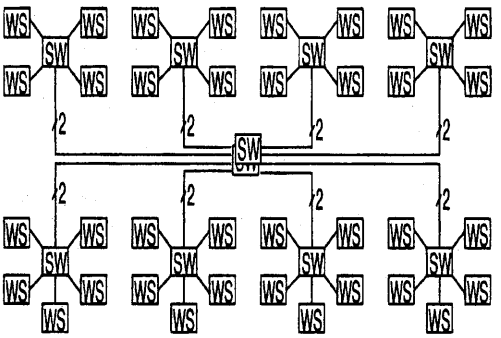

図 2　WSC/Myrinet の構成図

降では測定結果を報告する．これらの性能を測定する場合には，何をもって通信時間とするかなど測定方法を慎重に検討する必要があり，測定結果の他に測定方法に関しても述べる．測定した collective 通信は，**バリア同期**, **broadcast**, **reduction**, **scan**, **shift** および **complete exchange** である．バリア同期は全ノードが同期をとって，実行を開始するという操作である．broadcast はある 1 つのノードから全てのノードにメッセージが送られ，送り手以外のノードが送られたデータを読みとるという操作である．reduction は全ノードが保持している値すべてを用いて加算や乗算などの演算をほどこし，その結果を全ノードに返すという操作である．scan は reduction に似ているが，各ノードは自分より番号の小さなノードが保持している値のみを用いた演算結果を格納する．基本的には scan は reduction と同じアルゴリズムで実装することができ，実験結果も reduction とほぼ同様であったため，本稿では scan の結果は記述していない．shift は「ノード i はノード $i-1$ からのデータの受信とノード $i+1$ へのデータの送信を行なう」という操作を全ノードが行なう操作である．complete exchange は全対全の通信を行なう操作であり，行列計算などに良く用いられる．

2. 分散メモリ並列計算機での実験

時間の測定は CM-5 や Paragon では組み込みのタイマ関数を用い，WSC ではハードウェアタイマを直接読みとり，共有メモリ計算機では *gettimeofday()* を用いて行なった．

2.1 バリア同期 (図 3, 図 4)

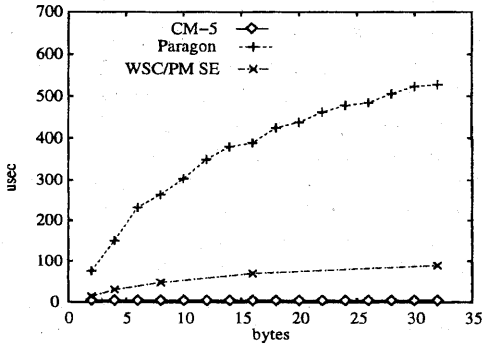

図 3　バリア同期におけるノード数と時間 (CM-5,Paragon,WSC/PM)

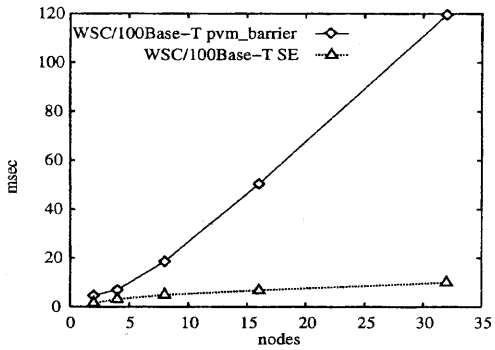

図4　バリア同期におけるノード数と時間 (WSC/PVM)

バリア同期を行う組み込みの関数を用いてノード数とバリア同期にかかる時間との関係を調べた．CM-5 では *CMMD_sync_with_nodes()*, Paragon では *gsync()*, WSC/PVM では *pvm_barrier()* を用い，WSC/PM では 1 対 1 通信を shuffle exchange(SE) アルゴリズムの方法で組み合わせてバリア同期を行った (shuffle exchange アルゴリズムの詳細は後述)．CM-5 はコントロールネットワークを使って (ハードウェアで) バリア同期をとるため，ノード数に依存せずに 5μsec 程度でバリア同期を行うことができる．WSC/PM では Paragon より高速にバリア同期をとれることが分かる．

2.2 Broadcast

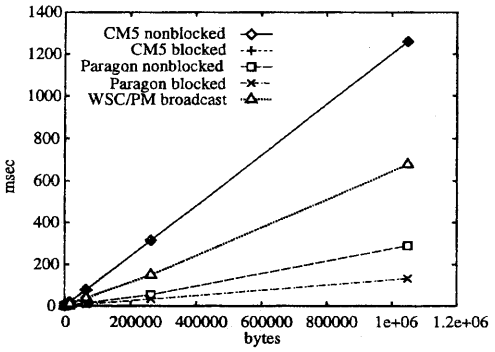

図5　Broadcast におけるデータ量と通信時間 (CM-5,Paragon,WSC/PM : 32 ノード)

Broadcast を行う組み込みの関数を用いてデータ量と通信時間，あるいはノード数と通信時間の関係を調べた．CM-5 では *CMMD_bc_to_nodes()*, Paragon では *csend()*(送信先に- 1 を指定すると broadcast になる), WSC/PVM では *pvm_bcast()* を用い，WSC/PM では 1 対 1 通信を shuffle exchange アルゴリズムにより組み合わせて broadcast を行なった．broadcast にかかる時間は，あるノードがデータを全ノードに送った時に始まり，全ノードがデータを受けとった時点までとする．CM-5 と Paragon では以下の 2 つの方法で時間を計測した．

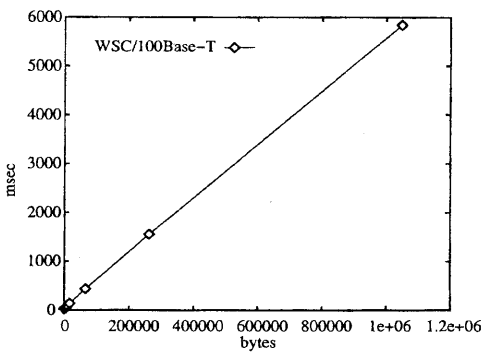

図6　Broadcast におけるデータ量と通信時間 (WSC/PVM : 32 ノード)

(1)　0 番のノードはデータ送信を繰り返し，他のノードは受信を繰り返す．

(2)　1 度のデータ送受信のたびに同期をとる．

メッセージ送信は **blocked** 型と **non-blocked** 型の 2 つの型に分類できる．blocked 型は受信側が受信を完了するまで次の処理に進まないが，non-blocked 型は受信側の受信を待たずに次の処理に進んでしまう．broadcast が non-blocked 型の場合，方法 1 では 0 番のノードは他のノードがデータの受け取りを完了する前に次々にデータの発送を行うため，計測した時間は実際の broadcast にかかる時間よりも短くなる可能性がある．WSC では上記の問題を考慮して 0 番ノードと 1 番のノードを交互に送信元として broadcast を繰り返し，1 回あたりの時間を測定した．

2.2.1 データ量 v.s. 通信時間 (図 5, 図 6)

ノード数を 32 に固定し，データ量を変化させてデータ量と通信時間の関係を調べた．CM-5 の broadcast は block 型であるため，方法 1 でも方法 2 でも結果が同じとなっている．

2.2.2 ノード数 v.s. 通信時間 (図 7, 図 8)

データ量 8Kbyte から 1Mbyte まで何通りかに固定し，ノード数を変化させてノード数と通信時間の関係を調べた．CM-5 の broadcast はコントロールネットワークを使うためノード数に依存せずに通信時間はほぼ一定であった．Paragon と WSC ではノード数によって通信時間が変化している．Paragon は 2 次元メッシュであるので，あるノード数を越えると通信ステップが増加するため，グラフが階段状になっている．

2.3 Reduction(図 12, 図 13, 図 14)

以下の 3 種類のアルゴリズムを用いて double 型の reduction を実装し，ノード数を変化させてノード数と通信時間の関係を調べた．また WSC では，

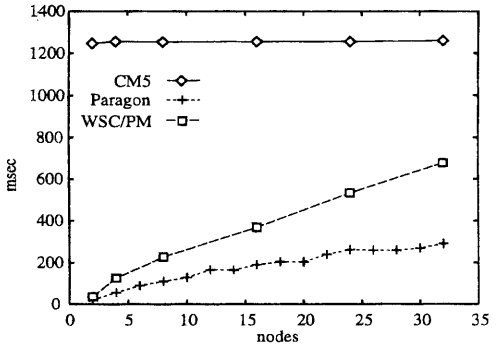

図7 Broadcast におけるノード数と通信時間 (Paragon,WSC/PM:1Mbyte)

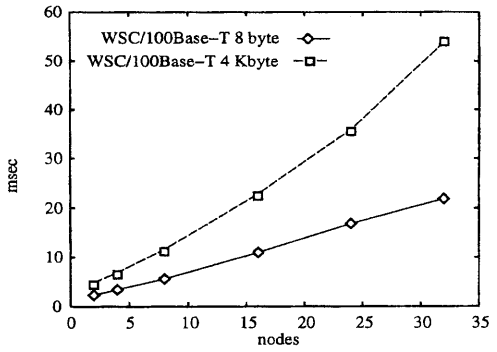

図8 Broadcast におけるノード数と通信時間 (WSC, 8Byte および4Kbyte)

pvm_reduce() 関数を用いて reduction を行なった結果も加えた.

(1) Master/slave アルゴリズム

あるノードが他の全てのノードから値を受けとり，その値に対して演算をほどこした結果を他の全てのノードに送る．ノード数 Nに対して $N-1$ 回の繰り返し処理が行なわれる．master/slave アルゴリズムで加算の reduction を行う様子を図 9に示す.

(2) Cascade アルゴリズム

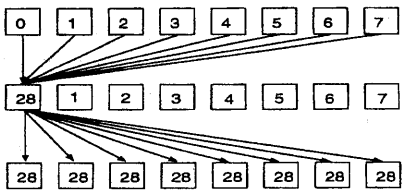

図9 master/slave アルゴリズム

Cascade アルゴリズムで加算の reduction を行う様子を図 10 に示す．図の太線で表した構造は，通常の 2 進木による和の計算グラフであり，Cascade アルゴリズムは太線の構造を n 個巡回的に重ね合わせたものである．ノード数 Nに対して，$\log_2 N$回の繰り返し処理が行なわれる.

(3) Shuffle exchange(SE) アルゴリズム

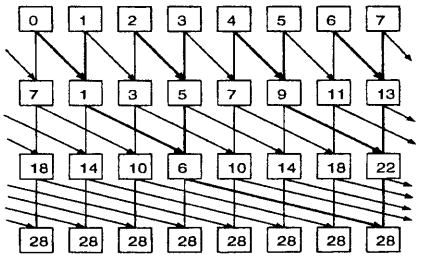

図10 Cascade アルゴリズム

SE アルゴリズムは Cascade アルゴリズムとほとんど同じであるが，各ステップにおける通信相手が若干異なる．ノード数 Nに対して，$\log_2 N$回の繰り返し処理が行なわれる．SE アルゴリズムで加算の reduction を行なう様子を図 11に示す． cascade アルゴリズム

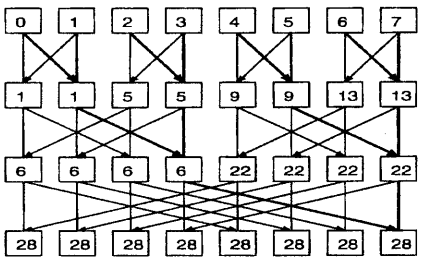

図11 Shuffle Exchange アルゴリズム

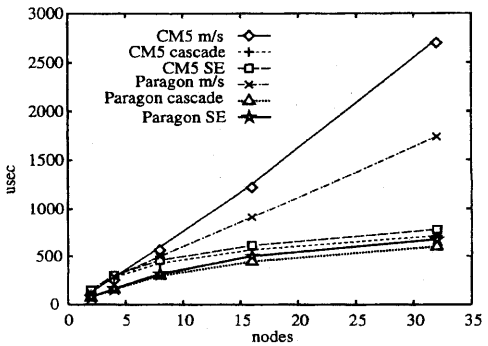

図12 Reduction におけるノード数と通信時間 (CM-5,Paragon)

と SE アルゴリズムはほぼ同じ性能である．得られた

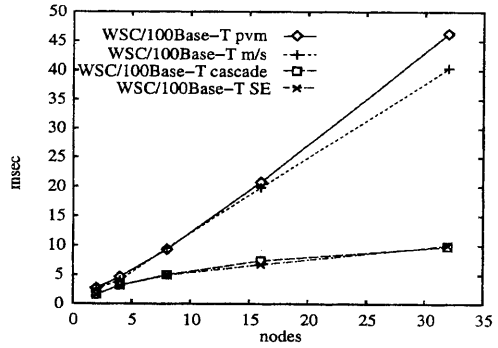

図13 Reduction におけるノード数と通信時間 (WSC/PVM)

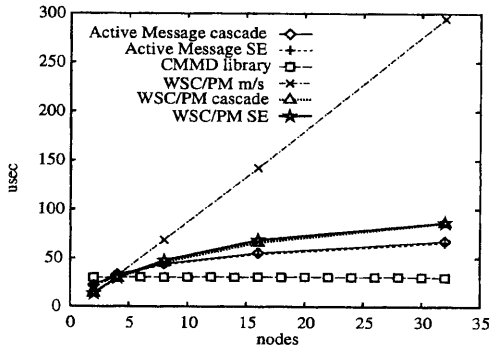

図14 Reduction におけるノード数と通信時間 (AcitveMessage,CMMD,WSC/PM)

結果はアルゴリズムのオーダー通りの結果となった. CM-5 においては CMMD ライブラリを用いる場合はコントロールネットワークを使用するため, ノード数に依存せずにほぼ一定の値を示し, Active Message の機能を用いれば CMMD ライブラリの倍程度の時間で終えることが分かる.

2.4 Shift(図15, 図16, 図17, 図18)

データ量と通信時間の関係と, ノード数と通信時間の関係を調べた. ノード数を変化させた場合, CM-5, Paragon や WSC/Myrinet では差が見られないが WSC/100Base-T はノード数が増加するにつれて時間も増加する.

2.5 Complete Exchange(図20, 図21, 図22, 図23)

linear, pairwise および SE の 3 種類のアルゴリズムについて, ノード数を 32 に固定し, データ量と通信時間の関係を調べた. 具体的には転置行列を作成するプログラムを用いた. linear アルゴリズムでは, ノード数 N に対して N ステップの繰り返し処理が行なわれる. i 回目のステップでは i 番のノードが他のノードからデータを受けとる. pairwise アルゴリズムを図19に示す. どちらのアルゴリズムもステップ数はノード数

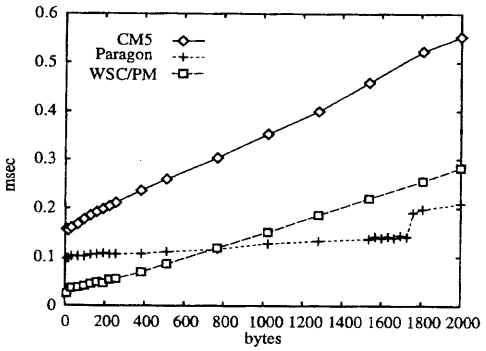

図15 Shift におけるデータ量と通信時間 (CM-5,Paragon,WSC/PM : 32 ノード)

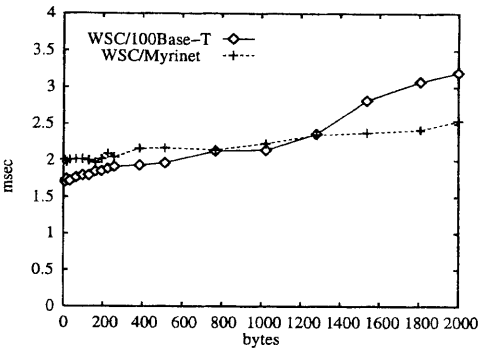

図16 Shift おけるデータ量と通信時間 (WSC/PVM : 32 ノード)

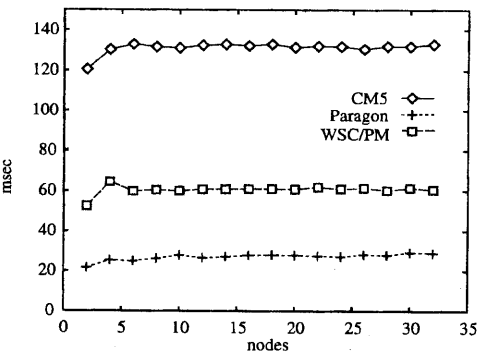

図17 Shift おけるノード数と通信時間 (CM5,Paragon,WSC/PM: 500Kbyte)

N に比例するが, linear アルゴリズムでは各ステップごとにあるノードが $N-1$ 回のメッセージ送受信を行うのに対し, pairwise アルゴリズムでは各ステップごとに全ノードが 1 度のメッセージ送受信を行うだけで済む. また, SE アルゴリズムは基本的には reduction の SE アルゴリズムと同じであるが, 各ステップにお

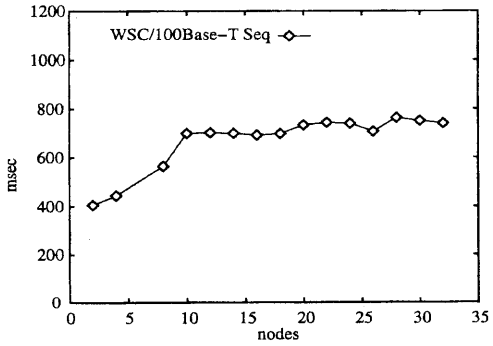

図 18　Shift おけるノード数と通信時間 (WSC/PVM: 500Kbyte)

いて送受信するデータは linear および pairwise アルゴリズムよりも多く，さらに送受信用のバッファに詰め込んだり分けて取り出す必要があるというオーバーヘッドがある．アルゴリズムのオーダー的には SE が

```
for (i = 1; i < N; i++) {
  node = me ^ i;
  if (me < node) {
    receive(node);
    send(node);
  }
  else {
    send(node);
    receive(node);
  }
}
```

図 19　pairwise アルゴリズム

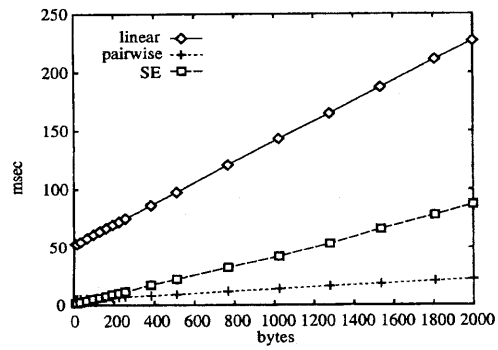

図 20　Complete exchange におけるデータ量と通信時間 (CM-5：32 ノード)

最も優れているが，データ量が多くなると前述のようなデータ処理に関するオーバーヘッドがネックとなる

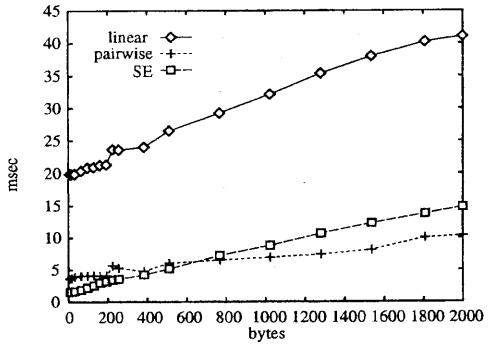

図 21　Complete exchange におけるデータ量と通信時間 (Paragon：32 ノード)

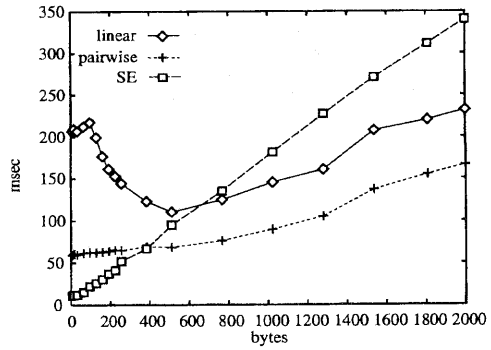

図 22　Complete exchange におけるデータ量と通信時間 (WSC/PVM：32 ノード)

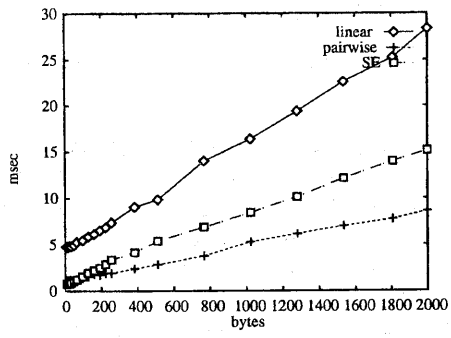

図 23　Complete exchange におけるデータ量と通信時間 (WSC/PM：32 ノード)

ため，pairwise アルゴリズムよりもかえって時間がかかってしまうことが分かる．また，WSC では CM-5 や Paragon とは異なる傾向が得られた．データ量が多くなると SE アルゴリズムが最も時間がかかる．また，linear アルゴリズムではデータ量が少ない時にかえって時間がかかるという傾向が見られる．

3. 共有メモリ並列計算機での実験

スレッドとは，1つのプロセスの中でメモリを共有しながら異なったコンテキストを持ち，並列にCPUを割り当てられる実行単位である．Solaris2では，カーネルがLight Weight Process(LWP)を実現し，ライブラリはLWPを用いてスレッドをユーザに提供している．マルチプロセッサ機では，LWPに対してCPUが割り当てられる．LWPとスレッドによって物理的なCPUの数に依存せずに並列プログラムが書けるようになっている．すなわち，1つのプロセスが，メモリを共有する多数のスレッドで構成されるようにプログラムすれば，ライブラリが実行可能なスレッドをLWPにマップし，Solarisのカーネルが LWPを物理CPUにマップする．メモリアクセスに関する実行順序はスレッド間で保証されない．スレッドは1つのプロセスの中で並行に走るので，自然にメモリを共有できるが，各スレッドが同一の変数を読み書きする場合は同期をとってアクセスを直列化しなければならない．そのための同期プリミティブとして，Solarisは *mutex_lock,condition_variable* などを提供している．本実験では，スレッドはすべてバウンドスレッドとして生成し，常にLWPと結合状態にあるようにしている．

3.1 バリア同期と Reduction(図 26)

SC2000 および CS6400 では mutex 変数と condition 変数を組み合わせた方法 (方法1，図24) と，mutex 変数と counter を組み合わせた方法 (方法2，図25) の2通りの方法でバリア同期を行なった．

```
typedef struct {
  mutex_t br_lock; /* lock for this structure */
  cond_t br_cond;  /* condition variable */
  int br_count;    /* counter */
  int br_n_thread; /* the number of thread */
} barrier_t;

barrier(barrier_t *b)
{
  mutex_lock(&b->br_lock);
  b->br_count++;
  if(b->br_count < b->br_n_thread)
    cond_wait(&b->br_cond,&b->br_lock); /* wait */
  else {                /* all process has come */
    b->br_count = 0;    /* clear counter */
    /* release all waiting threads */
    cond_broadcast(&b->br_cond);
  }
  mutex_unlock(&b->br_lock);
}
```

図 24　Mutex + Condition によるバリア同期

また，Reduction は方法2の counter を処理する部分に演算を追加するという方法で行えるため，Reduction とバリア同期は同じ結果になる．SC2000 や

```
typedef struct {
  mutex_t br_lock; /* lock for this structure */
  int br_incount;  /* in counter */
  int br_outcount; /* out counter */
  int br_n_thread; /* the number of thread */
} barrier_t;

barrier(barrier_t *b)
{
  while(b->br_outcount != 0)
    /* spin lock for enter */;
  mutex_lock(&b->br_lock);
  b->br_incount++;
  mutex_unlock(&b->br_lock);
  while(b->br_incount != b->br_n_thread)
    /* spin lock */;
  mutex_lock(&b->br_lock);
  b->br_outcount++;
  if(b->br_outcount == b->br_n_thread){
    b->br_incount = 0; /* reset */
    b->br_outcount = 0; /* reset, open door */
  }
  mutex_unlock(&b->br_lock);
}
```

図 25　Mutex + Counter によるバリア同期

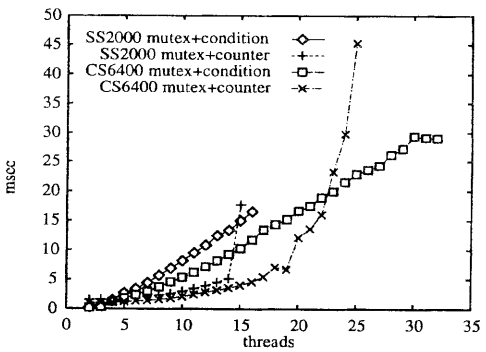

図 26　バリア同期におけるスレッド数と時間 (SC2000,CS6400)

CS6400では，方法1ではノード数に比例して時間が増える．方法2は方法1に比べると若干速いが，生成したスレッド数が実際のCPUの数に近付くと急激に時間が増加する．これはスレッドがスピンロックによって待ち，リソースを解放しないためである．

3.2 Matrix Transpose(図 27)

Complete exchange に相当するものとして，行列の転置を行うプログラムを作成し，実験を行なった．SC2000 と CS6400 ではスレッド数を16あるいは32に固定し，データ量と実行時間の関係を調べた．cache を考慮して転置行列の作成は各 PE の cache の内容がきちんとフラッシュされるように考慮して実験を行った．complete exchange というオペレーションは各PE

が持つデータを全体全で交換するものである．しかし共有メモリで転置行列を作成しても，必ずしもその意味での complete exchange にはならない．なぜならば，普通の共有メモリではグローバルメモリのアクセスになるからである．しかし，オペレーションとして転置行列の作成は complete exchange とほぼ等価である．実験の結果，共有メモリでの転置行列の作成は

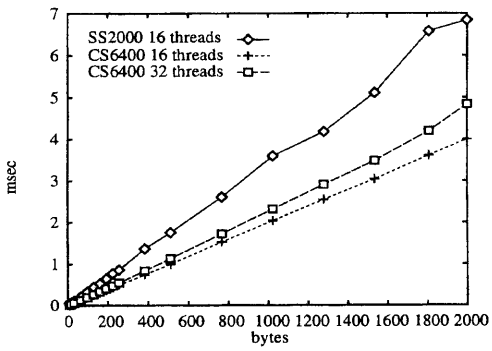

図 27　Matrix Transpose におけるデータ量と実行時間 (SC2000,CS6400)

分散メモリでの結果よりも倍以上速いことが分かった．しかし，先にも述べたように分散メモリと共有メモリの比較を行う場合にこの方法で良いのかどうかは慎重に検討する必要がある．

4. おわりに

今回は分散メモリ並列計算機として CM-5, Paragon, WSC を，共有メモリ並列計算機として SC2000 と CS6400 を用いて，collective 通信の性能を比較した．PM を使えば WSC でも CM-5 などの並列計算機とほぼ同じあるいはそれ以上の性能が得られることが分かった．また，バリア同期や reduction などには共有メモリの方が分散メモリよりも時間がかかってしまう．本稿では分散メモリと共有メモリの比較を行なっているが，例えば共有メモリの場合バリア同期には分散メモリとほぼ同じ程度の時間がかかっているのに，転置行列の作成はかなり短い時間で終ってしまう．このような点を考えても，分散メモリと共有メモリの比較にはもっと比較方法などを慎重に検討する必要がある．今回得られた結果をもとに具体的なアプリケーションプログラムの性能予測などが可能となる．collective 通信を行うようなプログラミングモデルを用いた場合，従来のように「分散メモリ」と「共有メモリ」を独立させた性能測定はプログラムの性能予測を行うだけでは不十分であることがわかった．すなわち，これらの collective 通信が個々の使用システムではいかなる性能で実現されているかがユーザの視点において重要である．我々もすでにシステムのアーキテクチャ，メモリモデル，プログラムスタイル等に依存しない統一的な性能評価指標の提案を行ってきた[7]．これを collective 通信の性能指標としても適応することが今後の目標のひとつである．なお，本研究で得られた実験データは以下の URL にて公開予定である．

`http://www.trc.rwcp.or.jp/mpperf/`
`http://phase.etl.go.jp/`

謝　辞

本研究に対して議論に参加頂き，多くのアドバイスを頂きました NPB グループの皆様に感謝致します．また，実験に際して WSC を使用する機会を与えて頂きました新情報処理開発機構超並列ソフトウェア研究室の皆様に感謝致します．なお，本研究の一部は工業技術院国際特定共同研究「ハイパフォーマンスコンピューティングシステム性能評価モデルの研究」に基づくものである．

参考文献

1) "CMMD Reference Manual", Thinking Machines Corporation, (1995).
2) R. Ponnusamy, A. Choudhary and G. Fox: "Communication Overhead on CM5: An Experimental Performance Evaluation", Proceedings of Frontiers '92, pp. 108–115 (1992).
3) "Paragon System Administrator's Guide", Intel Corporation, (1995).
4) 手塚 宏史, 堀 敦史, 石川 裕: "ワークステーションクラスタ用通信ライブラリ PM の設計と実装", 情報処理学会並列処理シンポジウム JSPP'96, pp. 44–48 (1996).
5) N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su.: "Myrinet – A Gigabit-per-Second Local-Area Network", IEEE MICRO, Vol. 15, No. 1, pp. 29–36 (1996).
6) A. Geist, A. Geguelin, J. Dongarra, W. Jiang, R. Mancheck and V. Sunderam: "PVM: Paralle Virtual Machine", The MIT Press, (1994).
7) S., Sekiguchi and M., Sato: "A Performance Model and Metrics for Fine Grain Parallel Computing Systems", Proceedings of 1996 IEEE Second International Conference on Algorithms and Architectures for Parallel Processing, (1996).