地球シミュレータのMPI性能評価

上原 均[†] 田村正典[††]
板倉憲一[†] 横川三津夫[†††,☆]

地球変動研究のための超大規模科学技術計算向けプラットフォームとして，地球シミュレータが開発された．分散メモリ型並列計算機である地球シミュレータ上で並列プログラムを効率的に実行するには，プログラム内部での通信処理の効率化が必要で，そのためには事前の通信性能評価が不可欠である．地球シミュレータでは通信用ライブラリとしてMPIが提供されるので，MPIの通信性能を詳細かつ多角的に計測するMBLを用いて，地球シミュレータのMPIの性能計測と評価を行った．その結果，ping-pong通信の最大スループットはノード内で14.8 GB/s，ノード間で11.8 GB/sであることを確認した．主要MPI関数のレイテンシは，MPI_Send関数が5.58マイクロ秒，MPI_Put関数が6.36マイクロ秒であった．また512ノード実行で各ノードに1MPIプロセスを生成した場合，MPI_Barrier関数の処理時間は3.25マイクロ秒，MPI_Win_fence関数の処理時間は223.75マイクロ秒であった．これらの計測結果から，地球シミュレータのMPIが良好な性能を持つことが確認された．

MPI Performance Evaluation on the Earth Simulator

Hitoshi Uehara,[†] Masanori Tamura,[††] Ken'ichi Itakura[†]
and Mitsuo Yokokawa[†††,☆]

The Earth Simulator is an ultra high-speed supercomputer which was developed for global environment change simulations. For achieving high performance computing on large scale distributed memory parallel computers such as the Earth Simulator, an optimization of communication processings in user applications is required, and the optimization needs an evaluation for performance of communication methods. On the Earth Simulator, Message Passing Interface (MPI) is supported as the communication method. We have evaluated performance of the MPI-1/MPI-2 functions on the Earth Simulator in detail using MBL which was developed for the measurements of MPI performance on various parallel computers. The results show that the maximum throughputs of ping-pong communication using MPI_Send are 14.8 GB/s within a node and 11.8 GB/s between two nodes. Latencies of MPI_Send and MPI_Put are 5.58 microseconds and 6.36 microseconds, respectively. On the condition that run one MPI-process on one node and use 512 nodes, latencies of MPI_Barrier and MPI_Win_fence are 3.25 microseconds and 223.75 microseconds, respectively. We found out that the MPI on the Earth Simulator has excellent performance.

1. はじめに

地球変動研究のための超大規模科学技術計算向けプラットフォームとして，地球シミュレータが開発された[1)~3)]．地球シミュレータは，640台の計算ノードを単段クロスバスイッチで接続した分散メモリ型並列計算機である．2002年5月現在，LINPACKを用いたベンチマークで35.86 Tflops（640ノード使用時）を達成した．これはピーク比にして87.5%にあたる．

地球シミュレータのような大規模分散メモリ型計算機上でプログラムを効率的に実行するには，並列プログラムの高速化，特に通信処理の最適化が重要であり，そのためには通信性能の評価が必要不可欠である．アプリケーション性能を予測するためにも，通信性能の評価は必要である．通信ライブラリとしてはMPI[4),5)]が，NECによって地球シミュレータ向けに実装されている．よって，この地球シミュレータ向けMPI実装の性能評価が必要となる．また，ユーザアプリケー

† 海洋科学技術センター地球シミュレータセンター
Japan Marine Science and Technology Center
†† 株式会社日本電気第1コンピュータソフトウェア事業部
NEC Japan First Computer Software Division
††† 日本原子力研究所
Japan Atomic Energy Research Institute
☆ 現在，産業技術総合研究所グリッド研究センター
Presently with National Institute of Advanced Industrial Science and Technology

ションの最適化等に有用なデータを得るには，単なる最高性能の評価だけではなく，ユーザ側から見た性能特性を詳細かつ多角的に評価しなければならない．

そこで，地球シミュレータ向け MPI 実装において特に重要と考えられる MPI-1 の 1 対 1 通信関数，バリア同期関数，MPI-2 RMA 関数について性能測定を行った．また，実際のシミュレーションプログラムでしばしば見られる通信パターンについて複数の手法でプログラミングした場合の性能測定も行った．MPI の性能測定には，Pallas MPI Benchmarks（PMB）[6] や Effective Bandwidth Benchmark [7] 等を用いることも可能だが，計測項目や計測方法が不十分であると考えられる．そこで，我々が MPI 関数の性能を詳細かつ多角的に測定するために開発した MPI benchmark program library（MBL）[8),9)]（付録 A.1 で概説）を用いて，地球シミュレータの一般的ユーザと同じ立場から計測した．

2. 地球シミュレータと MPI 実装の概要

2.1 地球シミュレータの概要

地球シミュレータは，気象・気候分野のシミュレーションにおいて約 5 Tflops の実効性能の達成を目指して，宇宙開発事業団，日本原子力研究所，および海洋科学技術センターによって開発された．2002 年 5 月現在，気候シミュレーション AFES で 26.58 Tflops（640 ノード使用時）を達成している．

この地球シミュレータは，**図 1** に示すように 640 台の計算ノード（以降，ノードと記す）を単段クロスバスイッチで結合した分散メモリ型並列計算機である．各ノードは，**図 2** に示すように算術演算プロセッサ（AP）8 台，容量 16 GB の主記憶装置（MS），ノード間通信を処理するリモートアクセス制御装置（RCU），外部との入出力を処理する入出力プロセッサ（IOP）から成る共有メモリ型並列計算機である．

AP はピーク性能 8 Gflops のベクトル型計算プロセッサである．各 AP と MS のバンド幅は 32 GB/s で，1 ノードでは計 256 GB/s のバンド幅を確保している．このメモリバンド幅はロード/ストアの両方で使われるため，AP によるメモリコピーの最大スループットは 16 GB/s である．RCU はクロスバスイッチと直接に接続されて，クロスバスイッチを介した送受信処理を AP と独立に処理できる．RCU と MS のバンド幅は load/store 方向ともに 16 GB/s である．クロスバスイッチの理論最大スループットは送受信方向ともに 12.3 GB/s である．

地球シミュレータの特徴の 1 つに，グローバルメモリという異なるノード上のプロセスから共有できる共有メモリがある．グローバルメモリの領域は通常の SystemV 共有メモリのそれと同じく動的に確保され，さらにその確保時には RCU にスタートポイントとサイズが登録される．その登録済み領域にのみ RCU はアクセスでき，逆に未登録領域にはアクセスできない．よって RCU が制御するノード間通信を行うには，いったんはグローバルメモリ上に通信データが配置されなければならない．そのため，MPI では内部的に転送バッファ（以下，MPI 通信バッファ）をグローバルメモリ上に確保する．グローバルメモリ領域間では DMA 転送が可能である．なお，グローバルメモリは SystemV 共有メモリの代替としても使うことができる．

グローバルメモリ領域は，**図 3** のようにプロセス仮想空間とグローバル仮想空間の両方に存在する．つまり同一の領域が 2 種類のアドレスを持つ．グローバルメモリ領域への操作は，通常のロード/ストア系命令ではプロセス仮想空間でのアドレスを指定し，グローバル仮想空間でのアドレスは MPI 関数内部等で

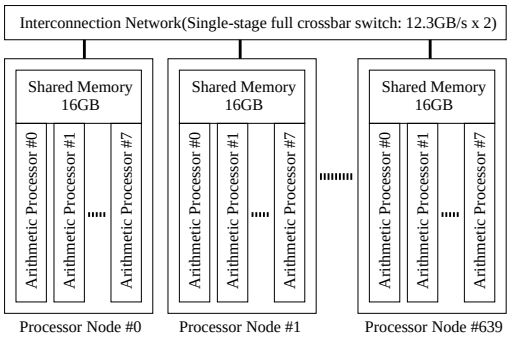

図 1 地球シミュレータの全体構成
Fig. 1 Overview of the Earth Simulator.

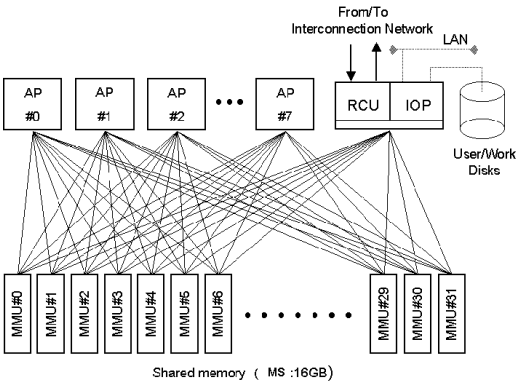

図 2 地球シミュレータの計算ノードの構成
Fig. 2 Overview of processor node.

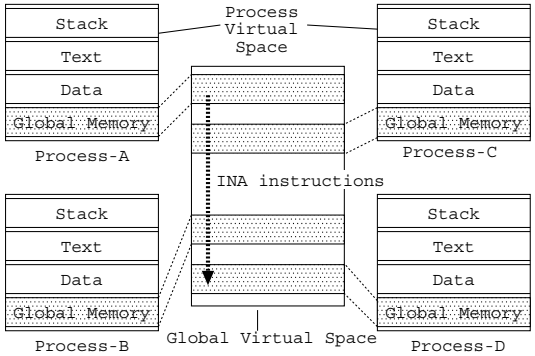

図 3 グローバル仮想空間
Fig. 3 Global virtual space.

のノード間通信処理で用いられる．なお以降では，プロセス仮想空間での Data 領域等の非グローバルメモリ領域をローカルメモリと記す．

プロセス仮想空間全体をグローバルメモリとして割り付けることは，グローバルメモリ領域の割付けコストとノードの搭載メモリ量の 2 つの観点から行われない．グローバルメモリを RCU へ登録するにはシステムコール（OS）を介する必要があり，そのコストは相当に高い☆．そのコストの高さから，スタック等の動的かつ頻繁に確保される領域を確保のたびにグローバルメモリ上に割り付けることは処理時間の増大を招くため，現実的でない．

また地球シミュレータのハードウェアおよび OS ではアドレス空間の管理は仮想方式であるが，メモリ割付けについては実メモリ割付け方式を採用している．そのため，前記のオーバヘッド問題を回避するために実行プログラム開始時点でユーザの使える仮想メモリ空間のすべて（地球シミュレータでは 4 TB）を，ノードの搭載メモリ（16 GB）に割り付けることはできない．以上から，ユーザのメモリ空間は基本的にローカルメモリに割り付けられ，グローバルメモリ領域は明示的な割付け要求に基づいて確保される．

グローバルメモリを Fortran90 から利用するには，**図 4** のように動的割付けとコンパイルオプション（-gmalloc）を併用する．いったん割り付けたらローカルメモリ上のものと同様に利用でき，最適化等を含めて，その利用に関して特別なプログラミングテクニックは必要ない．グローバルメモリ上のデータには RCU が直接アクセスできるので，通信対象となるデータの格納が主用途である．

☆ Fortran での allocate 文によるグローバルメモリへの動的割付けは，確保するメモリサイズによらず約 2.67 ミリ秒であるが，そのコストの 90%以上が RCU への登録コストである．

```
! コーディング方法
real,allocatable,dimension(:):: a
real b(10)       ! 配列 b はローカルに確保
allocate(a(10)) ! a の領域を動的に確保
a(1) = b(1)      ! 利用方法は通常どおり
```

```
% mpif90 -gmalloc t.F
```

図 4 グローバルメモリを利用するコーディングとコンパイル方法
Fig. 4 Use of global memory on the Earth Simulator.

地球シミュレータでは，ジョブ実行は排他的にスケジューリングされており，各ユーザは確保したノードを排他的に利用する．そのため，他のジョブ実行にかかわらず，性能的に安定した状態でユーザは地球シミュレータを利用できる．

2.2 地球シミュレータ上の MPI 実装の概要

地球シミュレータでは MPI-1 と MPI-2 の両方が提供される．MPI_Send 等の引数となる送信データの格納場所あるいは MPI_Recv 等の引数となる受信データの格納場所（以降，送受信バッファと記す）の位置（ローカルメモリ上，グローバルメモリ上）と，通信の種類（ノード内，ノード間）という条件の組合せから，MPI 関数内部での処理は以下の 4 つのケースに大別できる．

(1) 送受信バッファがローカルメモリにあり，かつノード内通信が行われる場合．
(2) 送受信バッファがローカルメモリにあり，かつノード間通信が行われる場合．
(3) 送受信バッファがグローバルメモリにあり，かつノード内通信が行われる場合．
(4) 送受信バッファがグローバルメモリにあり，かつノード間通信が行われる場合．

送受信バッファがローカルメモリにある場合は，送受信バッファと MPI 通信バッファの間でメモリコピーが必要だが，送受信バッファがグローバルメモリにある場合には送信側バッファから受信側バッファへ直接コピー（シングルコピー）できる．ノード間通信では，図 3 の INA instructions で示すように，送信元から送信先へのクロスバスイッチを介したデータ転送が行われる．

メモリコピー回数とデータ転送関数の和を処理数とすると**表 1** のようになる．メモリコピーはベクトル化され，さらにパイプライン処理もされるので，極言すればデータ量が大きい場合には 2 回のメモリコピー処理を 1 回の処理と見なせる．そのため，ノード内通信で送受信バッファをローカルに配置した場合（処理回数 2 回）の性能が，通信データ量が多くなるにつれて，グローバルメモリに配置した場合（処理回数 1 回）の

表 1 MPI 実装内部での処理数
Table 1 The number of data movements.

通信種別	MPI 関数で指定した送受信バッファの位置	処理数
ノード内	ローカル	2
	グローバル	1
ノード間	ローカル	3
	グローバル	1

```
if (rank == 0) then
  call MPI_Irecv(...)
  call MPI_Barrier(...)
  t1=MPI_Wtime()
  call MPI_Send(...,1,...)
  call MPI_Wait(...)
  time = (MPI_Wtime()-t1)/2
else if (rank == 1) then
  call MPI_Irecv(...)
  call MPI_Barrier(...)
  call MPI_Wait(...)
  call MPI_Send(...,0,...)
endif
```

図 5 ping-pong 通信の計測コード
Fig. 5 Ping-pong code.

性能に近づくことが予測できる．

3. 地球シミュレータでの MPI 性能計測

3.1 MPI_Send を用いた ping-pong 通信性能

ここでは，MPI_Send を用いた ping-pong 通信時の通信性能について述べる．**図 5** のように MPI プロセスランク 0，1 間で ping-pong 通信を行い，その所要時間を計測した．送受信バッファのメモリ配置（ローカルメモリ/グローバルメモリ）と，通信種別（ノード内通信/ノード間通信）という条件を組み合わせた，4 つのケースについて計測した．通信メッセージ長を 4 B から 64 MB まで変化させて各々の処理時間を 200 回計測し，その平均をとった．

図 6 にメッセージ長を平均所要時間で除算して得たスループットを示す．縦軸がスループット，横軸がメッセージ長である．64 MB 時で通信性能はほぼ飽和しており，ノード内通信の最大スループットはグローバルメモリ使用時の 14.8 GB/s，ノード間通信の最大スループットはグローバルメモリ使用時の 11.8 GB/s であった．所要時間の分散は，グローバルメモリを使用した 256 KB でのノード内通信時における 0.17e-13 が最大で，そのときの性能値とピーク性能値の比は 0.94 と 1 に近く，性能が安定していることが確認された．

グローバルメモリを用いた MPI_Send によるノード内通信は，MPI_Send 内部においてメモリコピー

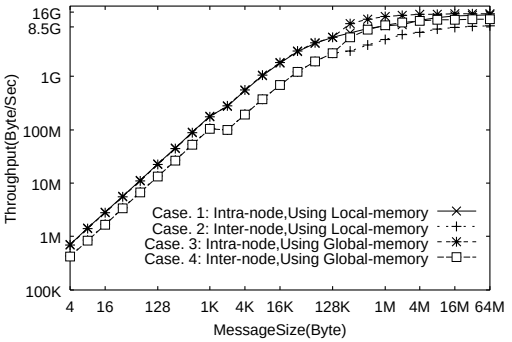

図 6 MPI_Send を用いた ping-pong 通信のスループット
Fig. 6 Throughput of ping-pong using MPI_Send.

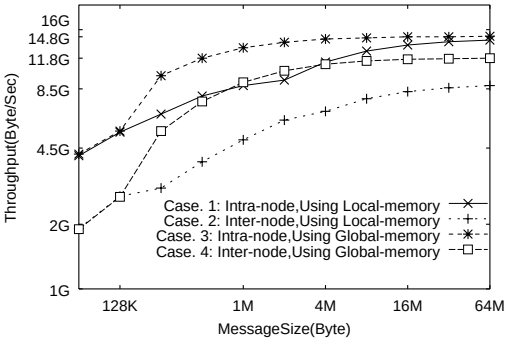

図 7 128 KB 以降のスループット
Fig. 7 Throughput of message transfer over 128 KB.

1 回で処理することができる．2.1 節で述べたようにメモリコピーの最大スループットは 16 GB/s なので，MPI_Send によるノード内通信の最大効率はメモリコピーの最大スループットに対して 92.5%である．同様に，グローバルメモリを用いた MPI_Send によるノード間通信は，MPI_Send 内部においてクロスバスイッチを挟んだメモリコピー 1 回で処理できる．クロスバスイッチの理論最大スループットは 12.3 GB/s なので，MPI_Send を用いたノード間通信の最大効率はネットワークの理論最大スループットに対して 95.9%といえる．以上から，地球シミュレータ上での MPI_Send は高効率な実装がされているといえる．

図 6 の 1 KB 付近と 128 KB 付近に性能ギャップがある．特に 128 KB 以降ではケースごとに性能が異なるため，**図 7** に拡大図を示す．地球シミュレータ向け MPI では，MPI_Send 内部において 1 KB 以下のメッセージ送信では受信側の MPI 通信バッファに通信データを直接書き込み，1 KB 超 200 KB 未満では受信リクエストが発行されてから通信データが送信される．200 KB 以上の場合は，送受信プロセス間で同期して最適化された処理を行う．この切替えは，各条件下での 1 KB 前後と 200 KB 前後での所要時間を計測

表 2 MPI_Send での通信方式切替え前後のレイテンシ
Table 2 Latencies of MPI_Send with 1 KB and 200 KB.

	ローカルメモリ使用		グローバルメモリ使用	
サイズ	ノード内	ノード間	ノード内	ノード間
1020	3.40	6.11	2.93	6.22
1024	3.40	6.21	2.94	6.23
1028	3.44	11.50	3.00	11.24
204796	15.93	40.16	15.95	39.79
204800	33.38	64.46	18.59	41.84
204804	33.48	65.61	18.66	83.26

（単位：マイクロ秒）

した結果（**表 2** 参照）からも確認できた．性能ギャップは，これらの通信方式切替えによるものである．

MPI_Send では，グローバルメモリ使用時のシングルコピーによる通信処理は，メッセージ長が 200 KB 以上でのみ行われ，200 KB 未満ではローカルメモリ使用時と同様に処理される．そのため，ローカルメモリ使用時とグローバルメモリ使用時の性能差は図 7 から分かるように，200 KB 以降で起こる．

また 2.2 節末で述べた，ローカルメモリ使用時性能のグローバルメモリ使用時性能への接近が図 7 で確認できる．ローカルメモリを用いた場合のノード内通信とノード間通信の性能差は，パイプライン処理を適用できないノード間転送によるものと思われる．

なおローカルメモリを用いたノード内通信において 2 MB 付近で性能が若干低下する原因は開発側で調査中であるが，グローバルメモリの利用がスループット向上の面で有効と考えられる．

3.2 MPI_Barrier の性能

ここでは，MPI_Barrier の性能について述べる．地球シミュレータでは，ノード内とノード間でバリア同期のメカニズムが異なる．ノード間バリア同期は Global Barrier Counter というハードウェアを用いて実装されている．一方，ノード内バリア同期は共有メモリを用いて実装されている．2 個以上のプロセスを 1 ノードで起動した際のバリア同期は，ノード内バリア同期とノード間バリア同期の組合せで実現される．各ノードに 1 プロセスのみ起動する場合はノード間バリア同期だけで処理される．

ここでは，各ノードで 1，2，4，5，6，7，8 個の MPI プロセスを起動させた条件下での MPI_Barrier の所要時間を，ノード数を 1 から 512 まで変化させて計測した．各条件について 200 回計測し，その平均をとった．

図 8 に各条件での平均所要時間を示す．縦軸が平均所要時間，横軸がノード数である．各ノードに 1 プロセスずつ起動させた場合のコストは約 3.25 マイクロ秒，各ノードに 8 プロセスずつ起動させた場合のコストは 64 ノード以下の場合で 8 マイクロ秒前後，512 ノード（4096 プロセス時）で約 40 マイクロ秒であった．1 プロセスの場合と 2 プロセス以上の場合の差は，主にノード内バリア同期のコストと考えられる．また全条件において平均所要時間と最短所要時間の比が 0.998 以上であり，性能が安定していることを確認した．

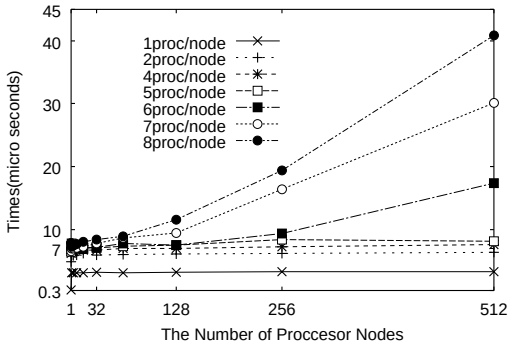

図 8 MPI_Barrier のスケーラビリティ
Fig. 8 Scalability of MPI_Barrier.

```
if (rank == 0) then
  call MPI_Barrier(...)
  t1=MPI_Wtime()
  call MPI_Put(...,1,...)
  call MPI_Win_fence(...)
  time = MPI_Wtime()-t1
else if (rank == 1) then
  call MPI_Barrier(...)
  call MPI_Win_fence(...)
endif
```

図 9 RMA 関数を用いた ping 通信コード
Fig. 9 Ping code using RMA functions.

処理時間が増大する原因の 1 つとしては，非同期通信メッセージの到着チェックの割込みが性能ツールであるプロファイル機能で確認された．また 8 プロセス時については，各ノードの AP が 8 個であることから OS 等のシステム側との競合も考えられる．

今後は，スケーラビリティ阻害要因の調査を含めてさらなる高速化について開発側と検討する予定である．

3.3 RMA 関数を用いた ping 通信性能

ここでは，RMA 関数の MPI_Put，MPI_Get，MPI_Accumulate を用いた ping 通信の通信性能について述べる．**図 9** のようにランク 0，1 間で ping 通信を行い，その所要時間を計測した．通信するデータ型は integer（4 byte 整数）で，MPI_Accumulate のオペレーションには総和を指定した．送受信バッ

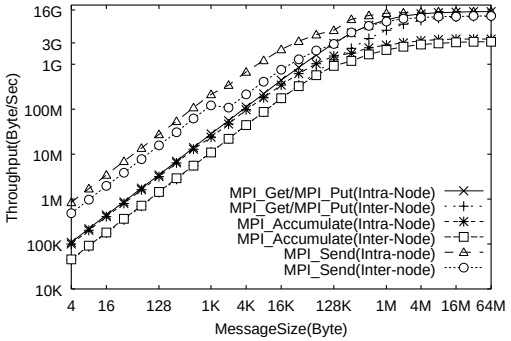

図 10 RMA 関数を用いた ping 通信時のスループット
Fig. 10 Throughput of ping using RMA functions.

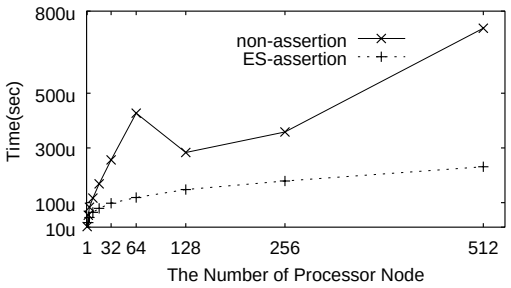

図 11 MPI_Win_fence のスケーラビリティ
Fig. 11 Scalability of MPI_Win_fence.

```
t1 = MPI_Wtime()
do i = 1, 200
  call MPI_Send(...)
enddo
latency=(MPI_Wtime()-t1)/200
```

図 12 レイテンシの計測コード（MPI_Send の場合）
Fig. 12 Latency measurement code using MPI_Send.

ファとウィンドウはグローバルメモリ上に確保し，1) MPI_Get/MPI_Put/MPI_Accumulate，2) ノード内通信/ノード間通信，といった条件を組み合わせた 6 つのケースについて計測した．通信メッセージ長は 4 B から 64 MB まで変化させ，各条件について 200 回計測して平均をとった．

図 10 に各条件でのスループットを示す．縦軸がスループット，横軸がメッセージ長である．比較のため，グローバルメモリを用いた場合の MPI_Send による ping-pong 通信でのスループットも示す．MPI_Put と MPI_Get はほぼ同じ性能であるため，図 10 のグラフでは重なっている．64 MB 時において通信性能はほぼ飽和しており，MPI_Put/MPI_Get に関しては，ノード内通信の最大スループットは 14.78 GB/s で，ノード間通信の最大スループットは 11.62 GB/s であった．MPI_Accumulate に関しては，ノード内通信の最大スループットは 3.66 GB/s で，ノード間通信の最大スループットは 3.16 GB/s であった．RMA 関数はグローバルメモリ使用時はつねに同一の通信方式で処理されるので，MPI_Send のような通信方式切替えによる性能ギャップはない．

所要時間の分散は 3 関数ともに 64 MB でのノード間通信時が最大であり，MPI_Put/MPI_Get で 0.40e-10，MPI_Accumulate で 0.30e-07 であった．また，そのときの平均性能値とピーク性能値の比は 3 関数ともに 0.99 であった．両者とも性能が安定していることが確認された．

3.4 RMA fence の性能

ここでは RMA 通信で用いられる MPI_Win_fence の性能について述べる．いくつかの MPI 関数では，その内部処理の最適化に有効な情報を MPI 処理系に与えることができるように，assertion 引数が定義されている．地球シミュレータ向け MPI では MPI_Win_fence について，(1) RMA 通信で MPI_Put または MPI_Get だけを利用している，(2) 送受信バッファとウィンドウをグローバルメモリ上に割り付けている，といった情報を MPI 処理系に与えるための assertion 仕様が実装されている．この assertion を MPI_Win_fence に引数として渡すことで，その内部処理は最適化され，実行時間の短縮が図られる．

ここでは，前述の最適化できる条件を満たしたうえで，assertion を指定せずにデフォルトの fence 処理を行った場合と指定して内部処理を最適化させた場合の所要時間を計測した．各ノードで 1 プロセスを起動させる条件でノード数を 1 から 512 まで変化させ，各々での所要時間を 200 回計測し，それらの平均をとった．

図 11 に，2 種類の fence の平均所要時間を示す．縦軸が平均所要時間，横軸がノード数である．図中の non-assertion が指定しない場合で，ES-assertion が指定した場合である．指定した場合の fence 処理が，指定しない，すなわちデフォルトの fence 処理より高い性能を持つことが確認された．また全条件において平均所要時間と最短所要時間の比は 0.999 以上で，性能が安定していることを確認した．

なおデフォルトの fence 処理では 64 ノードでのコストが 128 ノードでのコストより高いが，これは MPI_Win_fence 内部で用いられる MPI_Allreduce の問題と開発側は考えている．

3.5 MPI-1/MPI-2 関数のレイテンシ

ここでは主要な MPI-1/MPI-2 関数のレイテンシについて述べる．**図 12** のように 200 回連続でコールした場合の 1 コールに要する処理時間をレイテンシ

表 3 レイテンシ
Table 3 Latencies.

関数名	ノード間	ノード内
MPI_Send	5.58	1.38
MPI_Isend	5.90	1.75
MPI_Put	6.36	1.35
MPI_Get	6.68	1.27
MPI_Accumulate	7.65	3.87

（単位：マイクロ秒）

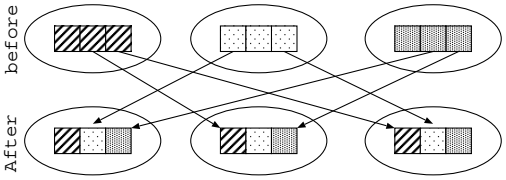

図 13 exchange 通信
Fig. 13 Exchange communication.

とした．メッセージ長は，MPI_Send/MPI_Isend では 0 byte，MPI_Put/MPI_Get/MPI_Accumulate では 4 byte とした．これは RMA 関数で 0 byte を指定すると内部処理をスキップしてしまうためである．送受信バッファとウィンドウはグローバルメモリ上に割り付けた．

各関数のレイテンシを**表 3** に示す．地球シミュレータ設計時において，MPI 関数の目標レイテンシは 6〜10 マイクロ秒と設定されていたが，この計測によって設計目標値が達成されていることが確認された．

3.6 exchange 通信での性能評価

ここでは，より複雑な通信パターンでの性能として，exchange 通信での性能について述べる．exchange 通信では，**図 13** のように各プロセスは自分以外の全プロセスとデータの送受信を行う．プログラミングでは**図 14** (a) のようにループ文を用いるのが一般的だが，全プロセスで同じ順番でメッセージを送信すると，多数のメッセージが一時に 1 つのプロセスに到着して受信処理が滞ってしまったり，通信経路選択でメッセージ衝突が生じたりしやすい．ネットワークトポロジーや MPI の実装方法によっては大幅な性能の悪化がありうる．地球シミュレータのノード間ネットワークはクロスバスイッチなので，図 14 (b) のように転送順序を操作することで衝突のない通信が可能である．

exchange 通信の実現方法としてはいくつか考えられるが，MPI_Isend を用いる方法（**図 15**），MPI_Send を用いる方法（**図 16**），MPI_Put を用いる方法（**図 17**）の 3 種について計測した．いずれも図 14 (b) のようにメッセージ衝突を起こしにくいようにコーディングした．送受信バッファとウィンドウはグローバルメモリ上に割り付けた．この 3 種類について 1 ノードに 1 プロセスのみ起動する条件で，使用ノード数を 16，32，64，128 とした場合の所要時間を MPI ランク 0 のプロセスにおいて各々200 回計測し，それらの平均をとった．

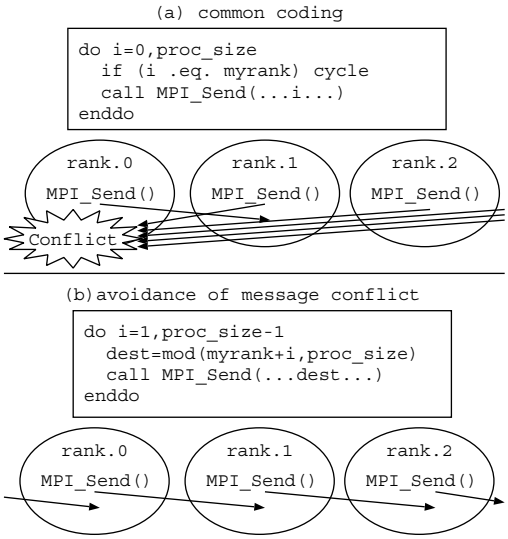

図 14 メッセージ衝突とその回避
Fig. 14 Message conflict and its avoidance.

```
! Already posted receive requests
do i = 1,size-1
  dest=mod(rank+i,size)
  call MPI_Isend(...,dest,...)
enddo
call MPI_Waitall(...)
```

図 15 MPI_Isend を用いる方法
Fig. 15 Implementation using MPI_Isend.

```
! Already posted receive requests
do i = 1,size-1
  dest=mod(rank+i,size)
  call MPI_Send(...,dest,...)
enddo
call MPI_Waitall(...)
```

図 16 MPI_Send を用いる方法
Fig. 16 Implementation using MPI_Send.

```
do i = 1,size-1
  dest=mod(rank+i,size)
  call MPI_Put(...,dest,...)
enddo
call MPI_Win_fence(...)
```

図 17 MPI_Put を用いる方法
Fig. 17 Implementation using MPI_Put.

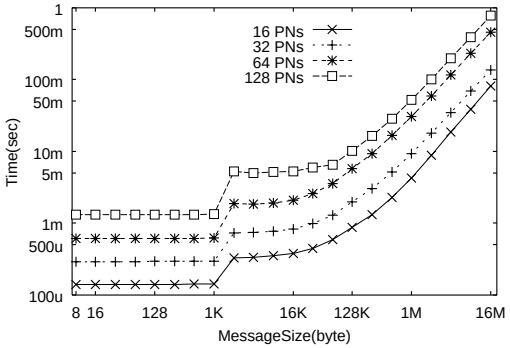

図 18 MPI_Isend を用いる方法での処理時間
Fig. 18 Performance of implementation using MPI_Isend.

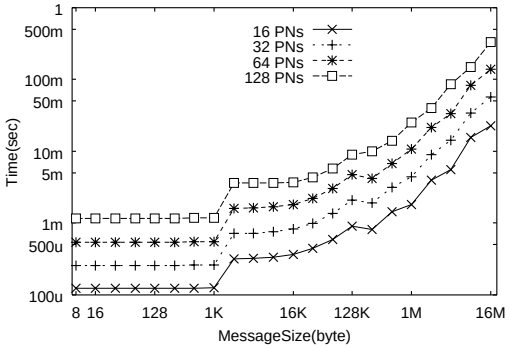

図 19 MPI_Send を用いる方法での処理時間
Fig. 19 Performance of implementation using MPI_Send.

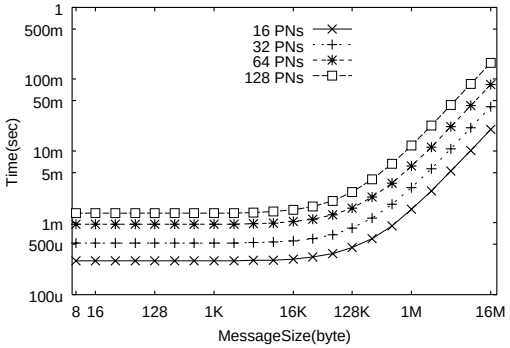

図 20 MPI_Put を用いる方法での処理時間
Fig. 20 Performance of implementation using MPI_Put.

図 18，**図 19**，**図 20** に，各方法での平均処理時間を示す．縦軸が所要時間，横軸が 1 メッセージのサイズである．どの方法もプロセス数比でのスケーラビリティは良好であった．MPI_Isend や MPI_Send を用いる方法は，メッセージ長が長くなるにつれて処理時間が特に延び，性能が落ちている．また MPI_Send を用いる方法は 1 KB 付近と 200 KB 付近に，MPI_Isend を用いる方法は 1 KB 付近に性能ギャップが生じている．MPI_Isend は 1 KB で MPI_Send 同様に通信方式切替えを行っているので，これらのギャップは通信方式切替えによるものと思われる．

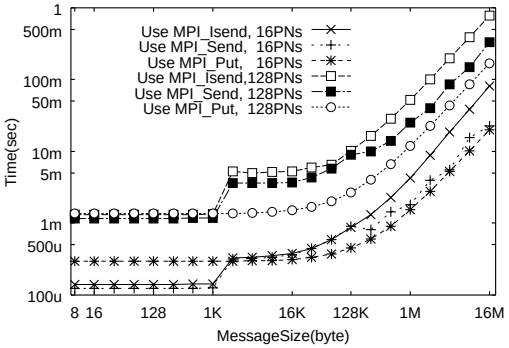

図 21 3 種類の方法での性能比較
Fig. 21 Comparison of results.

3 種類の方法について，16 ノードと 128 ノードで実行した場合の処理時間を，**図 21** で比較した．16 ノード時の 1 KB 以下のメッセージ長では，MPI_Put を用いる方法よりも MPI_Isend や MPI_Send を用いる方法の方が性能が良い．しかし，より長いメッセージ長の場合，あるいは 128 ノードの場合では MPI_Put を用いる方法が同等以上の性能を示している．その理由としては，MPI_Send や MPI_Isend を用いた場合では送受信プロセス間でハンドシェイク処理を行うが，RMA である MPI_Put ではハンドシェイク処理を行わないので，その点が特に多数のプロセスで通信した場合に有効になることが考えられる．また MPI_Put はグローバルメモリ使用時にはつねにシングルコピー処理するので，メッセージ長によってはシングルコピーしない MPI_Send や MPI_Isend よりも効率的に通信できることも理由として考えられる．

なお図 14 (b) のように送信先をずらしても，各プロセスの動作がずれてしまった場合にはメッセージ衝突が起こりうる．これは，ブロッキング通信を用いているならば**図 22** のようにバリア同期で回避できる．この例では，全プロセスでの MPI_Send コールの終了が MPI_Barrier によって保証されるので，プロセスの動作ずれによるメッセージ衝突は起こらない．

この MPI_Send を用いた場合でのバリア同期の有無による性能変化を**図 23** に示した．図 23 では，バリア同期をとる場合ととらない場合，さらに比較用に MPI_Put を用いる方法での各々16 ノードと 128 ノードでの計測結果を示した．縦軸は処理時間，横軸は 1 メッセージのサイズである．メッセージ長が長い場合

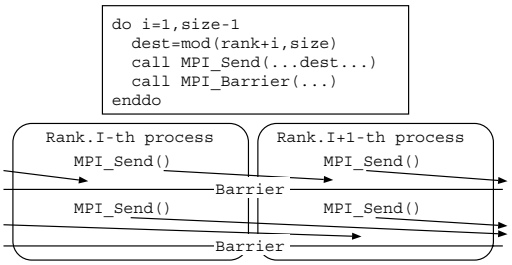

図 22 MPI_Send と MPI_Barrier による衝突回避
Fig. 22 Avoidance using MPI_Send and MPI_Barrier.

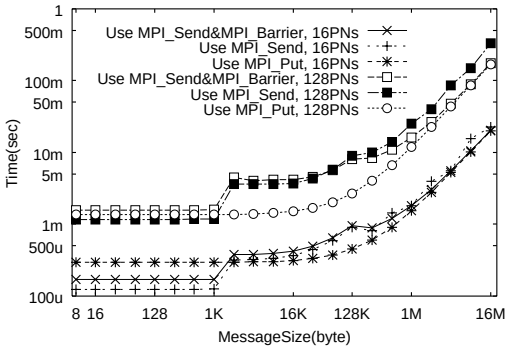

図 23 バリア同期の有無による影響
Fig. 23 Influence of MPI_Barrier.

はバリア同期をとった方が処理時間が短い．これは，メッセージ長が長い場合にはメッセージ衝突からの回復コストが大きいために，バリア同期のコストを加えてもメッセージ衝突を回避した方が全体としては速くなるためと思われる．この傾向はメッセージ長が 1 MB を超えたあたりから顕著である．また 16 ノード時では，必ずしもバリア同期をとった方が性能が良いわけではない．これはプロセス数が少ないために全プロセスの動作がほぼ同期していてバリア同期をとる意味がなく，バリア同期をとった場合にはそのコスト増というデメリットだけが見えているためと思われる．しかし，地球シミュレータ用アプリケーションではしばしば，長いメッセージを通信し，かつ相当な MPI プロセス数で実行されることを考慮すると，バリア同期を併用した MPI プログラミングが地球シミュレータに適していると考えられる．

なお，所要時間に関する分散の最大とその条件（ノード数とメッセージ長）は，MPI_ISend を用いる方法と MPI_Send のみを用いる方法では，各々，0.54e-02，0.64e-02（ともに 128 ノード，16 MB 時）であった．MPI_Barrier と MPI_Send を併用する方法では 0.25e-06（32 ノード，4 MB 時），MPI_Put を用いる方法で 0.55e-07（32 ノード，16 MB 時）であった．MPI_Barrier と MPI_Send を併用する方法と MPI_Put を用いる方法の 2 種類が，性能の安定性において優れていることが確認された．

表 4 乱流計算ソフト Trans7 での通信性能
Table 4 Communication performance of Trans7.

メモリ配置，使用 MPI 関数	計測範囲	通信処理
ローカルメモリ，MPI_Send	299.36	87.53
ローカルメモリ，MPI_Put	305.94	93.08
グローバルメモリ，MPI_Send	298.70	86.86
グローバルメモリ，MPI_Put	292.10	80.24

（単位：秒）

3.7 アプリケーションの通信性能と性能予測

ここでは，一様等方性乱流シミュレーション Trans7 で，送受信バッファとウィンドウのメモリ配置（ローカルメモリ，グローバルメモリ）と使用する MPI 関数（MPI_Send，MPI_Put）を変化させた場合の通信性能を計測した．MPI_Send を用いる場合には図 22 のように MPI_Barrier も併用した．格子点数 1024^3 の計算を 100 ループ，1 ノードに 1 プロセスのみ起動する条件で 64 ノード用いた際の実行時間と通信時間を計測した．1 回のループで 24 回の exchange 通信が行われる．1 回の送信でのメッセージ長は 6150 KB で，計算と通信はオーバラップしていない．

計測結果は**表 4** に示したが，グローバルメモリを用いた MPI_Put による実現が最も高速であった．ただしローカルメモリを用いた場合では，MPI_Send を用いた方が MPI_Put を用いた方よりも高速であったことは興味深い．これは地球シミュレータ向け MPI での RMA 関数がグローバルメモリとの併用を前提に最適化されているためと思われる．

この Trans7 での通信処理を，6 MB のメッセージ長での exchange 通信を 64 MPI プロセスで 2400 回行ったものと見なすと，前節でのグローバルメモリを用いた計測結果から，MPI_Put を用いた場合は 79.2 秒，MPI_Send を用いた場合は 84.8 秒と所要時間を算出できる．これらの算出値と実測値のずれは 2.4%以下であり，十分な精度と思われる．これらから，本論文で述べた性能評価値は，実際のアプリケーションの通信性能予測にも活用できるものと考える．

4. 効率的な通信の実現方法

地球シミュレータでは，3.1 節で述べたように MPI_Send の最大スループットはノード内通信で 14.8 GB/s，ノード間通信で 11.8 GB/s と，3.3 節で述べた MPI_Put/MPI_Get でのそれ（14.78 GB/s，11.62 GB/s）よりも若干高い．しかし MPI_Send を

使いさえすれば高効率な通信ができるわけではないことは，3.6 節から明らかである．

地球シミュレータで効率的な通信を行うには，グローバルメモリの利用が重要なポイントの 1 つとなる．今回の計測では，ノード内通信/ノード間通信各々で，グローバルメモリを用いた通信ではローカルメモリを用いた通信と同等以上の性能が得られた．よって地球シミュレータ上で通信性能の向上を図るならば，通信データをグローバルメモリに配置することが不可欠といえる．

次に，ユーザが地球シミュレータ向けアプリケーションをプログラミングするうえで用いる MPI 関数は，メッセージ長と通信パターン，実行時のプロセス数等を考慮することで選択できる．たとえば ping-pong 通信のように単純かつ少数プロセスで実行される場合には，最大スループットのより高い MPI_Send が望ましい．これは図 10 からも確認できる．

一方で，exchange 通信のように多数のプロセスで通信する場合では，図 21 から見て MPI_Put の方が効率的な通信を実現できる可能性は高い．またグローバルメモリを併用した MPI_Put では通信方式切替えによる性能低下が見られないので，様々なメッセージ長での通信を行いながら良好な性能を得たい場合にも MPI_Put を用いたプログラミングが簡便と思われる．

ただし MPI-1 のみに慣れ親しんだユーザには，MPI-2 で定義された MPI_Put 等を用いた RMA プログラミングはやや難しく，開発効率が落ちる可能性がある．そういったユーザが特に長いメッセージを通信するプログラムを開発する場合には，図 23 に見られるように MPI_Put を用いた場合に迫る性能を示した MPI_Send と MPI_Barrier を組み合わせた実装をするとよい．

地球シミュレータ上での MPI_Barrier のコストは一般的な計算機でのそれに比べてかなり低い．ゆえに exchange 通信以外でも，多数のプロセスによる通信であるならば，バリア同期の活用が総合的な意味での効率的な通信の実現につながる可能性は高いと思われる．また 1 ノード 1 プロセス時に特に低いので，ノード内も MPI で並列化するフラットプログラミングではなく，ノード内並列はコンパイラで自動並列化するハイブリッドプログラミングが，地球シミュレータ上では他の SMP クラスタに比べて効果的になりやすい[10),11)]．実際に地球シミュレータ上で運用利用している大気大循環シミュレーションも，本測定の結果をうけてハイブリッドプログラミングで実現されている．

5. おわりに

本論文では，地球シミュレータでの MPI 性能について評価した．MPI_Send を用いた ping-pong 通信の最大スループットはノード内通信で 14.8 GB/s，ノード間通信で 11.8 GB/s で，各関数のレイテンシも設計目標値を達成した．MPI_Barrier や MPI_Win_fence については高いスケーラビリティが確認された．exchange 通信での評価では，いずれの方式にも高いスケーラビリティが確認され，特に MPI_Put を用いた方法と MPI_Send と MPI_Barrier を併用した方法が優れていた．後者では，MPI_Barrier によるコスト増大を加味しても処理時間全体の減少が確認された．また本論文で示した性能値から，アプリケーションでの通信性能を予測できた．これらは地球シミュレータのユーザにとって有益な情報と考える．

参考文献

1) Yokokawa, M., Habata, S., Kawai, S., Ito, H., Tani, K. and Miyoshi, H.: Basic Design of the Earth Simulator, *High Performance Computing* (LNCS1625) (1999). ISHPC'99.
2) 谷 啓二，横川三津夫：地球シミュレータ計画，情報処理，Vol.41, No.3, pp.249–254 (2000).
3) 横川三津夫，谷 啓二：地球シミュレータ計画，情報処理，Vol.41, No.4, pp.369–374 (2000).
4) MPI Forum: MPI: A Message-Passing Interface Standard (1995).
5) MPI Forum: MPI-2: Extensions to the Message-Passing Interface (1997). http://www.mpi-forum.org
6) Pallas: *Pallas MPI Benchmarks.* http://www.pallas.com
7) Rolf Rabenseifner: Effective Bandwidth Benchmark.http://www.hlrs.de/organizations/par/services/models/mpi/b_eff/
8) 上原 均，津田義典，横川三津夫：MPI-2 用ベンチマークプログラムライブラリ MBL2 の構築と評価，情報処理学会研究報告 2001-HPC-87, Vol.2001, No.77, pp.67–72 (2001).
9) Uehara, H., Tamura, M. and Yokokawa, M.: An MPI Benchmark Program Library and Its Application to the Earth Simulator, *High Performance Computing* (*Proc. ISHPC2002*), LNCS2327 (2002).
10) 板倉憲一，宇野篤也，上原 均，斎藤 実，横川三津夫：地球シミュレータ上のハイブリッドプログラミングの性能評価，情報処理学会研究報告 HPC 90-4, pp.19–24 (2002).
11) 宇野篤也，板倉憲一，横川三津夫，石原 卓，金田行雄：地球シミュレータ上での流体コードのス

表 5 MBL での主な性能測定対象
Table 5 Major measurement items of MBL.

主な項目	計測内容
1 対 1 関数	MPI_Send や MPI_Isend 等の主要 1 対 1 送信関数（ping，ping-pong）
集合通信	MPI_Bcast や MPI_Gather 等の主要集合通信関数
並行通信	shift，exchange 通信
リモートメモリアクセス（RMA）	MPI_Get/Put/Accumulate（ping，shift，exchange 通信）
MPI-I/O	MPI_Win_fence，MPI_Win_create MPI_File_read，MPI_File_write 等の全 MPI-I/O 関数，MPI_File_Sync

ケーラビリティ評価，情報処理学会研究報告 HPC 91-10, pp.55–60 (2002).

付　　録

A.1 MBL の概要

MBL は，MPI-1/MPI-2 に関して詳細かつ多角的な測定を行うことを特長としたベンチマークソフトウェアである．主な評価項目は**表 5** のとおりで，shift や exchange 等の実際のアプリケーションでしばしば用いられる通信パターンについては複数のプログラミング（例：RMA 通信を用いたプログラミング等）について計測し，性能差や特性の相違を直接的に評価できる．大規模計算機環境では特に重要な評価対象であるが，PMB [6] では計測できない File Sync や RMA fence も計測する．MBL の実現ではフレームワーク技術を用い，保守性や拡張性が高い．想定した MPI ユーザが Fortran 利用者であることから，Fortran で記述している．

(平成 14 年 6 月 7 日受付)
(平成 14 年 10 月 10 日採録)

上原　　均（正会員）

昭和 45 年生．平成 12 年茨城大学大学院理工学研究科博士後期課程情報・システム科学専攻修了．博士（工学）．平成 12 年日本原子力研究所地球シミュレータ研究開発センター博士研究員．平成 14 年 3 月より海洋科学技術センター地球シミュレータセンター研究員．並列分散計算，可視化技術等の研究に従事．

田村　正典

昭和 37 年生．昭和 58 年詫間電波工業高等専門学校卒業．同年日本電気株式会社入社．科学技術系言語処理の開発に従事．平成 5 年より分散メモリ型並列処理の研究に参加，特に HPF（High Performance Fortran）および MPI（Message Passing Interface）の研究開発に従事．

板倉　憲一（正会員）

昭和 44 年生．平成 5 年筑波大学第三学群情報学類卒業．平成 11 年同大学院工学研究科博士課程修了．博士（工学）．平成 11 年より 12 年まで筑波大学計算物理学研究センターリサーチ・アソシエイト．平成 13 年日本原子力研究所地球シミュレータ研究開発センター博士研究員．平成 14 年より海洋科学技術センター地球シミュレータセンター研究員．計算機性能評価技術，並列計算機アーキテクチャ等の研究に従事．IEEE-CS 会員．

横川三津夫（正会員）

昭和 35 年生．昭和 59 年筑波大学大学院修士課程理工学研究科修了．同年日本原子力研究所入所．原子力分野における高速数値シミュレーションの技術開発に従事．平成 9 年地球シミュレータ研究開発センターにて「地球シミュレータ」を開発．平成 14 年 7 月産業技術総合研究所グリッド研究センター．平成 6 年〜7 年コーネル大学コーネル理論センター客員研究員．工学博士．大規模数値シミュレーション，並列数値計算に興味を持つ．日本応用数理学会会員．