ハイパフォーマンス
コンピューティング 90－4
（２００２．５．２７）

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

板倉憲一[†] 宇野篤也[†] 上原均[†]
斎藤実[††,☆] 横川三津夫[††]

地球シミュレータは，640 の計算ノードを持ち，理論ピーク性能は 40Tflop/s である．各ノードはベクトルプロセッサ 8 個と 16GB の共有メモリから構成される．ノード間の並列処理手法は MPI によって行う．計算ノードにおける主要な並列処理手法は，1) ノード内 MPI プロセス 2) マイクロタスクによる自動並列化である．ノード内の 2 つの並列化手法は計算処理性能と通信性能とそれぞれにおいて違いがあるため，これらの特性をふまえて並列化のチューニングを行う必要がある．本研究では，a) 単純なノード内の計算性能評価，b) ノード間データ転送処理性能評価，c) 計算とデータ転送を併せたアプリケーションによる性能評価を行う．これによって，地球シミュレータの特性を明らかにし性能チューニングのポイントを明示する．

3 次元 FFT を用いた一様等方性乱流シミュレーションコードの 256^3 と 512^3 の問題サイズを 8 ノード (64APs) において評価を行った．それぞれ 4 秒と 30 秒程度の実行時間に対してノード内をマイクロタスクの自動並列化を行うハイブリッドプログラミングと全てを MPI で並列化するフラットプログラミングを比較した時，その差は 1 秒弱であった．この結果，ハイブリッドプログラミングは，フラットプログラミングとほぼ同程度の性能が出せることが分った．

Performance Evaluation of Hybrid Programming on Earth Simulator

Ken'ichi ITAKURA,[†] Atsuya UNO,[†] Hitoshi UEHARA,[†]
Minoru SAITO[††,☆] and Mitsuo YOKOKAWA [††]

The Earth Simulator has 640 processor nodes and its peak performance is 40 Tflop/s. Each node has 8 vector processors, each of which has 8 Gflop/s peak performance, and 16GByte shared main memory.

There are two programming methods on a node of the Earth Simulator. One, called flat programming, is MPI on shared memory architecture, the other, called hybrid programming, is "microtask" processing by automatically parallelization by the compiler. In this study, we have evaluated three basic performances. The first one is a calculation time in a node with 8 vector processors which include microtask starting and closing overhead or MPI barrier. The second one is a data transfer time between two nodes with 1-by-1 MPI processes or 8-by-8 MPI processes. The last one is a time for application with the calculation and data transfer.

Finally, we evaluated an application program which is large-scale direct numerical simulations of the Navier-Stokes equations. The most of calculation time of this application is 3 dimensional FFT. The total running time with 8 nodes (64 APs) are 4 and 30 seconds for 256^3 and 512^3 problem size, respectively. Since the difference time between two programming models is almost 1 second, the hybrid programming can be achieved the same performance as the flat programming.

1. はじめに

地球シミュレータ [1)2)] は，気候・気象分野及び固体地球のシミュレーションを始めとした様々な大規模問題に挑戦するための超高速並列計算機である．本体製作及び海洋科学技術センター横浜研究所 (横浜市金沢区) に建設された地球シミュレータ棟への据付調整作業が平成 14 年 2 月末に完了し，同年 3 月より運用を開始している．

本研究では，地球シミュレータのような大規模なベクトル型並列計算機の利用方法を促進させるアルゴリズム解析およびコード開発の一つとして，ノード内並列化手法による実効性能の違いを解析する．

† 海洋科学技術センター
Japan Marine Science and Technology Center
†† 日本原子力研究所
Japan Atomic Energy Research Institute
☆ 現在，(株)NEC 情報システムズ
Presently with NEC Informatec Systems, Ltd.

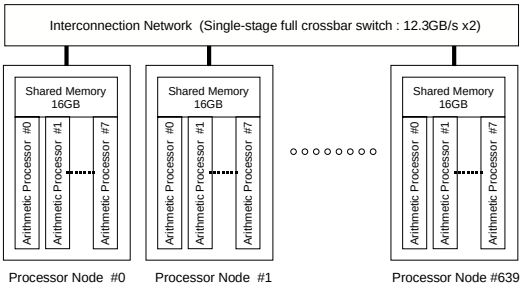

図 1 地球シミュレータ概念図

2. 地球シミュレータのアーキテクチャ

地球シミュレータ概念図を図 1に示す．地球シミュレータは 640 台の 計算ノード (PN: Processor Node) を単段のクロスバネットワークで結合する分散メモリ型並列計算機である．地球シミュレータで想定されるアプリケーションプログラムでは，全てのプロセッサ間での通信が高速に行われる必要性があるとともに，自由なジョブの配置を可能とするため，このような柔軟性のあるネットワークを採用した．

PN はピーク性能 8Gflop/s の計算用ベクトルプロセッサ (AP: Arithmetic Processor) 8 台が 16GB の主記憶を共有する共有メモリ型並列計算機である．したがって，地球シミュレータ全体では AP 5120 台， ピーク性能は 40Tflop/s， 主記憶 10TB となる．各 PN の主記憶は，32 台の主記憶ユニット (MMU: Main Memory Unit) から構成される．PN にはその他にリモートアクセス制御装置 (RCU: Remote Access Control Unit) 及び入出力プロセッサ (IOU: I/O Processor) を持っている．主要な LSI には 0.15μm CMOS テクノロジーを，冷却方式には空冷を採用した．

AP はベクトル処理部 (VU: Vector Unit)， スカラ処理部 (SU: Scalar Unit)，プロセッサネットワークユニット (PNU: Processor Network Unit) 及びアドレス制御部 (ACU:Address Control Unit) からなり，1LSI(20.79mm × 20.79mm) で実装されている．SU は 4 ウェイのスーパースカラであり 128 個の汎用レジスタ， 2 ウェイセットアソシエティブ方式の命令キャッシュとデータキャッシュをそれぞれ 64KB づつ実装している．さらに，分岐予測や投機実行機能も持つ．VU は 6 種類 (加減算，乗算，除算，論理，ビット列演算，ロード/ストア) のベクトル演算器と 72 個のベクトルレジスタからなるベクトル演算器セット 8 個で構成され，最大 8Gflop/s の性能を有している．32 個の MMU には主記憶素子として DRAM ベースの 128Mbit の高速 RAM を採用し，2048 バンク構成とした. 各々の AP は主記憶システムとの間に 32GB/s のバンド幅を持っており，1PN で 256GB/s を確保している．

地球シミュレータの OS は NEC の SX シリーズで使われている SUPER-UX をベースにしており，超大規模システム向けに拡張されている．プログラムの開発環境は以下の通りである．

- HPF コンパイラ: HPF2.0 に加えて，JAHPF がまとめた拡張仕様 HPF/JA1.0 に準拠
- Fortran90: JIS 規格に完全準拠．自動ベクトル化，ノード内自動並列化，OpenMP 機能を含む
- C/C++コンパイラ: ISO 国際規格に準拠．自動ベクトル化，ノード内自動並列化，OpenMP 機能を含む
- MPI-1，MPI-2 ライブラリ: 地球シミュレータの通信機構向けに最適化されたメッセージパッシング型並列処理ライブラリ

3. ハイブリッドプログラミング

地球シミュレータのアーキテクチャは SMP クラスタである．これは，現在大規模な並列システムを作成する上で標準的なものである．分散メモリアーキテクチャ上でソフトウェア共有メモリをサポートするシステムもあるが，その性能は実用的なレベルに達してはいない．よって，ノード間が分散メモリアーキテクチャでは，1) ノード内の共有メモリ並列とノード間の分散メモリ並列を組み合わせて並列化する手法と 2) 全てのプロセッサを論理的にフラットな構成とみて全て MPI で並列化する手法が現実的である．ここでは，1) をハイブリッドプログラミング，2) をフラットプログラミングと呼ぶ．

地球シミュレータで推奨されているプログラミング方法は，ハイブリッドプログラミングに AP 内のベクトル化を行った 3 重の並列化である．ノード内の共有メモリ並列処理には，マイクロタスクによる自動並列化を利用する．マイクロタスクは，基本的にサブルーチン単位の生成/消滅を行う．更にサブルーチン内変数を共有し DO ループなどの処理を分割して行う．マイクロタスクの DO ループの分割などはコンパイル指示行によって行う．コンパイラの自動並列化はループの依存関係を調べ，並列化可能なループのサブルーチン化とマイクロタスク指示行の挿入によって行われる．

一般に，ハイブリッドプログラミングよりもフラットプログラミングの方が性能が出ることが報告されている．地球シミュレータは，これまでにないベクトルプロセッサを用いた大規模 SMP クラスタであり，このアーキテクチャにおいて，プログラミング手法による性能相違を把握し，実アプリケーションプログラムに対する性能チューニングの指標を示すことが本研究の目的である．

本研究での性能評価で用いたプログラムのコンパイ

ル環境は以下の通りである．

- プログラム言語: Fortran90
- 自動並列化有コンパイルオプション: -P auto -C hopt -gmalloc -float0 -V -R2 -Wf"-pvctl noloopchg fullmsg vwork=stack"
- 自動並列化無コンパイルオプション: -C hopt -gmalloc -float0 -V -R2 -Wf"-pvctl noloopchg fullmsg vwork=stack"

4. ノード内計算性能評価

最初に，ノード内計算性能とプログラミング手法の関係を評価する．ここでは，ノードあたりの確保するメモリ領域量をハイブリッドプログラミングとフラットプログラミングで共通にし，簡単な DO ループによるベクトル加算処理:

```
do i=1, n
  a(i)=a(i)+1.0d0
enddo
```

に対してマイクロタスクの並列化と MPI の並列化 (図 2) の性能差を測定した．結果を表 1に示す．

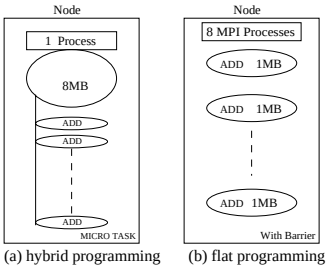

図 2 ベクトル加算処理

表 1 ベクトル加算処理の時間

メモリ領域 [Byte/node]	ハイブリッド [sec]	フラット [sec]	差 [sec]
16K	6.42e-05	4.16e-07	6.38e-05
32K	6.39e-05	4.64e-07	6.35e-05
64K	6.41e-05	7.36e-07	6.34e-05
128K	6.40e-05	1.25e-06	6.28e-05
256K	6.60e-05	2.26e-06	6.37e-05
512K	6.83e-05	4.31e-06	6.40e-05
1M	7.29e-05	8.42e-06	6.45e-05
2M	8.08e-05	1.66e-05	6.42e-05
4M	9.75e-05	3.30e-05	6.45e-05
8M	1.28e-04	6.58e-05	6.22e-05
16M	1.94e-04	1.31e-04	6.24e-05
32M	3.25e-04	2.62e-04	6.24e-05
64M	5.87e-04	5.25e-04	6.23e-05
128M	1.11e-03	1.05e-03	6.41e-05
256M	2.16e-03	2.10e-03	6.42e-05
512M	4.26e-03	4.19e-03	6.44e-05
1G	8.45e-03	8.39e-03	6.43e-05

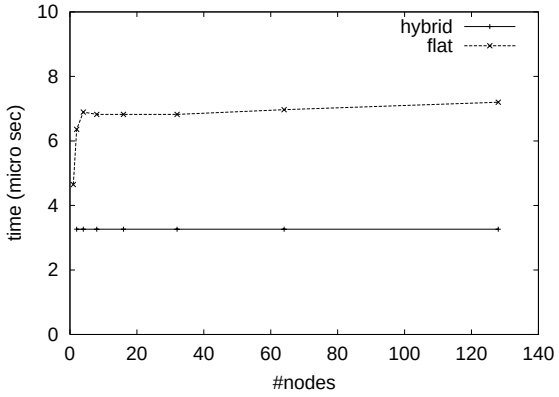

図 3 バリア同期性能

ハイブリッドプログラミングで用いるマイクロタスク自動並列化では，並列化の対象となるループをコンパイラがサブルーチン化をする．実行時には，スレッド生成，ループの動的範囲割り当て，サブルーチンコール，同期待ちの処理を行う．これに対して，MPI の並列化では，予め 1/8 の領域だけ MPI プロセスはメモリを確保し，独立したループ処理の後でバリア同期をとる．

表 1 の「差」で示した通り，フラットプログラミングに対してハイブリッドプログラミングの処理時間はメモリ領域のサイズにかかわらず，約 63 マイクロ秒大きい．フラットプログラミングでは，本来の加算処理の他にバリア同期のコストがある．後で詳細は示すがノード内 8 MPI の場合は約 4.6 マイクロ秒である．これに対して，ハイブリッドプログラミングでは，マイクロタスク起動および終了にかかるオーバヘッドのコストが加わる．このオーバヘッドは処理するデータ量によらず一定であるため，メインループ内に多数のマイクロタスクの並列化ループを含むアプリケーションプログラムでは，その並列化を外側のループに変更するか，並列化を断念するなどの対策を取る必要がある．

5. ノード間通信性能評価

5.1 バリア同期性能

ノード間通信性能の基本的なデータとして，バリア同期のコストは重要である．ハイブリッドプログラミングでは，ノードに MPI プロセスを 1 つしか割り当てないが，フラットプログラミングでは，ノードに 8 個の MPI プロセスを割り当てる．この環境でノード数とバリア同期コストの関係を評価した [3]．連続した MPI_Barrier の 2 回目にかかった時間を 300 回測定し，MPI のランク番号 0 における最短時間を結果とした．バリア同期時間を図 3 に示す．

ハイブリッドプログラミングでは，バリア同期はノー

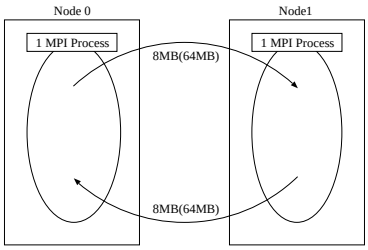

図 4　ハイブリッドプログラミングのピンポン転送

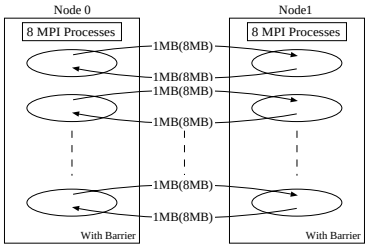

図 5　フラットプログラミングのピンポン転送

表 2　ピンポン転送の時間

メモリ領域 [Byte/node]	ハイブリッド [sec]	フラット [sec]
8MB	1.38e-03	1.43e-03
64MB	1.06e-02	1.07e-02

ド間でのみ行われる．地球シミュレータはノード間バリア用のハードウェアカウンタをクロスバネットワークのコントローラの部分に持っている．ノード間のバリア同期は，予めセットされた参加するノード数をデクリメントすることで実装されている．競合するデクリメントの要求は逐次化されるが，MPI_Barrier の呼出しのオーバヘッドやデクリメント命令が実際に発行されるのに要するレイテンシの時間に比べると短い時間で済むため，ノード数に関らずバリア同期の時間はほぼ一定で約 3.2 マイクロ秒である．一方，フラットプログラミングでは，1 ノード内の 8MPI プロセスでは約 4.6 マイクロ秒である．これはノード内にあるカウンタレジスタを用いて実装されている．複数ノードになった場合には，約 6.8 マイクロ秒である．この時は，最初にノード内での 8MPI プロセスでのバリア同期が行われ，その後でノード間のバリア同期がクロスバスイッチのカウンタによって行なれる．単純にノード内の時間とノード間の時間を合わせた時間より 1 マイクロ秒ほど速いが，これは呼出しのオーバヘッドが 1 回減ることによるものと考えられる．

両方のプログラミングモデルにおいて MPI のバリア同期は非常に高速であり，これを利用した最適化も行える．例えば，MPI での通信発行順序とノードからの転送順序，さらに単段クロスバスイッチでの出力ポート競合間での順序関係は実装依存であり，ユーザーレベルでは性能にどのような影響があるかはわかりにくい．そこで，全対全の通信でバリア同期をはさみながらユーザレベルでフェーズを区切り，転送先の衝突を回避する．具体的には，MPI_Irecv を前もって全て発行しておき，n 回目のフェーズで MPI_Barrier と mod(自ランク番号 $+n$, プロセス数) のランク番号のプロセスへの MPI_Send を行う．これによって，完全に通信先の衝突を回避した転置転送を実装すると，MPI_Barrier の無い場合よりも時間を短くできることを確認した．

5.2　ピンポン転送性能

次に，ピンポン転送の実験によってノード間の通信性能を測定した．ハイブリッドプログラミングではノード間通信は 1 対 1 の MPI プロセス間通信となる (図 4)．フラットプログラミングにおいては 8 組の 1 対 1 の MPI プロセス間通信でノード間のデータ転送量が同じになるようにする (図 5)．アプリケーションプログラムによっては，その並列方法によってノード間のデータ転送量自体が変ることもあるが，基本的な通信性能を測定するために，このような人工的な通信パターンによる評価を行った．なお，どちらのプログラミングモデルにおいてもバリア同期の時間が計測範囲に含まれている．

測定結果を表 2に示す．これは，ピンポン転送一往復を 100 回の測定したうちの最短時間である．この結果から，ハイブリッドプログラミングの方が転送時間が短いことがわかる．これは，ノードあたりのデータ転送回数がフラットプログラミングの方が多く，そのオーバヘッドが見えるためである．これはメッセージ長が短い程顕著に表われる．地球シミュレータのデータ転送性能の $N_{1/2}$ は 100KB 程度 [3] なので，8MB 転送 1 回と 1MB 転送 8 回では 3% 程度の性能差が表われるが，64MB 転送 1 回と 8MB 転送 8 回では性能差は 1%未満と小さい値となった．

6.　通信と計算の総合評価

6.1　単純化したモデルでの評価

ここでは，簡単化したモデルでの通信と計算の総合評価を行った．すなわち，最初に示したノード内の DO ループによるベクトル加算処理にノード間のピンポン転送を合せた時間を測定した．

測定結果を表 3に示す．この時間は，表 1に示したベクトル加算処理の 2 回分と表 2に示したピンポン転送を足したものとほぼ一致している．ノード内演算においてはマイクロタスクのオーバーヘッドによりフラットプログラミングが速く，ノード間転送においては転送オーバヘッドによりハイブリッドプログラミングが速い結果となっており，それを併せた本結果ではハイ

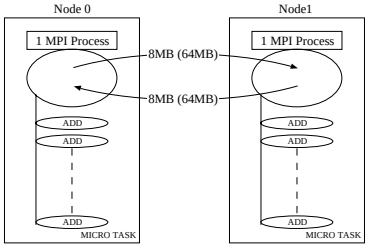

図 6　ハイブリッドプログラミングのピンポン転送+ベクトル処理

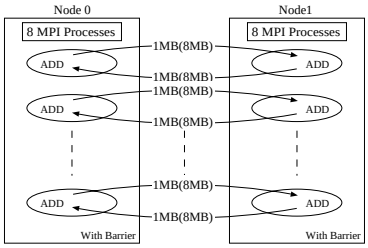

図 7　フラットプログラミングのピンポン転送+ベクトル処理

表 3　ピンポン転送+ベクトル処理の時間

メモリ領域 [Byte/node]	ハイブリッド [sec]	フラット [sec]
8MB	1.54e-03	1.59e-03
64MB	1.17e-02	1.19e-02

ブリッドプログラミングの方が速くなった．これは通信と処理のトレードオフによって決まるものであり，どちらの方式がすぐれているかは一概には言えないが，少くともハイブリッドプログラミングがフラットプログラミングに対してアドバンテージを持つ場合もあることが確認できた．

6.2　実アプリケーションによる評価

最後に，実アプリケーションでの評価例として一様等方性乱流シミュレーション [4] のコード (trans7) の評価を行った．このコードは，ナビエ・ストーク方程式の直接数値シミュレーション (DNS) をフリーエスペクトル法を用いて行う．主な処理は，3 次元 FFT であり，1 タイムステップにおいて逆変換 48 回，順変換 24 回を行う．MPI の 1 次元プロセス空間に対して，3 次元のデータを割り当てるため 1 回の変換につき全 MPI プロセスにおける転置転送が 1 回必要となる．

SX-5 での評価では，フラットプログラミングに対してハイブリッドプログラミングは遅い結果となっている [4]．しかし，本研究で行った基礎的な性能評価では，ハイブリッドプログラミングにはマイクロタスクの起動コストがあるものの，通信ではフラットプログラミングよりも良い性能を出しており，通信を多く行うアプリケーションモデルでは結果としてハイブリッドプログラミングの方が高速に処理が可能であると予測できる．

表 4　trans7 の結果 [sec]

問題サイズ	256^3		512^3	
	flat	hybrid	flat	hybrid
逆 FFT 演算	1.37	1.55	12.04	12.02
FFT 演算	0.77	0.97	6.65	7.09
データ通信	1.60	1.04	8.25	7.83
その他	0.33	0.36	2.55	2.64
合計	4.12	3.92	29.49	29.58

まず，地球シミュレータ上で動作しているソースプログラムに対して，4 基底の FFT を組み込み，ノード内処理の高速化を行った．さらに，通信データをノードの通信制御装置が直接アクセスできるグローバル領域 [3] に置き，MPI2 の MPI_Put による片側通信を用いる最適化を進めた．

3 次元領域のサイズが 256^3 と 512^3 の 2 つの問題サイズについて 10 タイムステップ分の実行時間を計測した．使用ノード数は 8(64AP) に固定しフラットプログラミングとハイブリッドプログラミングの比較を行った．実行したプログラムは，コンパイル/リンク時の自動並列化オプションの有無だけで，ソースは同じものを使用した．

評価結果を表 4に示す．FFT の演算部分でのマイクロタスク起動のオーバヘッドとデータ通信の起動オーバヘッドの時間がほぼ同じぐらいで，結果としてどちらの問題サイズでも総実行時間がほぼ同じ時間となった．

今後 4096^3 の問題を 512 ノードで実行することを考えている．この問題を 1 次元分割のままでフラットプログラミングをすると仮定する．1 辺 4096 要素の実数をフーリエ空間に変換すると 1 辺が 2048 の共役複素数となり，8AP × 512 ノード = 4096 の MPI プロセスで分割することができない．この場合プロセッサ空間を 2 次元とし問題空間を分けることになり，転置転送が 2 回必要になる．これに対して，ハイブリッドプログラミングでは，元々MPI による並列化の次元方向とマイクロタスクによる並列化の次元方向は別にでき，更に最後の次元方向をベクトル化することができるため，地球シミュレータのアーキテクチャに非常にマッチしたプログラミングモデルであると言える．本評価で，ハイブリッドプログラミングがフラットプログラミングとほぼ同じ性能であることがわかったので，今後の最大規模での実行はハイブリッドプログラミングで作成したコードで行う予定である．

7. 関 連 研 究

早川ら [5]，吉川ら [6] は 4way SMP-PC クラスタにおいて，また，Cappello ら [7] は Dual SMP-PC クラスタにおいて Pthread ライブラリや OpenMP を用い

たハイブリッド並列化の評価を行っている．この環境においては，メモリバスのボトルネックによる性能低下が大きいため，CPU のキャッシュメモリをいかに有効に活用するかがハイブリッドプログラミングの高速化の要点であり，フラットプログラミングの方がプログラムと CPU の関係を直接的に書けるため高速化しやすい．これに対して，地球シミュレータではノード内並列には十分なメモリバンド幅が用意されており，ベクトル処理機構によりキャッシュメモリを意識せずに高速化が行えるアーキテクチャとなっている．また，ノード内のマイクロタスク並列もコンパイラの自動並列化機構により OpenMP 以上に利用しやすい環境となっている．

8. おわりに

一般に SMP クラスタでは，ハイブリッドプログラミングはそのコストがかかる割に得られる速度向上が少なく，全てを MPI によるフラットなプログラミングを行って，MPI プロセスをプロセッサに 1 対 1 で対応させた方が良い結果となる場合が多い．これに対して，地球シミュレータでは，自動並列化によるノード内のマイクロタスク処理のオーバヘッドとノード間およびノード内通信のデータ転送性能との関係で，ハイブリッドプログラミングにおいてもフラットなプログラミングと変わらないか，より良い性能となることが分った．

共有メモリの利点を活かせるノード内並列処理は，ノード間並列とは別の軸で考えた方が有効にプロセッサを利用できる．共有メモリアーキテクチャでは，OpenMP や自動並列化マイクロタスクによってプログラミングにかかるコストは明らかに小さくなっている．フラットプログラミングをしながら，ノードの内外のアーキテクチャの違いを意識し通信の最適化をユーザが考えるよりは，ハイブリッドプログラミングの方が全体としてプログラミングのコストを低くできると考えられる．

謝辞

本研究に関して御議論頂いた，地球シミュレータ研究開発センター並びに地球シミュレータセンターの諸氏及び関係各位に感謝致します．

参考文献

1) 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).
2) 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).
3) 上原 均, 田村 正典, 横川 三津夫, 「地球シミュレータの計算ノード上での MPI 性能評価」, 計算科学シンポジウム HPCS2002 pp. 73–80 (2002).
4) 横川 三津夫, 斎藤 実, 石原 卓, 金田 行雄, 「地球シミュレータ上の一様等方乱流シミュレーション」, 計算科学シンポジウム HPCS2002 pp. 125–131 (2002).
5) 早川 秀利, 近藤 正章, 板倉 憲一, 朴 泰祐, 佐藤 三久, 「共有メモリ PC クラスタにおけるハイブリッド並列プログラムの性能評価」, 情処研報 99-HPC-77 pp. 131–136 (1999).
6) 吉川 茂洋, 早川 秀利, 近藤 正章, 板倉 憲一, 朴 泰祐, 佐藤 三久, 「SMP-PC クラスタにおける OpenMP+MPI の性能評価」, 情処研報 2000-ARC-137/2000-HPC-80 pp. 155–160 (2000).
7) Frank Cappello, Olivier Richard and Daniel Etiemble, “Investigating the performance of two programming models for clusters of SMP PCs”, IEEE HPCA6, January 2000.