ハイパフォーマンス
コンピューティング 79－9
（1999．12．3）

Cenju-4 における HPF の評価

高橋　正樹[†]　　末広　謙二[††]
妹尾　義樹[††]　　横川　三津夫[†]

HPF は，MPI と同様に分散メモリ型並列計算機上のプログラミングインタフェイスの一つであり，簡単かつ明瞭にプログラミング出来るように仕様が決められた高級なデータ並列言語である．ユーザは，分散メモリ上のデータ配置を指定する指示行を挿入して，逐次プログラムを並列化することができる．地球環境変動研究のために開発されている地球シミュレータにおいても，HPF をプログラミングインタフェイスの一つとして採用する計画である．

本稿では，逐次実行用に作られている既存のアプリケーションプログラムを HPF を用いて分散並列化し，HPF のプログラム言語としての記述性及び実行性能について評価した．この結果，プログラムのわずかな修正，追加で十分な並列実行性能が得られた．

An Evaluation of HPF Implementation on Cenju-4

Masaki Takahashi [†], Kenji Suehiro [††], Yoshiki Seo [††]
and Mitsuo Yokokawa [†]

High Performance Fortran (HPF) is considered one of the major parallel programming interfaces as well as Massage Passing Interface (MPI). HPF is a high-level data parallel language designed to provide a clear and easily understood programming interface. Users can parallelize their sequential programs by mainly inserting directives specifying data mapping on distributed memories. We plan to adopt HPF as a common parallel programming interface on the Earth Simulator, which is a distributed memory parallel system under development mainly targeting earth science.

In this paper, we evaluated efficiency and describability of HPF by parallelizing two application programs originally developed for sequential execution. The evaluation results showed that users can obtain good scalability by HPF programming with relatively small effort.

1. はじめに

近年，科学技術計算分野においても種々の並列計算機が普及しており，大規模並列応用ソフトウェアが開発されるようになった．しかし，それらのメモリアーキテクチャは，分散メモリ型，共有メモリ型，およびそれらを組み合わせたものなどさまざまであり，並列プログラム開発の困難な点となっている．

並列計算機向けのプログラミングインタフェイスとしては，MPI-2 (Extensions to the Message-Passing Interface)[1]，HPF (High Performance Fortran)[2][3]，および OpenMP[4][5] が標準仕様としてまとめられ，特に共有メモリ型並列計算機向けには OpenMP が有望であると考えられている．しかし，大規模分散メモリ型の並列計算機に対してはデータの分散配置を考慮する必要があるため，OpenMP では並列プログラムを記述できない．

† 日本原子力研究所
Japan Atomic Energy Research Institute
†† NEC C&C メディア研究所
C&C Media Research Laboratories, NEC Corporation

一方，分散メモリ型並列計算機向けのプログラミングには MPI が多く使われているが，MPI はデータ転送手続きの記述が複雑で，既存の逐次処理型のアプリケーションプログラムを並列化する場合や新規に分散並列プログラムを開発する場合には，高度なプログラミング技術と相当なプログラミング量が必要になり，開発コストの点で問題がある．

それに対して HPF は，既存の逐次処理型プログラムに指示行を追加することで並列化できるので，開発コストの問題が大幅に軽減される上に，分散並列プログラミングの経験のない新規ユーザにとっても取り組みやすいものであると考えられる．

現在日本原子力研究所では，宇宙開発事業団および海洋科学技術センターと協力して平成 13 年度の完成を目標に超高速並列計算機「地球シミュレータ」[6] の開発を行なっており，HPF を地球シミュレータのプログラミングインタフェイスの一つとして採用し，強力に進める計画である．したがって，HPF を事前に評価しておくことは，地球シミュレータのユーザ開拓の一環として重要であると考えている．

しかしながら現状では HPF による分散並列プログ

ラミングの実績が少なく，プログラムの記述性や並列実行性能，またコンパイラに代表される処理系の使いやすさに大きく依存する分散並列化チューニング作業のしやすさなどいくつか懸念されることがある．

そこで，プラットフォームとして NEC 並列処理センタに設置されている並列コンピュータ Cenju-4[7] を用い，既存の 2 つの実アプリケーションプログラム MiFlow3D および Trans6 を HPF を用いて並列化し，その並列実行性能，HPF の記述性について評価した．また，HPF 処理系の使用感についても述べる．

2. HPF による MiFlow3D の並列化

2.1 MiFlow3D の概要

MiFlow3D は，3 次元非圧縮性ナビエ・ストークス方程式に基づく流体をシミュレーションするプログラムである[8]．空間の離散化は，移流項について 1 次風上差分，拡散項について中心差分が使用されている．時間積分は，time splitting method に基づく陽的差分法である．圧力に関するポアソン方程式は，中心差分による離散化の結果得られる連立一次方程式を解いて求める．連立一次方程式の解法として，SOR 法，ICCG 法などが選択できるようになっている．本研究では，2 次元 red-black 法に基づく解法について，HPF 化を行った．

計算に用いた問題は，正方形断面を持つ直方体の中のポアズイユ流れである．レイノルズ数を 100 とし，入口に一様流を与え，出口で流速のノイマン条件，壁面で滑りなしの境界条件を与えた．プログラムの動作は，逐次版においてすでに正常であることが確認されている．

2.2 主要ループとそこでアクセスする配列

このプログラムの主要な処理はポアソン方程式を解くサブルーチン `lsor4c` で，全実行時間の約 98 % を消費している．このサブルーチン内の典型的なループは以下のように，外側を `k` で，内側を `ip` でそれぞれ増分 2 で繰り返す二重ループになっている．

```
      do  200  k = 1, n, 2
        kp  =  k + 1

        do  210  ip = 1, lm, 2
          ix   =  ip + 1
          xtmp = ( b(ip,k)
     &             - coef(ip,1,k)*x(ix  , kp-1)
     &             - coef(ip,2,k)*x(ix-1, kp  )
     &             - coef(ip,3,k)*x(ix-1, kp  )
     &             - coef(ip,5,k)*x(ix+1, kp  )
     &             - coef(ip,6,k)*x(ix+1, kp  )
     &             - coef(ip,7,k)*x(ix  , kp+1) )
     &           / coef(ip,4,k)
          x(ix,kp) =  s1omg*x(ix,kp) + omega*xtmp
210     continue
        ........
200   continue
```

ここで使用する配列は `coef`, `b`, `x` の 3 つの仮配列で

```
      dimension coef(lm,7,n)
      dimension b(lm,n), x(lm+2*1,n+2)
```

と宣言されており，上のループの外側の DO 変数`k` (および `kp`) でアクセスするそれぞれの最終次元の大きさは `coef` と `b` は `n`, `x` は `n+2` となっている．

各々の最終次元にのみ着目すると上記のループにおける各配列要素へのアクセスは，表 1 のような定義・参照関係にあり，図示すると 図 1 のパターンになっている．

表 1　各配列要素へのアクセス (MiFlow3D)

配列	定義	参照
`coef`	なし	`k` 番目
`b`	なし	`k` 番目
`x`	`kp(=k+1)` 番目	`kp-1(=k )` 番目 `kp (=k+1)` 番目 `kp+1(=k+2)` 番目

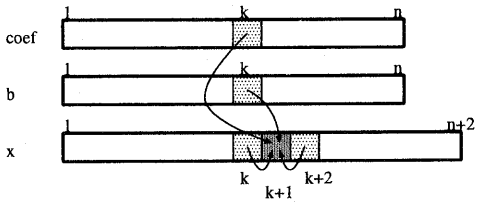

図 1　配列要素アクセスのイメージ (MiFlow3D)

2.3 HPF による分散並列化

2.3.1 データの分散

上で述べたように配列アクセスは最終次元で独立しているので，各配列を最終次元で BLOCK 分散することにし，最も大きな配列 `x` に対して 2 要素分小さな配列 `coef` と `b` については 1 要素ずれたアクセスを行なっているので，`coef` と `b` を `x` に対して 1 要素ずらして align することにした．データの分散指示のための HPF 指示行は以下の通りである．データ分散のイメージを 図 2 に示す．

```
!hpf$ align coef(*,*,i) with x(*,i+1)
!hpf$ align b(*,i)      with x(*,i+1)
!hpf$ distribute x(*,block) onto p
```

2.3.2 ループの並列化

上述の分散指示行を挿入してコンパイルしたところ，そのままではループの並列化可能性が判定できず，ループが並列化されないので，問題の `k` のループに次のような independent 指示行を追加した．

```
!hpf$ independent,new(kp,ip,ix,xtmp)
```

しかしこの指示行を挿入しても，`k` のループは増分 2 で繰り返すため`kp(=k+1)` も 2 ずつ増加するので，

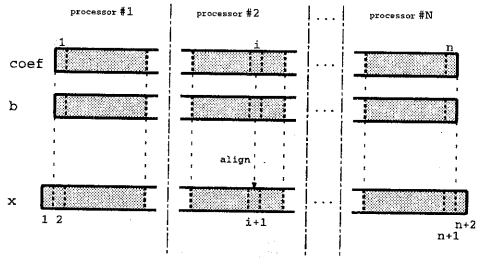

図 2 配列分散のイメージ (MiFlow3D)

`x(ix, kp-1)` および `x(ix, kp+1)` と `x(ix,kp)` とは重なる心配がなく安全に並列化できるはずなのに，コンパイラにはそれが判定できず，並列化できなかった．これについては，ループ中の `kp` を `k+1` に置き換えることでうまく並列化できた．

2.4 評価結果

2.4.1 記述性

オリジナルソース 1700 行に対して，追加した HPF 指示文 14 行 (内訳を 表 2 に示す)，追加したインタフェイスブロック 7 行，添え字の置き換えなどのために変更した行 42 行とわずかの改造で並列化することができた．また各々の追加した HPF 指示文についても内容的に特に難しいものは必要でなかった．

表 2 HPF 指示文の内訳 (MiFlow3D)

指示文	行数
processors 指示文	3
align 指示文	6
distribute 指示文	3
independent 指示文	2

2.4.2 性能

1 ～ 31 台のプロセッサを用いて実行時間を計測し，HPF で分散並列化したプログラムの実行時間と逐次プログラムの実行時間とを比較した．問題サイズは，プロセッサ 1 台で実行できることを前提に分散次元の配列サイズ `n` を 101 とした．計測結果を 表 3 および図 3 に示す．

HPF 化してプロセッサ 1 台で実行した場合の経過時間を逐次実行の場合と比較すると，HPF 化することによるオーバーヘッドが約 30 % あることがわかる．

1 ～ 7 台のプロセッサ台数が少ない間はほぼ線形に性能が向上しており，台数効果が十分に得られていることがわかる．8，9 台で加速率が大きく向上している点については現在調査中である．プロセッサ台数が多くなるにしたがって台数効果が得られなくなっているが，これは逐次実行との性能比較を念頭に置いて評価したために多数のプロセッサで並列実行するには問題が小さすぎるためであると考えられる．すなわち，分散次元の配列サイズ `n` が 101 しかなく 10 台以上のプロセッサで分散すると各々のプロセッサが担当する計算量が減少するためである．問題サイズを十分に大きくすれば台数効果も得られると考えている．

表 3 経過時間と加速率 (MiFlow3D)

台数	経過時間 (秒)	加速率 (逐次比)
逐次	534.456	—
1	733.530	0.729
2	372.040	1.437
3	253.753	2.106
4	178.204	2.999
5	143.779	3.717
6	119.249	4.482
7	103.882	5.145
8	69.106	7.734
9	67.851	7.877
10	84.748	6.306
11	76.131	7.020
12	74.072	7.215
13	67.397	7.930
14	69.912	7.645
15	68.165	7.841
16	70.621	7.568
17	67.204	7.953
18	61.503	8.690
19	63.615	8.401
20	64.569	8.277
21	64.366	8.303
22	65.699	8.135
23	66.508	8.036
24	65.240	8.192
25	67.553	7.912
26	61.034	8.757
27	62.706	8.523
28	63.131	8.466
29	64.395	8.300
30	65.522	8.157
31	65.197	8.198

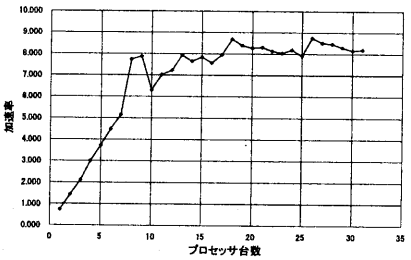

図 3 加速率のプロセッサ台数依存性 (MiFlow3D)

3. HPF による Trans6 の並列化

3.1 Trans6 の概要

Trans6 は，一様等方性乱流シミュレーションのためのプログラムであり，正方領域における 3 次元非圧縮性ナビエ・ストークス方程式をスペクトル法によって解いている[9)10)]．正方領域の境界で周期境界条件を課しているため，スペクトル法の直交系はフーリエ級数である．また，時間積分は 4 段 4 次ルンゲ・クッタ法を用いている．すべての変数はスペクトル空間で記述されるが，非線形部分の計算は convolution となるため，計算量削減の観点から実空間の積として計算する．このため，計算時間の大部分は，変数をスペク

トル空間と実空間との間でフーリエ変換する部分で使用される．フーリエ変換には Gentleman-Sande のアルゴリズムが用いられている[11]．

3.2 主要な処理とそこでアクセスする配列

このプログラムの主要な処理はフーリエ変換する部分で，逐次版において n が 64 の時は全実行時間の約 73 % を消費している．n が大きくなるとこの比率はさらに大きくなる．フーリエ変換とフーリエ逆変換について別々のサブルーチンが用意されており，特に 3 次元フーリエ変換では引数で渡した 3 次元配列を z 方向 (3 次元目)，y 方向 (2 次元目)，x 方向 (1 次元目) の順で変換している．

一例として x 方向 (1 次元目) の変換を行なう処理の一部を示すと次のようになっている．

```
      do  110  k = 1, n
        do  110  i = 1, nh
          i2  =  2*i
          i1  =  i2 - 1
          do  110  j = 1, n
            cr(i1,j,k) = fr(i,j,k) + fr(i+nh,j,k)
            ci(i1,j,k) = fi(i,j,k) + fi(i+nh,j,k)
            t1         = fr(i,j,k) - fr(i+nh,j,k)
            t2         = fi(i,j,k) - fi(i+nh,j,k)
            cr(i2,j,k) = t1*cn(i+ibase) + t2*sn(i+ibase)
            ci(i2,j,k) = t2*cn(i+ibase) - t1*sn(i+ibase)
110   continue

      ibase  =  nh*istage
      do  120  k = 1, n
        do  120  i = 1, nh
          i2  =  2*i
          i1  =  i2 - 1
          do  120  j = 1, n
            fr(i1,j,k) = cr(i,j,k) + cr(i+nh,j,k)
            fi(i1,j,k) = ci(i,j,k) + ci(i+nh,j,k)
            t1         = cr(i,j,k) - cr(i+nh,j,k)
            t2         = ci(i,j,k) - ci(i+nh,j,k)
            fr(i2,j,k) = t1*cn(i+ibase) + t2*sn(i+ibase)
            fi(i2,j,k) = t2*cn(i+ibase) - t1*sn(i+ibase)
120   continue
```

ここで，fr, fi, cr, および ci が引数として渡された 3 次元配列で，それぞれ

```
      dimension  fr(n,n,n), fi(n,n,n)
      dimension  cr(n,n,n), ci(n,n,n)
```

と宣言されている．

この例のように x 方向 (1 次元目) の変換を行なう際の各配列要素へのアクセスイメージは，図 4 に示したように他の 2 つの次元についてはデータの依存性がないので他の 2 つの次元のいずれかまたは両方について並列に実行可能である．その事情は y 方向 (2 次元目) および z 方向 (3 次元目) の変換についても同様で並列に実行可能である．

3.3 HPF による分散並列化

3.3.1 データの分散

上述のように，3 次元配列に対して 3 方向の変換を行なうので固定的な配列分散では全てをうまく並列化することはできない．そこで各々の配列について，基本的には z 方向 (3 次元目) に沿って BLOCK 分散することにし，z 方向の変換の前に分散次元を切替えて再分散し z 方向の変換が終った後で元の分散に戻す転置操作を行なうことにした．データの分散および再分散のイメージを 図 5 に示す．

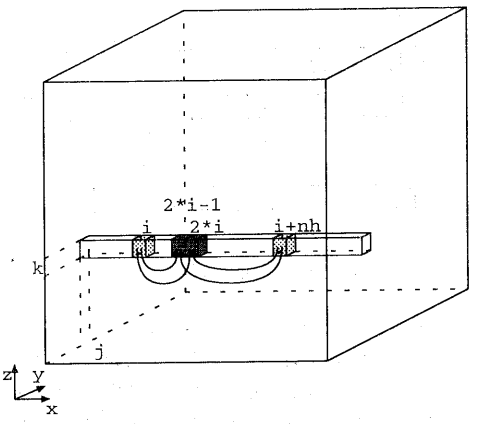

図 4 配列要素アクセスのイメージ (Trans6)

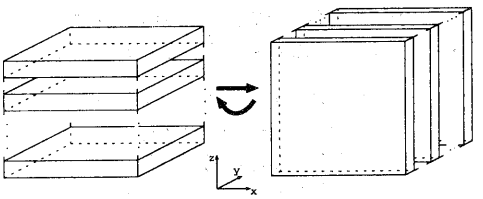

図 5 配列分散のイメージ (Trans6)

データの分散指示のための HPF 指示行の例は，

```
!hpf$ distribute (*,*,block) onto p
!hpf$&                      :: fr, fi, cr, ci
```

再分散指示のための HPF 指示行の例は，

```
!hpf$ dynamic                :: fr, fi, cr, ci
!hpf$ redistribute (*,block,*) onto p
!hpf$&                       :: fr, fi, cr, ci
```

である．同様に関連する配列についてもデータ分散のための HPF 指示行を追加した．

しかしながら，動的に再分散するために dynamic 宣言したことでコンパイラの解析能力が及ばないためか問題のループがうまく並列化できなかった．そこで z 方向の変換部分を新たにサブルーチン化しその呼び出し境界で再分散と復旧が行なわれるようにすることで，dynamic 宣言しないように変更した．

3.3.2 ループの並列化

関連する各々のループには適宜 independent 指示行を追加した．上述の 2 つの DO ループに対して挿入した independent 指示行は以下の通りである．

```
!hpf$ independent,new(i1,i2,j,t1,t2)
```

3.4 評価結果

3.4.1 記述性

このプログラムでは，多くの配列が擬寸法仮配列として受け渡しされていたが，HPF の言語仕様上の制約で擬寸法仮配列に対しては分散を指示することはできないので，まずこれらを整合配列に変更することが必要であった．

オリジナルソース 1400 行に対して，追加した HPF 指示文 114 行 (内訳を 表 4 に示す)，呼び出し境界で動的に再分散と復旧を行なうサブルーチン化のための改造 40 行，擬寸法仮配列を整合配列に変更するための改造 30 行で並列化することができた．この HPF 化においても複雑な HPF 指示文は必要でなかった．

しかしながら上述のように，redistribute 指示文を使った動的再分散が性能の面から利用できなかったことは残念である．

表 4 HPF 指示文の内訳 (Trans6)

指示文	行数
processors 指示文	25
distribute 指示文	43
independent 指示文	46

3.4.2 性能

1 ～ 31 台のプロセッサを用いて，プロセッサ 1 台で実行できる問題サイズ $64 \times 64 \times 64$ に対する実行時間を計測し，HPF で分散並列化したプログラムの実行時間と逐次プログラムの実行時間とを比較した．計測結果を 表 5 および図 6 に示す．

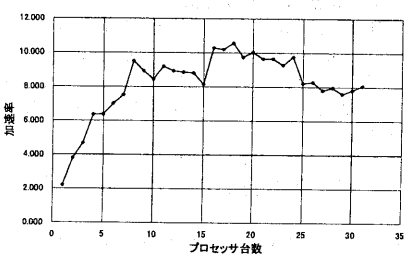

図 6 加速率のプロセッサ台数依存性 (Trans6)

HPF 化してプロセッサ 1 台で実行した場合の経過時間は逐次実行の経過時間よりも速いことがわかる．これは，ループ内で集中的にアクセスしている配列のデータマッピングが変わったため，逐次実行の際に頻繁に発生していたデータキャッシュミスが発生しなくなったためである．

表 5 経過時間と加速率 (Trans6)

台数	経過時間 (秒)	加速率 (逐次比)
逐次	721.680	—
台数	経過時間 (秒)	加速率 (逐次比)
1	326.922	2.207
2	190.285	3.793
3	154.140	4.682
4	113.628	6.351
5	113.460	6.361
6	103.016	7.005
7	95.920	7.524
8	75.855	9.514
9	80.823	8.929
10	85.402	8.450
11	78.390	9.206
12	80.796	8.932
13	81.426	8.863
14	81.803	8.822
15	88.570	8.148
16	70.173	10.284
17	70.776	10.197
18	68.273	10.570
19	73.955	9.758
20	71.894	10.038
21	74.755	9.654
22	74.750	9.655
23	77.650	9.294
24	73.881	9.768
25	87.788	8.221
26	87.002	8.295
27	92.428	7.808
28	90.564	7.969
29	94.961	7.600
30	92.239	7.824
31	89.333	8.079

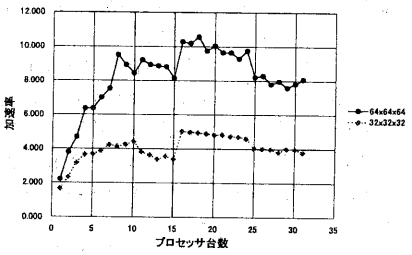

図 7 加速率に対する問題サイズの依存性 (Trans6)

MiFlow3D の場合と同様，1 ～ 7 台のプロセッサ台数が少ない間はほぼ線形に性能が向上しており，台数効果が十分に得られていることがわかる．プロセッサ台数が多くなるにしたがって台数効果が得られなくなっているが，これは逐次実行との性能比較を念頭に置いて評価したために，多数のプロセッサで並列実行するには問題が小さすぎるためであると考えられる．問題サイズを十分に大きくすれば台数効果も十分に得られるものと思われる．比較のため，問題サイズを $32 \times 32 \times 32$ として計測した結果を図 7 に示す．図から，問題サイズが大きいほど台数効果も大きくなっていることがわかる．プロセッサ台数が増えるにしたがって各々のプロセッサが分担する格子数が減少し，それにともなって計算時間が減少することにより実行

時間のうち並列化のオーバーヘッドが占める割合いが大きくなるために台数効果が得られなくなるので，問題サイズが大きくなれば加速率は増加する．これは一般の並列化について言えることである．

また，MiFlow3D および Trans6 のいずれについても経過時間の計測においては複数回の計測値のうちで最速値 (最も短時間で実行できた時の経過時間) を選んだが，各々の計測値にはかなりのばらつきが見られた．これは，マルチユーザ環境で実行したことによる計測値のばらつきであると思われる．

4. HPF 処理系の使用感

今回，Cenju-4 において HPF 処理系を使って分散並列化の評価を行なったが，その過程で感じた処理系の使用感についてここでまとめておく．

- ループ内に現れる配列要素参照の添字の書き方によって並列化できたりできなかったりする．ごく簡単なパターンと思われるものでも並列化可能性が判定できなくなる場合がある．たとえば MiFlow3D の例では，DO 変数 k について増分 2 で繰り返すループ内で常に k+1 が代入され，さらに内側のループ内で参照される変数 kp が使われているが，この kp を内側ループ内で配列参照の添字に使うと並列化可能性が判定できなくなった．コンパイラの解析能力の強化が必要である．
- コンパイル時にループ毎に HPF の並列化メッセージが出力されるがその内容がわかりにくい，あるいは不十分なため，意図通りに並列化できたかどうかがわかりにくい．またうまく並列化できなかった場合，何が問題となって並列化できなかったのかがわかりにくい．ユーザの手で並列化チューニングを進めるためには，特に並列化できなかった場合にその要因が容易にわかるメッセージが欲しい．
- この HPF 処理系は分散並列処理を MPI で実現しており，HPF コンパイラが生成する中間ソースを通常の Fortran コンパイラに入力する形で作られている．ユーザが見ることのできる中間ソースには MPI ルーチンの呼び出しが直接現れるわけではなく，処理系固有の実行時ルーチンの呼び出しが現れるため，それを見てもチューニングの助けにはならない．

5. ま と め

本稿では，分散並列処理言語として HPF を取り上げ，二つの実アプリケーションプログラムを使って HPF による分散並列化，チューニングを行ない実行性能等を評価した．

現状の HPF 処理系についてはユーザの立場から見て未熟な点もいくつか見受けられるが，分散並列処理言語としての HPF は記述性に代表される並列プログラムの生産性の面で大いに期待が持てると感じている．HPF 処理系の今後の改善に期待したい．

今後は，さらに評価を進めるとともに，MPI を用いて同一の指針で並列化した場合との性能および記述性の比較を行なっていきたいと考えている．

謝辞

Trans6 の HPF 化および性能評価に御協力いただいた NEC C&C メディア研究所の荒木拓也氏に感謝致します．また，並列化および性能評価において御討論いただいた地球シミュレータ研究開発センタのメンバ諸氏に感謝致します．

参 考 文 献

1) "MPI-2: Extensions to the Message-Passing Interface," Message Passing Interface Forum, July (1997).
2) "High Performance Fortran Language Specification Version 2.0," High Performance Fortran Forum, January (1997).
3) "High Performance Fortran2.0 公式マニュアル," High Performance Fortran Forum, シュプリンガー・フェアラーク東京 (1999).
4) "OpenMP Fortran Application Program Interface Version1.0," OpenMP Architecture Review Board, October (1997).
5) "OpenMP C and C++ Application Program Interface Version1.0," OpenMP Architecture Review Board, October (1998).
6) M. Yokokawa, et al., "Basic Design of the Earth Simulator," High Performance Computing, LNCS 1615, Springer, pp.269-280 (1999).
7) T. Nakata, et al., "Architecture and Software Environment of Parallel Computer Cenju-4," NEC RESEARCH & DEVELOPMENT Vol.39, No.4, pp.385-390, October (1998).
8) 市原 他, "非圧縮性 NS 方程式における圧力方程式の並列解法と評価," JAERI-Data/Code 96-012 (1996).
9) 横川，蕪木, "等方乱流シミュレーションコードの性能評価," 情処研報 Vol.46 No.7, pp.37-44 (1993).
10) M. Yokokawa, et al., "Parallerization of a Fourier Pseudospectral CFD Code," PERMEAN '95, pp.54-59 (1995).
11) C.V.Loan, "Computational Frameworks for the Fast Fourier Transform," SIAM (1992).