地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価

村井 均[†,☆] 岡部 寿男[††]

地球シミュレータでは，並列化の手段として High Performance Fortran （HPF）を利用できる．我々は，HPF による NAS Parallel Benchmarks（NPB）の最適な実装を求め，地球シミュレータ上で評価を行った．その結果，HPF の基本仕様に加え，公認拡張と HPF/JA 言語仕様の機能を利用することで，CG と MG については MPI 版（NPB2.4）と同等の，LU についても近い性能が得られた．このことから，HPF の機能をフルに活用すれば，MPI に匹敵する性能を得ることが可能であることを確認できた．さらに，MPI 版の実装との詳細な比較から，より効率的な並列化をより容易に行うために必要な言語仕様や不足している機能の考察も行った．

Implementation and Evaluation of NAS Parallel Benchmarks with HPF on the Earth Simulator

Hitoshi Murai[†,☆] and Yasuo Okabe[††]

High Performance Fortran (HPF) is provided for parallelizing your programs on the Earth Simulator (ES). We developed an optimal implementation of NAS Parallel Benchmarks (NPB) with HPF and evaluated it on the ES. The result showed that the HPF implementation could achieve performance comparable to the MPI (NPB2.4) for the programs of CG and MG, and nearly close for LU. It can be said that a good HPF program written by fully exploiting the HPF features could be equal to the MPI in performance. We also studied the language specifications and features of HPF required for easier and more efficient parallelization from the detailed comparison of the HPF implementation with the MPI.

1. はじめに

分散メモリ型並列計算機上のプログラミングモデルとして現在広く使われている Message Passing Interface（MPI）に対して，データ並列言語 High Performance Fortran（HPF）[1] の利点はプログラミングの容易さにある．コンパイラ技術の未成熟から従来の HPF コンパイラでは性能の点で MPI に及ばなかったが，最新のコンパイラでは MPI に並ぶ性能を達成できるようになっている[2]．

世界最高速の並列計算機である地球シミュレータ（*ES: Earth Simulator*）[3] 上でも，*HPF/ES* と呼ばれる HPF コンパイラを利用できる．HPF の標準仕様とともに，HPF/JA 言語仕様[4] や HPF/ES が提供する種々の独自機能[5] を用いて，ES の性能を効果的に引き出すプログラムを容易に作成することができる[6]．

NAS Parallel Benchmarks（NPB）は，NASA Ames Research Center で開発された，並列計算機のためのベンチマークであり，5 つのカーネルコード（EP, MG, CG, FT, IS）と 3 つのアプリケーションコード（BT, SP, LU）から成る[7]．NPB については既に多くの研究[8)~10)] が行われており，さらに HPF による並列化についても NASA Ames Research Center 自身によるものを含めていくつかの研究がある[11),12)]．しかし，最新のコンパイラや言語仕様を用いた評価はあまり行われていない[2),13)]．

Frunkrin らは，NPB3.0 alpha において，EP と IS を除く 6 つのベンチマークに対して HPF による実装を与えた[11]．このうち FT は ES 上でも良好な性能を示したものの，他の 5 つについては，NPB2.4 の MPI による実装に比べたとき，実行時間とメモリサイズのいずれかまたは両方で大きく見劣りする．また，西谷

† 海洋科学技術センター 地球シミュレータセンター
The Earth Simulator Center, Japan Marine Science and Technology Center
☆ 現在，海洋研究開発機構 地球シミュレータセンター
Presently with The Earth Simulator Center, Japan Agency for Marine-Earth Science and Technology
†† 京都大学 学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University

らは，NPB の全てのベンチマークに対して HPF による実装を与えたが，一部のベンチマークでは MPI 版とは異なる，最適ではない並列化方法（データの分散, 計算の分割, 通信の生成）を採用したことに加え，HPF コンパイラに制約があったことから，MPI 版の 1.5〜1.8 倍程度の実行時間を要する結果となっている[12]．

これに対し，本研究は，HPF の機能をフルに活用した最適な実装と最新のコンパイラによる評価を通じて HPF の有用性と潜在能力を示すとともに，その作業を通じて HPF による並列化のノウハウを蓄積することを目的とする．また，MPI 版との比較から，HPF に求められる言語仕様や HPF/ES に不足する機能を抽出・提案することも行う．

NPB に含まれる 8 つのベンチマークのうち，容易に並列化できることが自明である EP，既に NPB3.0 alpha において MPI に匹敵する実装が与えられている FT，Fortran による実装が与えられていない IS を除く 5 つの実装と評価を行ったが，紙面の都合から本稿では LU, CG, MG の 3 つについてのみ述べる．

以下，2 章で HPF/ES による LU, CG, MG の実装についてそれぞれ述べ，3 章で評価を行う．4 章で実装と評価の結果について考察を行った後，5 章では本稿を総括する．

2. 実　　装

HPF/ES により NPB を並列化するに当たって，次の基本方針に従って作業を進めた．

- シリアル版（NPB2.3-serial）をベースとする．
- 基本的には指示文の追加のみを行うが，必要に応じてソースの書き換えも行う．
- 可能な限り，MPI 版（NPB2.4）と同じ並列化方法とアルゴリズムを用いる．
- 一部を除いて一次元 BLOCK 分散を用いる．

以下，LU，CG，MG の各ベンチマークの実装について順に述べる．

2.1 ベンチマーク LU

LU は，5×5 ブロック上下三角方程式を対称 SOR 法で解くベンチマークである．各 z 平面において “wavefront” スタイルの計算を行うため，小さいメッセージを多数回送受信する細粒度通信の性能が評価される．

MPI 版では，x と y の二つの次元を BLOCK 分散しているが，本研究では基本方針に従って y 次元のみの BLOCK 分散を用いた．

```
!HPF$ template t(0:isiz2+1)
!HPF$ distribute (block) :: t
!HPF$ align (*,*,i,*) with t(i) :: u, rsd, ...
!HPF$ shadow (0,0,2,0) :: u, rsd
```

2.1.1 パイプライン実行

文献 12) で用いられている HPF コンパイラは，DOACROSS 型ループをパイプライン並列化する機能を持つ．しかし，HPF/ES はこの機能を持たないため，下に述べる方法で擬似的にパイプライン実行を実現した．

下は LU の主要な処理を行うループの一つである．`DO k` ループの繰り返し k において，手続き `jacld` と `blts` は $z = k$ 平面に対する処理を行う．この “wavefront” 計算は，x 軸方向と y 軸方向に対する DOACROSS 型ループとなるため，HPF/ES では並列化できない．

```
      DO k = 2, nz-1
         call jacld(k)
         call blts( isiz1, isiz2, isiz3,
     >              nx, ny, nz, k,
     >              omega,
     >              rsd,
     >              a, b, c, d,
     >              ist, iend, jst, jend,
     >              nx0, ny0, ipt, jpt)
       END DO
```

そこで，繰り返し k において，プロセッサ P_p が $z = k - p$ 平面に対して `jacld` と `blts` を呼ぶようにコードを修正した．例えば繰り返し $k = 4$ において，プロセッサ P_0 は $z = 4$ 平面の，P_1 は $z = 3$ の，P_2 は $z = 2$ の，P_3 は $z = 1$ の，それぞれ保持する領域を独立に処理する．これにより，DOACROSS 型ループを擬似的にパイプライン実行することができる．プロセッサ間の同期は，`DO k` ループ内の `REFLECT` 指示文の実行によって実現される（**図 1**）．

```
      np = number_of_processors()
      me = my_processor()
      DO k = 2, nz-1+np-1
!HPFJ    reflect rsd
         kk = k - me
         if (kk < 2 .or. kk > nz-1) cycle
         call jacld(kk)
         call blts( isiz1, isiz2, isiz3,
     >              nx, ny, nz, kk,
     >              omega,
     >              rsd,
     >              a, b, c, d,
     >              ist, iend, jst, jend,
     >              nx0, ny0 )
      END DO
```

2.1.2 シャドウ部の重複計算

シャドウとは，差分法に基づくプログラムにおいて，隣接プロセッサ間で近傍要素の通信（シフト通信）を行う際に用いるバッファ領域のことをいう．手続き `erhs` と `rhs` には，各プロセッサがシャドウ部を重複して計算することで性能を向上させられる箇所がある．

図 2(a) に示すように，近傍計算により配列 `A`→ 配列 `B`→ 配列 `C` が順に更新される場合，HPF/ES コンパイラは配列 `A` と `B` の各々に対して幅 1 のシフト通信を一回ずつ，計二回の通信を生成する．ここで，図 2(b) のように，配列 `A` に対し幅 2 のシフト通信を実行した

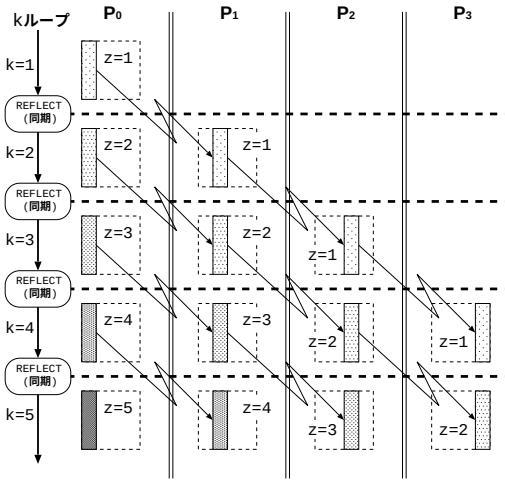
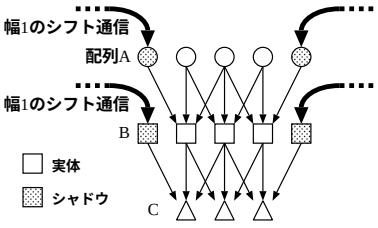
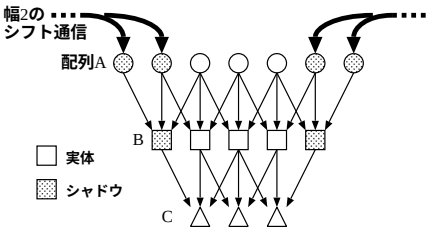


図 1 LU の擬似パイプライン実行
Fig. 1 Pseudo-pipelined Execution in LU



(a) オーナコンピュートルールに従う実行



(b) シャドウ部の重複計算を行う実行

図 2 シャドウ部の重複計算
Fig. 2 Overlapped Execution of Shadow Region

上で，各プロセッサが配列 B のシャドウ部も計算・更新するようにすれば，通信の回数を一回に減らすことができる．このとき，総通信量は変わらず，一つのプロセッサが実行すべき計算の量は増える．

一般に，配列の n 要素を更新するのに要する時間は，プロセッサ間で n 要素を通信する際のレイテンシよりもはるかに小さいため，通信の回数を減らすことにより性能を改善できる．

HPF では，HPF/JA 言語仕様[4] の EXT_HOME 節を指定すれば，シャドウ部の重複計算を行うことができる[4]．しかし，現在の HPF/ES は EXT_HOME 節をサポートしていないため，HPF/ES が出力する中間プログラムに修正を加えることで等価な処理を実現した．

2.1.3 部分的 REFLECT

本実装では，ループ内で通信が繰り返し実行されるのを避け，パイプライン実行を実現するために，ループおよび手続きから通信を括り出しておく．そのために，HPF/ES コンパイラの機能によってシフト通信を自動的に生成するのではなく，HPF/JA 言語仕様の REFLECT 指示文を指定して明示的に通信を指定する．ここで，HPF/ES コンパイラが自動的に生成するシフト通信では，配列に指定されたシャドウ領域のうち実際にアクセスされる部分のみが更新されるのに対して，REFLECT 指示文では，対象の配列が持つ**全ての**シャドウ領域が更新されると HPF/JA 言語仕様は規定している．すなわち，配列に指定されたシャドウ領域の一部のみを必要とする場面において REFLECT を使うと，本来は不要である通信が発生することになる．この問題を避けるためには，例えば文献 15) で提案されている「部分的 REFLECT」のような機能が必要になる．本研究では，ソースプログラムを修正して，部分的 REFLECT にほぼ等価な処理を実現した．

2.2 ベンチマーク CG

CG は，正値対称な疎行列の最小固有値を共役勾配法により求めるベンチマークである．疎行列を格納する方法として標準的な Compressed Row Storage（CRS）形式[16] で格納された疎行列に対する，行列ベクトル積の性能が評価される．NPB2.3-serial において，行列ベクトル積を実行するコードを下に示す．

```
      do i=1, N
       do j=rowstr(i), rowstr(i+1)-1
        c(i) = c(i) + aa(j)*b(colidx(j))
       end do
      end do
```

CG を並列化する際のポイントは，CRS 形式を各プロセッサ上へ分散する方法である．CRS 形式は，対象の疎行列の非零要素の値を行優先の順序で保持する aa，対応する aa の要素の列インデックスを保持する colidx，各行に属する最初の aa の要素の番号を保持する rowstr の 3 つの一次元配列から構成される．従来手法[11),12)] では，配列 aa と colidx の次元を拡張して，疎行列の一次元目に対応付けることを行っているが，この方法はソースプログラムの修正を伴う上に，3.4 章で述べるようにメモリの使用効率が悪い．これに対し，次の二つの方法を新しく考案し，実装を行った．

2.2.1 実装 1: GEN_BLOCK 分散

CRS 形式を構成する配列 rowstr には通常の BLOCK 分散を適用する．配列 aa は，元の疎行列において第 i 行に属していた要素が rowstr(i) と同

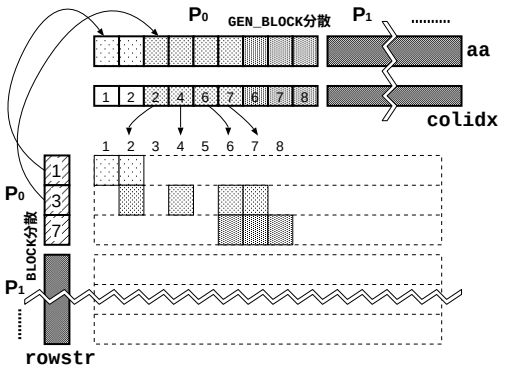


図 3 GEN_BLOCK 分散による CG の実装
Fig. 3 Implementation with GEN_BLOCK distribution

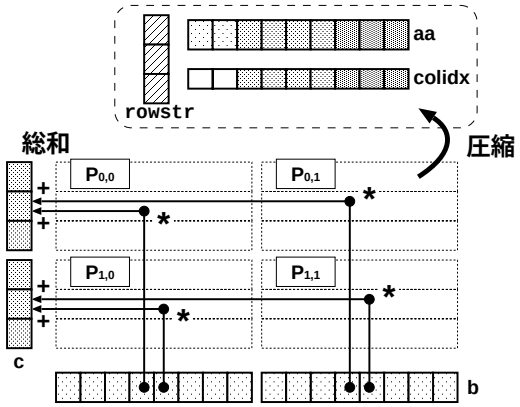


図 4 二次元分散による CG の実装
Fig. 4 Implementation with 2-dimensional distribution

じプロセッサに配置されるように GEN_BLOCK 分散する．同じく，配列 colidx も aa と対応するように GEN_BLOCK 分散する．例えば，**図 3** において，aa(4) は元の疎行列の第 2 行に属する．したがって，aa(4) が，rowstr(2) を保持するプロセッサ P_0 に配置されるように GEN_BLOCK 分散のマッピング配列を設定する．

```
      integer :: map(NP) = (/9, .../)
!HPF$ distribute (block) :: c, rowstr
!HPF$ distribute (gen_block(map)) :: aa, colidx
```

この実装では，全く通信を伴わずに行列ベクトル積を実行することができる．

2.2.2 実装 2: 二次元分散

MPI 版（NPB2.4）の実装で使われている手法である[9),17)]．二次元のプロセッサ配列上へ元の疎行列を分散したものとして，各プロセッサは自分に割り当てられた領域に属する要素のみを CRS 形式で保持する．このとき，配列 b と c は，それぞれ疎行列の二次元目と一次元目へ整列するように分散する．CRS 形式を構成する aa，rowstr，colidx の 3 つの配列は**ローカルデータ**として，あらかじめ分散後のサイズで宣言しておき，分散は行わない．それらのローカルデータの初期化はローカル手続きで行う．

```
!HPF$ template t(N,N)
!HPF$ distribute t(block,block)
!HPF$ align b(j) with t(*,j)
!HPF$ align c(i) with t(i,*)
```

この実装では，配列 c に対する総和処理を除いて通信を必要とせずに行列ベクトル積を実行することができる．

2.3 ベンチマーク MG

MG は，三次元ポアソン方程式を，簡略化したマルチグリッド法で解くベンチマークである．MG を並列化する際のポイントは，次の二点である[10)]．

- 階層実行を実現する手続き呼び出し
- 階層データの分散

2.3.1 従 来 手 法

- MPI 版（NPB2.4）の方法
  あらかじめ割り付けた一次元配列の要素を実引数として渡し，手続き側ではこれを三次元配列の仮引数として受け取り，渡された要素を起点とする領域を階層データとして利用する．HPF の言語仕様では，分散された実引数と仮引数の形状が異なることは許されないため，そのままでは実装できない．

```
      call psinv(u(ir(k)),m1(k),m2(k),m3(k))
      ...
      subroutine psinv(u,n1,n2,n3)
      doube precision u(n1,n2,n3)
```

- 西谷らの方法[12)]
  階層データを別々に宣言し，IF 文で階層毎に切り分けた手続き呼び出しの引数として明示的に渡す．階層が増えると，それに伴ってデータの宣言や手続き呼び出しを追加する必要があるため，プログラムの修正量が大きくなってしまう．

```
      double precision u1(n1,n1,n1), u2(n2,n2,n2)
     >                 u3(n3,n3,n3)
!HPF$ distribute (*,*,block) :: u1, u2, u3
      ...
      if (k == 1)      call psinv(u1,n1,...)
      else if (k == 2) call psinv(u2,n2,...)
      else if (k == 3) call psinv(u3,n3,...)
```

- NPB3.0 の方法[11)]
  階層データに階層（レベル）を表す次元を付加する．無駄な（実際には使用されない）領域が大きく，階層データ間のやり取りにおいて本来は不要なはずの通信が発生してしまう．

```
      double precision u(n1,n2,n3,nlevel)
!HPF$ distribute (*,*,block,*)
```

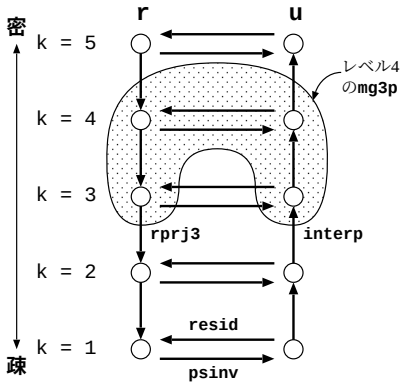


図 5 MG のデータフロー
Fig. 5 Dataflow in MG

2.3.2 提 案 手 法

本稿で提案する手法は，次の 2 つを特徴とする．

- 再帰呼び出しによる階層実行
- 階層データの階層的整列

この各々について順に説明する．

2.3.2.1 再帰呼び出しによる階層実行

MG の階層実行において，配列 r と u のデータフローは**図 5** のようになっている．

これらのデータフローは，元のプログラムでは，下に示すような DO ループによって手続き呼び出しを繰り返すことで実現されている．

```
      subroutine mg3P(u,v,r, ..., k)

      do  k = lt, 2, -1
         j = k-1
         call rprj3(r(ir(k)), ..., r(ir(j)), ..., k)
      enddo

      call psinv(r(ir(1)),u(ir(1)), ..., 1)

      do  k = 2, lt-1
         j = k-1
         call interp(u(ir(j)), ..., u(ir(k)), ..., k)
         call resid(u(ir(k)),r(ir(k)), ..., k)
         call psinv(r(ir(k)),u(ir(k)), ..., k)
      enddo

      end
```

これに対し，あるレベルの処理（図 5 の網掛け部分）だけを行うように手続き mg3P を書き直し，これを再帰的に呼ぶことによって階層実行を実現する．

```
      recursive subroutine mg3P(u,v,r, ..., k)

      call rprj3(r, ..., r2, ..., k)

      if (k > 2) then
         call mg3p(u2,v,r2, ..., k-1)
      else
        call psinv(r,u, ..., 1)
      end if

      call interp(u2, ..., u, ..., k)
      call resid(u,r,r, ..., k)
      call psinv(r,u, ..., k)

      end
```

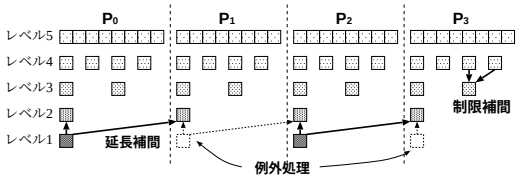


図 6 MG の階層データの整列（簡単のため，一次元配列で表記）
Fig. 6 Alignment of Hierarchical Data in MG
(For simplicity, 1-dimensional arrays are used.)

2.3.2.2 階層データの階層的整列

階層データ (手続き mg3P の引数である配列 u および r など) が，**図 6** のように整列して分散されるとき，階層間のデータのやり取り (制限補間と延長補間) において通信は発生しない．この分散を実現するためには，各階層データを，最も密なデータに対して，レベルに応じたストライドで整列させる必要がある．

手続き mg3P は，マルチグリッド実行のレベルを引数 k として受け取る．したがって，引数 u および r に対し，下のようにストライド 2^{lt-k} の整列を指定することで，通信が発生しない最適な分散を実現できる．

```
!HPF$ template t(N)
!HPF$ distribute (block) :: t

      double precision u(n), r(n)
!HPF$ align (i) with *t(i*(2**(lt-k))-m) :: u, r

      double precision, allocatable :: u2(:), r2(:)
!HPF$ align (i) with t(i*(2**(lt-k+1))-m2) :: u2, r2
```

さらに，再帰的に呼び出されるレベル $k-1$ の mg3P へ引数として渡す，レベル $k-1$ の階層データ (上図の u2 および r2) を割付けるとともに，ストライド 2^{lt-k+1} の整列を指定しておく．

この方法の利点は，手続き mg3P が，k と $k-1$ の 2 つのレベルの階層データのみを扱う点である．すなわち，西谷らの方法のような階層ごとのデータの宣言や手続き呼び出しは不要となる．また，階層データの割付けと分散が正しく行われることによって，NPB3.0 の方法のような無駄な領域の使用や不要な通信も起こらない．

手続きの仮引数をパラメータとする複雑な整列や分散は，HPF の言語仕様では許されているものの，従来の HPF コンパイラでは正しく扱えない場合が多かった[10),13),14)]．HPF/ES は，HPF の言語仕様に従った実装により，そのような整列や分散も正しく扱うことができる．

3. 評　　価

3.1 評 価 対 象

主に次の 3 種類のコードを評価し，ES 上で比較を

行った．問題のサイズはクラス C である．

HPF/ES 版　本研究で並列化したコード

MPI 版　NPB2.4 のコード

NPB3.0-HPF 版　NPB3.0alpha に，ES 上で実行するのに必要最小限の修正（バグ修正など）だけを施したコード

3.2 評価環境

ES は，計算ノード 640 台をクロスバネットワーク（バンド幅 12.3GB/s（双方向））で結合した分散メモリ型並列計算機システムである．各計算ノードは，16GB のメモリを共有する 8 台のベクトル型計算プロセッサ（ピーク性能 8GFLOPS）から成る．システム全体のピーク性能は 40TFLOPS に達する[3]．

評価に用いたソフトウェア環境は以下の通りである．

- `HPF/ES`
  `Rev.1.9.4(776) 2003/02/21`
- `FORTRAN90/ES Version 2.0`
  `Rev.269 ES 13 2003/05/14`
- `MPI/ES: command Version 7.0.0 (13,`
  `February 2003)`
- `MPI/ES: daemon Version 7.0.0 (18,`
  `February 2003)`
- `ESOS Release 1.1`

HPF/ES は，HPF プログラムを，MPI によって並列化された等価な中間 Fortran プログラムへ一旦トランスレートしてから，ES 上の通常の Fortran コンパイラと MPI ライブラリによってコンパイルおよびリンクを行う．HPF/ES に指定したコンパイル・オプション（性能に影響するもののみ）を以下に挙げる．

- `-C hopt`
- `-Wf"-pvctl vwork=stack"`
- `-Mnomapnew`
- `-Mscalarnew`
- `-Mnoerrline`
- `-Mnoentry`

各ベンチマークは ES の大規模（L 系）バッチジョブとして実行した．並列実行の方式として，HPF プロセッサを計算プロセッサに割り付け，ES 全体を単階層のフラットなシステムとして見る**フラット並列化**を用いた．また，HPF プロセッサの数が $1 \leq n \leq 8$ の場合には一台の計算ノードのみに n 個の HPF プロセッサを配置し，$n > 8$ の場合には $n/8$ 台の計算ノードに 8 個ずつの HPF プロセッサを配置して実行を行った．したがって，HPF プロセッサの数が 8 以下の場合には，結合ネットワークを介する計算ノード間通信は発生しない．

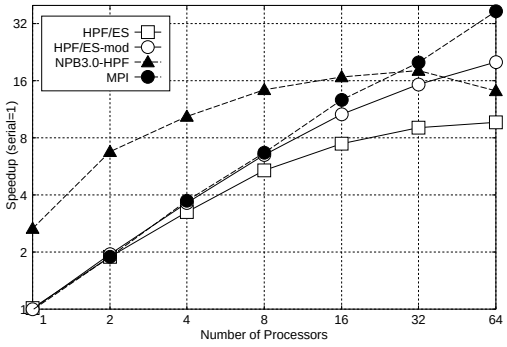


図 7　LU の評価結果
Fig. 7　Evaluation Result of LU

3.3 LU

2.1.2 章で述べたように，中間プログラムやソースプログラムの修正により，シャドウ部の重複実行と部分的 REFLECT に等価な処理を実現した．

図 7 にベンチマーク LU の評価の結果を示す．「HPF/ES」はパイプライン実行だけを実装した場合の，「HPF/ES-mod」はさらにシャドウ部の重複実行と部分的 REFLECT を実現した場合の結果である．グラフの縦軸はシリアル版 (NPB2.3-serial) の性能を 1 とする相対性能である．

HPF/ES はプロセッサ数が大きくなったときに，MPI 版に比べると性能の低下が大きい．シャドウ部の重複実行と部分的 REFLECT によって HPF/ES-mod ではかなりの改善が見られるが，なお MPI 版には及んでいない．これは，高並列時には，擬似パイプライン実行の同期処理 (REFLECT) のオーバヘッドが大きくなるためであると考えられる．

なお，NPB3.0-HPF 版は並列化の方法が他とは異なるため，かなり違った特性を示している．

3.4 CG

図 8 にベンチマーク CG の評価の結果を示す．「HPF/ES-gen」は実装 1（GEN_BLOCK 分散）を，「HPF/ES-2D」は実装 2（二次元分散）を示す．グラフの縦軸はシリアル版 (NPB2.3-serial) の性能を 1 とする相対性能である．横軸の括弧内に示した数字は，HPF/ES-2D と MPI 版で用いた二次元プロセッサ配列の構成を示す．

HPF/ES-2D と MPI 版は同等の性能を達成している．これらの性能がプロセッサ数について階段状になるのは，疎行列の二次元目の方向に発生する総和処理がオーバヘッドとなり，分散した 2 つの次元のうち一方の並列度（プロセッサ数）に対するリニアな速度向上が阻害されるためである．

NPB/ES-gen と NPB3.0-HPF 版は，1〜4CPU で

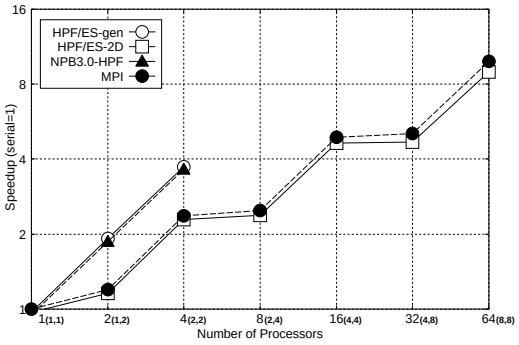


図 8 CG の評価結果
Fig. 8 Evaluation Result of CG

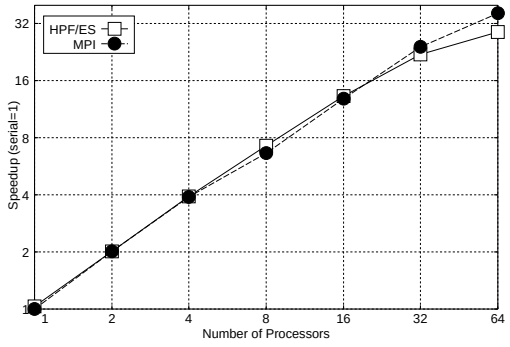


図 9 MG の評価結果
Fig. 9 Evaluation Result of MG

は，HPF/ES-2D と MPI 版を上回る性能を示すものの，8CPU 以上ではメモリ不足のため実行できなかった．このメモリの不足は，ベンチマークの測定対象である行列ベクトル積の部分ではなく，疎行列の初期化の部分で生じている．これらの実装では，初期化部において一時的に全てのプロセッサが疎行列の全要素を保持する必要があるためである．

3.5 MG

図 9 にベンチマーク MG の評価の結果を示す．グラフの縦軸はシリアル版 (NPB2.3-serial) の性能を 1 とする相対性能である．HPF/ES は MPI 版とほぼ同等の性能を示している．

高並列（32〜64CPU）時に，HPF/ES の効率がやや低下するのは，グリッドが疎になったときの例外的な処理による．すなわち，グリッド数がプロセッサ数を下回った場合，手続き `interp` の延長補間処理において，グリッドを保持しないプロセッサは例外的な処理を行う必要があり，これが若干のオーバヘッドを伴う（図 6）．MPI 版ではこの例外処理をうまく記述できるため効率は低下しない．

また，2.3 章でも述べたように，NPB3.0 の方法は無駄なメモリを大量に使用するため実行できなかった．

4. 考　　察

4.1 配列の分散

2 章の冒頭で述べたように，本研究では基本的に一次元 BLOCK 分散を使用している．これは，プログラムを並列化するという観点からは，多くの場合で一次元の分散が必要十分であり，かつ多次元分散よりも一次元分散が，他の分散フォーマットよりも BLOCK 分散が一般に性能的に優れているという理由による．

一方で，ES のような大規模な並列計算機上では，例えば本研究における MG のように，プロセッサ数が分散対象次元のグリッド数を上回ってしまうということも起こり得るが，そのような場合には二次元以上の分散を使用せざるを得ない．また，CG の実装では，GEN_BLOCK 分散や二次元分散の使用が，高効率の並列化を実現する決め手となった．さらに，本稿では取り上げなかったが，BT と SP を並列化するには，マルチパーティションと呼ばれる特殊な二次元 BLOCK 分散が有効であることが報告されている[15]．

以上のように，実際には，一次元 BLOCK 分散以外の分散方法を使用したい場面は少なくなく，HPF/ES でもそのような分散方法をさらに効率良く扱えることが重要となろう．

4.2 HPF/JA 言語仕様

本稿では，HPF/JA 言語仕様に含まれる機能の使用について特に詳しくは述べていない．しかし，どのベンチマークでも，`LOCAL` 節や `REFLECT` 指示文を始めとする HPF/JA 言語仕様の機能を使用している．HPF/JA 言語仕様をサポートする HPF コンパイラを利用する場合，まず配列に分散を指定してコンパイルを行ってみた後，並列化されなかったループに `INDEPENDENT` 指示文を，不要な通信が生成された箇所に `LOCAL` 節を，シフト通信が必要になる箇所に `REFLECT` 指示文を，それぞれ指定するのが標準的なプログラミングスタイルとなる．

4.3 HPF/ES の機能

本研究より，特に LU を効率的に並列化するために以下の機能が HPF/ES に不足していることが明らかになった．

(a) DOACROSS 型ループのパイプライン実行
(b) シャドウ部の重複計算（`EXT_HOME` 節）
(c) 部分的 REFLECT

現在の HPF/ES は，DOACROSS 型ループを含む「並列化できない（INDEPENDENT でない）」ループをうまく処理することができず，これが HPF/ES

によって並列化を行う際の大きなネックになっている．したがって，(a) の機能は，HPF/ES の適用範囲をさらに広めるためにも非常に重要であると考えられる．また，評価の結果は，(b) と (c) の機能を利用できれば，LU において最大で 2 倍程度性能が向上することを示している．(b) の機能 (`EXT_HOME` 節) は，HPF/JA 言語仕様に含まれていることからも実装が望まれる．

5. おわりに

HPF/ES による NAS Parallel Benchmarks の最適な実装を求め，ES 上で評価を行った．その結果，CG と MG では MPI 版と同等の性能を得られることを確認できた．LU でも，HPF/ES に不足している機能を使用できれば，MPI 版に近い性能を得られることがわかった．このことから，HPF の機能をフルに活用してプログラムを作成するとともに優れたコンパイラを利用すれば，HPF で MPI に匹敵する性能を得ることは可能であることがわかる．また，HPF/ES に不足する機能について検討を行った．

今後の課題として，4 章で検討した機能を HPF/ES 上で実現すること，PC クラスタなどの ES 以外のプラットフォーム上でも評価を行うこと，HPF の普及を図るためにもさらに多くのアプリケーションやベンチマークを HPF により並列化し，得られたノウハウをユーザに広く公開していくこと，などが挙げられる．

なお，本研究で用いた HPF コードは，HPF 推進協議会ホームページ (`http://www.hpfpc.org/`) にて公開されている．

謝辞 本研究は，平成 15 年度地球シミュレータ共同プロジェクト「並列処理言語 HPF （High Performance Fortran）を用いた大規模並列実行の性能検証および新規機能の検討」の一環として行った．

参考文献

1) High Performance Fortran Forum: *High Performance Fortran Language Specification* (1997).
2) Murai, H. et al.: Implementation and Evaluation of HPF/SX V2, *Concurrency and Computation – Practice & Experience*, Vol.14, No.8–9, Wiley, pp. 603–629 (2002).
3) Habata, S., Yokokawa, M. and Kitawaki, S.: The Earth Simulator System, *NEC Research & Development*, Vol. 44, No. 1, pp. 21–26 (2003).
4) JAHPF (Japan Association for High Performance Fortran), HPF/JA 言語仕様書 (1999).
5) 村井均ほか: 並列化コンパイラ HPF/ES の不規則問題向け機能, 情報処理学会研究報告 2002-HPC-90, pp. 61-66 (2002).
6) Sakagami, H., Murai, H., Seo, Y. and Yokokawa, M.: 14.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator, *Proc. SC2002* (2002).
7) Bailey, D. et al.: THE NAS PARALLEL BENCHMARKS, *Technical Report NAS-94-007*, Nasa Ames Research Center (1994).
8) Agarwal, R.C. et al.: High-performance parallel implementations of the NAS kernel benchmarks on the IBM SP2, *IBM Systems Journal*, Vol. 34, No. 2, pp. 263–272 (1995).
9) 板倉憲一ほか: 超並列計算機 CP–PACS における NPB Kernel CG の評価, 情報処理学会論文誌, Vol. 39, No. 06, pp. 1757–1765 (1998).
10) Chamberlain, B.L., Deits, S.J. and Snyder, L.: A Comparative Study of the NAS MG Benchmark across Parallel Languages and Architectures, *In proc. of SC2000* (2000).
11) Frumkin, M., Jin, H. and Yan, J.: Implementation of NAS parallel benchmarks in high peroformance fortran, *Technical Report NAS-98-009*, Nasa Ames Research Center (1998).
12) Nishitani, Y. et al.: Techiniques for compiling and implementing all NAS parallel benchmarks in HPF, *Concurrency and Computation – Practice & Experience*, Vol. 14, No. 8–9, Wiley, pp. 769–787 (2002).
13) Asaoka, K., Hirano, A. and Kanazawa, M.: Evaluation of the HPF/JA Extensions on Fujitsu VPP Using the NAS Parallel Benchmarks, *Proc. 4th International Symposium on HPC (ISHPC2002)*, LNCS2327, Springer, pp. 503-514 (2002).
14) Brandes, T.: ADAPTOR HPF Language Reference Manual Version 9.0, The Fraunhofer Institute for Algorithms and Scientific Computing (2003).
15) Chavarria-Miranda, D. and Mellor-Crummey, J.: An Evaluation of Data-Parallel Compiler Support for Line-Sweep Applications, *Journal of Instruction Level Parallelism*, Vol. 5, (2003).
16) Barrett, R. et al.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 2nd Edition (1994).
17) Lewis, J.G. and van de Geijn, R.A.: Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithm, *Proc. SC'93*, pp. 484–492 (1993).