NAS Parallel BenchmarksによるHPFの評価

村井 均[†] 岡部寿男[††]

最近，High Performance Fortran（HPF）に代表される並列言語に対する関心が再び高まりつつある．従来，性能の点でHPFはMPIに及ばなかったが，最近では，最新のコンパイラを使ってMPIに並ぶ性能を達成する例も出てきている．本研究では，HPFの有効性と能力を評価するために，NAS Parallel BenchmarksのLU，CG，MGについて，(1) 基本的には逐次Fortranプログラムに HPF指示文を追加することだけを行うという方針の下，HPFの機能を最大限に活用して実装し，(2) HPFの仕様の多くをサポートし，種々の高度な最適化および並列化の機能を備える，現在最も進んだHPFコンパイラで並列化を行い，(3) 地球シミュレータおよびPCクラスタ上で性能を測定した．その結果，これまでHPFで並列化するのが困難とされていた問題であっても，HPFの機能を最大限に活用してプログラムを開発するとともに，それに対し効率の良い並列コードを生成できる優れたコンパイラを利用すれば，MPIに匹敵する高い性能をより小さな手間で得られることを確認でき，並列プログラミングの場におけるHPFの有用性を示すことができた．

Evaluation of HPF Using NAS Parallel Benchmarks

Hitoshi Murai[†] and Yasuo Okabe[††]

Nowadays parallel languages such as High Performance Fortran (HPF) are attracting attention again. Although previously HPF had not been comparable to MPI in performance, it has been reported that some applications parallelized with the latest HPF compiler achieved as high performance as those with MPI. In this study, in order to evaluate the effectiveness and efficiency of HPF, we (1) implemented LU, CG, and MG of the NAS Parallel Benchmarks by making the best use of the HPF features, on the principle that we only insert HPF directives into the serial Fortran program; (2) parallelized them using the most advanced HPF compiler that supports many of the HPF specifications and has the capabilities of advanced optimization and parallelization; and (3) measured their performance on the Earth Simulator and a PC cluster. The result shows that we can obtain with less effort the performance comparable to that of MPI programs for the problems that have been considered to be difficult to parallelize with HPF if we write the programs by fully exploiting the HPF features and use a good HPF compiler that can produce highly efficient parallel codes for them. It can be said that HPF is useful for parallel programming.

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，High Performance Fortran（HPF）[1]に代表される並列言語の利点はプログラミングの容易さにある．

従来，性能の点でHPFはMPIに及ばないとされていたが，最近では，最新のコンパイラを使ってMPIに並ぶ性能を達成する例も出てきている[2]．しかしながら，さらに多種多様なアプリケーションで高性能を得るためには，HPF言語の機能を十分に活用したプログラミングを行うことが必要であろう．本研究は，HPFの機能を最大限に活用した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ことを目的とする．

地球シミュレータ（ES: Earth Simulator）[3]やPCクラスタの上で利用できるHPF/ES[4]は，HPFの標準仕様に加えてHPF/JA言語拡張[5]やいくつかの独自機能をサポートするとともに，種々の高度な最適化および並列化の機能を備える，現在最も進んだHPFコンパイラの1つである．

NAS Parallel Benchmarks（NPB）は，NASA Advanced Supercomputing（NAS）Division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

† NEC第一コンピュータソフトウェア事業部
1st Computers Software Division, NEC Corporation
†† 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

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）からなる[6),☆]．NPBについてはすでに多くの研究が行われており，さらにHPFによる並列化についても，NAS自身によるもの（NPB3.0 alpha）を含めていくつかの研究がある[7),8)]．しかし，最新のコンパイラと言語仕様を用いた評価が複数の計算機環境上で行われたことはこれまでなかった．

Frumkinらの NPB3.0 alpha[7)] と Nishitaniらの実装[8)] では，ベースとなる逐次 Fortran プログラムを改変することに制限を設けず，データ構造の変更をともなうような大きな改変も行われている．しかし，主に当時のコンパイラの能力が不足していたことからHPFの機能を十分には活用できておらず，実行性能に問題がある場合があった．

これに対し，本研究では，基本的には逐次 Fortran プログラムに HPF 指示文を追加することだけを行うという方針の下で，NPBに含まれる8つのベンチマークのうち，性質の異なるLU，CG，MGの3つをHPF/ESによって並列化する．さらに，それらの性能をESおよびPCクラスタ上で評価し，MPIおよびNPB3.0 alphaの実装と比較する．また，その評価結果と，各ベンチマークを並列化するのに利用したHPFの各機能の有用性と応用について考察を行う．以上により，これまでHPFで並列化するのが困難とされていた問題であっても，HPFの機能を最大限に活用してプログラムを開発するとともに，それに対し効率の良い並列コードを生成できる優れたコンパイラを利用すれば，MPIに匹敵する高い性能をより小さな手間で得られることが示される．

以下，2章でHPFプログラミングの基本方針について，3章で評価環境について説明する．4，5，6章で，それぞれLU，CG，MGのHPFによる実装と評価について述べた後，7章では評価結果を考察し，HPFに代表される並列言語のプログラミングモデルについても検討を行う．最後に，8章で本稿を総括する．

2. HPF プログラミングの基本方針

HPFは，Fortranの拡張として定義されるデータ並列言語である[1)]．プログラマが簡単な指示文の形式でデータの分散方法を指定すれば，コンパイラが自動的に計算の分割と通信の生成を行うため，並列プログラミングの手間は大きく軽減される．さらに，公認拡張仕様やHPF/JA言語拡張[5)]を用いれば，より効率の良い並列コードを得るために，計算の分割や通信の生成の方法をコンパイラに指示することもできる．

本研究では，次の基本方針に従ってHPFプログラミングを行った．この方針は，本研究に限らず，一般のHPFプログラミングにもあてはまるものである．

(1) 並列化を意識していない逐次Fortranプログラムをベースとする．
(2) 基本的には，HPF指示文の追加のみを行い，Fortran文の改変は行わない．Fortran文の改変を行うのは以下の場合に限り，かつ必要最小限とする．
  (a) 「HPF規格合致」でない場合
  (b) 期待する変形がコンパイラによって行われない場合

上の(a)および(b)について説明する．HPFが扱えるFortranプログラムには，通常のFortran仕様に比べてやや厳しい制約がある．たとえば，6章で述べるMGの実装のように，形状が異なる実引数と仮引数を分散することは「HPF規格合致」でない．このような場合には，当該Fortran文を，HPFの規格に合致する等価な文に修正する．また，主としてコンパイラの能力の不足からプログラマの期待する変形や最適化が行われない場合に，コンパイラを補助する目的で，Fortran文に修正または追加を加える．たとえば，4章で述べるLUの実装では，パイプライン並列化の機能を持たないHPF/ESを補助するために，Fortran文（手続き呼び出し）の追加を行った．

3. 評価環境

3.1 HPF/ESコンパイラ

HPF/ESは，地球シミュレータの上で利用できるHPFコンパイラであり，HPF2.0仕様，公認拡張仕様およびHPF/JA言語拡張に加え，いくつかの独自拡張をサポートしている[4)]．

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プログラムへいったんトランスレートしてから，ES上のFortranコンパイラでコンパイルし，ES上のMPIライブラリとリンクする．したがって，あるアプリケーションのHPF/ESによる最適な実装と，MPIによる最適な実装を比較したとき，前者の性能は後者のそれを上限とする．

HPF/ES for PC clusterは，HPF/ESをPCクラスタ用に移植したもので，ESのハードウェアに依存する一部の機能（ベクトル化指示行や並列入出力など）を除き，HPF/ES互換の機能を持つ．

☆ NPB3.1以降，3つのコード（UA，DC，DT）が追加されている．

本研究において，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）は次のとおりである．

```
-Mnomapnew -Mscalarnew
-Mnoerrline -Mnoentry
```

`-Mnomapnew` と `-Mscalarnew` は，`INDEPENDENT` ループ中のすべての非分散配列（暗黙的にマップされた配列）とスカラ変数を `NEW` 変数として扱うことを指示する．`-Mnoerrline` と `-Mnoentry` は，実行時エラーメッセージ用の情報を採取するコードを生成しないことを指示する．

3.2 NAS Parallel Benchmarks

本研究では，NPB の LU，CG，MG の 3 つについて，次の 3 種類の実装の評価を行った．

- **HPF/ES 版**：NAS によるシリアル版の実装（NPB3.1-SER）をベースに，HPF/ES で我々が並列化を行ったもの
- **NPB3.0 alpha 版**：NAS による HPF 版の実装（NPB3.0-HPF）に最小限のバグ修正を行ったもの
- **MPI 版**：NAS による MPI 版の実装（NPB3.1-MPI）

問題のサイズは，ES ではクラス C，PC クラスタではクラス B である．また，実際に測定の対象としたのは，各ベンチマークにおいて，NPB の仕様が規定する計算の実行に要する時間である．

3.3 計算機 1：地球シミュレータ

ES は，計算ノード 640 台をクロスバネットワーク〔バンド幅 12.3 GB/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，16 GB 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）からなる．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とおりである．

- `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`

Fortran コンパイラに指定したオプションは，`-C hopt -Wf"-pvctl vwork=stack"` である．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[9]．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 1 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．

3.4 計算機 2：PC クラスタ

Express5800/Parallel PC-Cluster は，計算ノード 64 台☆を Myrinet–2000 ネットワークで結合した SCore 型 PC クラスタである[10]．各計算ノードは，Pentium III プロセッサ（1.0 BGHz）1 個を搭載し，1 GB のメモリを持つ．

評価に用いたソフトウェア環境は以下のとおりである．

- `HPF/ES for PC Cluster Rev.4.0(1022)`
- `Intel Fortran Compiler 7.0`
- `SCore Ver.5.0.1`
- `MPICH-Score version 1.0 (ch_score)`
- `RedHat Linux 7.2 (kernel 2.4.7)`

Fortran コンパイラに指定したオプションは，`-O3` である．また，各ベンチマークは，SCore マルチユーザ環境で実行した．

4. 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 平面におけるウェーブフロント計算の実行に関し，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を評価することを目的とする．したがって，LU を実装するポイントは，ウェーブフロント計算の並列化である．

以下では，LU のウェーブフロント計算を縮約して得られる次のループを用いて，HPF によるウェーブフロント計算の並列化について説明する．

```
      DO k = 2, n-1
       DO j = 2, n-1
        DO i = 2, n-1
         v(i,j,k) = v(i-1,j,k)
     >            + v(i,j-1,k) + v(i,j,k-1)
        END DO
       END DO
      END DO
```

このウェーブフロント計算は，x，y，z のすべての軸に関してループ運搬依存を含む DOACROSS ルー

☆ ただし，評価を行った時点では，1 台以上の計算ノードが障害などにより利用できなかったため，64 台のすべてを用いた評価は実施していない．

プであるため，単純な DOALL 並列化を適用することはできない．一般に，このようなウェーブフロント計算は，パイプライン法[11] またはハイパープレーン法[12] によって並列化でき，MPI 版では二次元のパイプライン法が用いられている．しかし，現在の HPF の言語仕様では DOACROSS 並列性を記述することができないことから，HPF/ES を含む多くの HPF コンパイラはウェーブフロント計算を並列化することができなかった[7),13)]．

4.1 従来手法

- NPB3.0 alpha 版の実装
  配列 v の y 軸をブロック分散し，次のようにループの構造を変形することで，ハイパープレーン法によってウェーブフロント計算を並列化している．

```
!HPF$ DISTRIBUTE (*,BLOCK,*) :: v

      DO l = lst, lend
!HPF$  REFLECT v
!HPF$  INDEPENDENT
       DO j = jlow(l), jhigh(l)
        DO i = ilow(l,j), ihigh(l,j)
         k = l - i - j
         v(i,j,k) = v(i-1,j,k)
     >            + v(i,j-1,k) + v(i,j,k-1)
        END DO
       END DO
      END DO
```

  DO l ループの 1 回の繰返しが，1 枚のハイパープレーンに対応する．ある 1 枚のハイパープレーン上の計算にはループ運搬依存は存在しないため，内側の二重ループには DOALL 並列化を適用できる．このとき，v に対するシフト通信が，DO l ループの内側（上のコード中で，REFLECT 指示文が置かれている位置．ただし，多くの HPF コンパイラではこのシフト通信は自動的に検出できる）で必要になる．
  この方法の問題は，複雑なループ変形を行う必要があること，特に DO j ループと DO i ループの上下限 jlow，jhigh，ilow および ihigh を計算する処理を追加する必要があることである．
- Nishitani らの実装
  Nishitani らの使用した HPF コンパイラは，プログラム中のウェーブフロント計算を検出し，自動的にパイプライン並列化を行う機能を持つため，LU を並列化するのに特筆すべき工夫は用いられていない．

4.2 HPF/ES 版の実装

4.2.1 ローカル手続きによるパイプライン並列化

HPF/ES 自体はパイプライン並列化の機能を持っていないが，ローカル手続きを利用することで，次のようにパイプライン並列化を実現できる．

```
      DO k = 2, n-1

       CALL PIPELINE_RECV(v,...)

!HPF$  INDEPENDENT
       DO j = 2, n-1
!HPF$   INDEPENDENT
        DO i = 2, n-1
!HPF$    ON HOME(v(i,j,:)), LOCAL
         v(i,j,k) = v(i-1,j,k)
     >            + v(i,j-1,k) + v(i,j,k-1)
        END DO
       END DO

       CALL PIPELINE_SEND(V,...)

      END DO
```

ここで，手続き PIPELINE_RECV と PIPELINE_SEND は，HPF の外来手続きの機能による「ローカル HPF 手続き」である．HPF/ES では，ローカル HPF 手続き内から MPI 手続きを呼び出し，通信や同期を実行することができる．ON-HOME-LOCAL 指示文は，当該ループで通信が不要であることを表明している．したがって，HPF/ES は，当該ループを並列化するとき，HOME 配列 v のマッピングに従って繰返し空間を分割する以外の仕事は行わない．

実行時に，これら 2 つのローカル手続きにおいて，各プロセッサは，HPF ローカルライブラリに含まれるマッピング問合せサブルーチン GLOBAL_ALIGNMENT と GLOBAL_DISTRIBUTION によって，対象の配列のマッピング情報を取得する．次に，取得したマッピング情報に基づいて，指定された幅のパイプライン実行に必要な通信のスケジュールを生成し，それに従って通信を実行する．

v が 4×4 の二次元プロセッサ上へブロック–ブロックで分散されているとき，DO k ループのある繰返しにおける各プロセッサの実行時の振舞いは次のようになる．

(0) プロセッサ P(1,1) は PIPELINE_RECV の実行をただちに終了するが，他のプロセッサは PIPELINE_RECV の中で隣接するプロセッサからデータが到着するのを待つ．
(1) P(1,1) は DO j-DO i ループを実行して，配列 v のローカル部分を更新する．
(2) P(1,1) は PIPELINE_SEND の中で v の境界要素を P(2,1) と P(1,2) へ送信し，DO k ループの次の繰返しへ進む．
(3) 次のサブステップが並列に実行される．
  (a) P(2,1) は P(1,1) から v の境界要素を受信し，PIPELINE_RECV の実行を終了する．

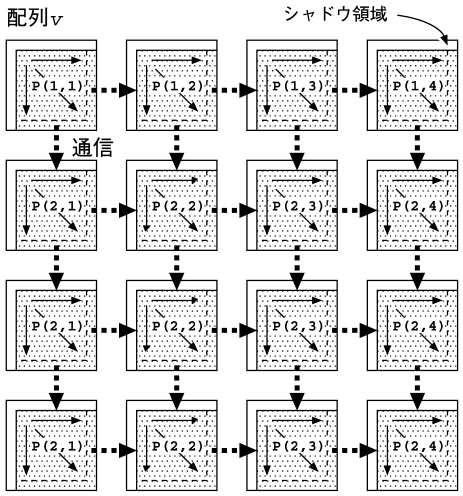


図 1 二次元パイプライン実行
Fig. 1 Two-dimensional pipelined execution.

(b) P(1,2) は P(1,1) から v の境界要素を受信し，PIPELINE_RECV の実行を終了する．

(4) 次のサブステップが並列に実行される．

(a) P(2,1) は DO j-DO i ループを実行して，配列 v のローカル部分を更新する．

(b) P(1,2) は DO j-DO i ループを実行して，配列 v のローカル部分を更新する．

(5) 以下，各プロセッサは同様の処理を続ける．

これらのステップを先に終了したプロセッサは，他のプロセッサが実行を続けているのと並列に，後続の計算を始められることに注意されたい．**図 1** は，4×4 の二次元プロセッサ上のパイプライン実行の様子を示している．

ただし，この実装は，HPF/ES が，PIPELINE_RECV や PIPELINE_SEND のようなローカル手続きの呼び出しの前後にバリア同期を生成しないことに依存している．HPF 仕様に厳密に従うならば，ローカル手続きの出口にはバリア同期が生成されるため，この実装でパイプライン並列化を実現することは不可能である．

4.2.2 PIPELINE 節と HPFX プリプロセッサ

我々は，ウェーブフロント計算をパイプライン並列化するための言語拡張である PIPELINE を提案するとともに，これを解釈してパイプライン並列化されたコードを生成するプリプロセッサ HPFX を開発した[14]．

PIPELINE は，NEW や REDUCTION と同様の，INDEPENDENT 指示文に付加できる節として定義されており，指定された配列に関する，指定された距離と方向のループ運搬依存が INDEPENDENT ループ内に存在することを表明する．下の例では．配列 v の部分 v(:,:,k) に関して，一次元目と二次元目にそれぞれ距離 1 のループ運搬依存が存在することを表明している．

```
      DO k = 2, n-1
!HPF$  INDEPENDENT, PIPELINE(v(:,:,k)(1,1,0))
       DO j = 2, n-1
!HPF$   INDEPENDENT
        DO i = 2, n-1
!HPF$    ON HOME(v(i,j,:)) BEGIN
         v(i,j,k) = v(i-1,j,k)
     >            + v(i,j-1,k) + v(i,j,k-1)
!HPF$    END ON
        END DO
       END DO
      END DO
```

HPFX は，PIPELINE 節で与えられた情報に基づいて，前節の PIPELINE_RECV および PIPELINE_SEND に相当するローカル HPF 手続きを生成するとともに，当該ループの前後にそれらの手続きの呼び出しを挿入する．

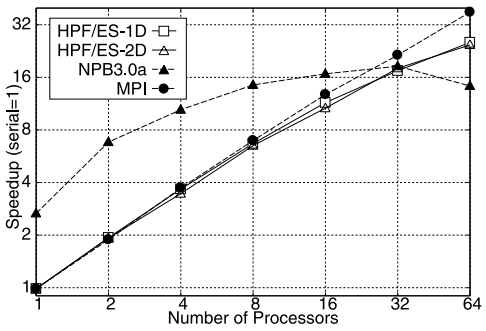
なお，PIPELINE 節は，対象の配列の属性（分散やシャドウ領域など）に関しては何も表明しないため，対象の配列が実際に満たすべき制約は処理系依存となる．現在の HPFX の実装では，パイプライン実行における境界要素の送受信を効率良く行うために，対象の配列の各次元は分散されていないかまたはブロックで分散されており，かつ各次元に対して必要な幅の（ループ運搬依存の距離以上の幅の）シャドウ領域が割り付けられていなければならない．これらの条件が満たされていない場合は，実行時エラーとして検出される．

以上により，PIPELINE 節を追加するという簡単な作業だけで，LU のウェーブフロント計算を並列化することができる．

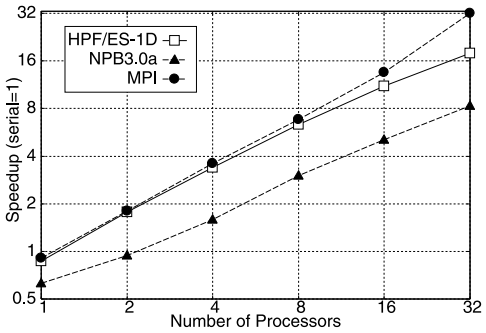
4.3 評　　価

図 2 にベンチマーク LU の評価の結果を示す．「HPF/ES-1D」は，配列の y 次元を分散して一次元パイプライン並列化を行った場合の，「HPF/ES-2D」は，x 次元と y 次元を分散して二次元パイプライン並列化を行った場合の結果である．ただし，本研究で使用した HPF/ES for PC cluster には，多次元分散された配列の扱いに制約があったため，PC クラスタ上では「HPF/ES-2D」は評価できていない．グラフの縦軸はシリアル版の性能を 1 とする相対性能である．

グラフより，HPF/ES が MPI とほぼ同等の性能を示していることが分かる．高並列（32〜64 プロセッサ）時に，HPF/ES の効率がやや低下するのは，HPFX が生成するローカル手続きにおいて，マッピング問合



(a) ES



(b) PC クラスタ

図 2 LU の評価結果
Fig. 2 Evaluation result of LU.

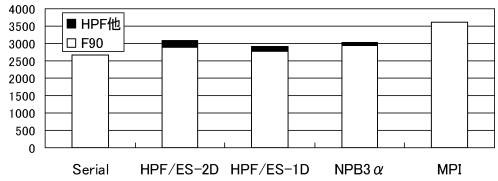


図 3 LU の各実装におけるプログラム行数
Fig. 3 Number of the program lines in each implementation of LU.

せサブルーチンの呼び出しと通信スケジュールの生成が頻発するためである．前者のオーバヘッドは，コンパイラ・ランタイムが管理する情報から直接にマッピング情報を取得することによって，後者は，通信スケジュール再利用の最適化を適用することによって，それぞれ削減することができる．現在の HPFX はこのどちらの機能も備えていないが，HPFX が生成したプログラムに対して HPF/ES が出力する中間 Fortran プログラムを，その 2 点について人手で修正して評価を行った結果，高並列時の HPF/ES の性能も MPI と同等になることを確認している．HPFX にこれらの機能を実装することは，今後の課題である．

NPB3.0 alpha は，特に ES の低並列（1〜16 プロセッサ）時に他の実装より性能が高い．これは，ハイパープレーン法によってループ運搬依存が解消されることによって，ウェーブフロント計算を並列化できるだけでなくベクトル化することも可能になったためである．パイプライン法による並列化では，ループ運搬依存が解消されるわけではないため，ベクトル実行の効率はそれほど良くない．以上は，（ベクトルプロセッサを持たない）PC クラスタ上では，NPB3.0 alpha の性能が他の 50%程度にとどまっていることからも分かる．一方，NPB3.0 alpha のスケーラビリティが ES の高並列（32〜64 プロセッサ）時に悪化するのは負荷不均衡のためである．ある 1 枚のハイパープレーン上の処理は，すべてのプロセッサに均等に配分されるわけではないため，負荷不均衡が起こりうるが，その影響は高並列時により顕著になる．

図 3 は，シリアル版，HPF/ES 版（一次元および二次元），NPB3.0 alpha 版および MPI 版の 5 つの実装における LU のプログラム行数を示す．プログラム行数はプログラミングの容易さを計る 1 つの指標になると考えられる．棒グラフの黒い領域が，HPF による並列化のために追加した HPF 指示文，`EXTRINSIC` 接頭辞，手続き引用仕様宣言などに対応し，白い領域はそれ以外の Fortran 文に対応する（次章以降の図においても同様）．HPF/ES 版のプログラム行数は，`PIPELINE` 指示文と HPFX プリプロセッサを利用する場合の値である．

HPF/ES による LU の実装では，シリアル版に比べ 10〜15%程度のプログラム行数の増加が見られるが，MPI 版に比べるとその増加量は半分以下に抑えられている．これは，HPF による並列化が，MPI による並列化に比べて容易であることを示している．また，NPB3.0 alpha 版のプログラム行数は HPF/ES 版にほぼ近い．

5. 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．Compressed Row Storage（CRS）形式[15] 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．

シリアル版において，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

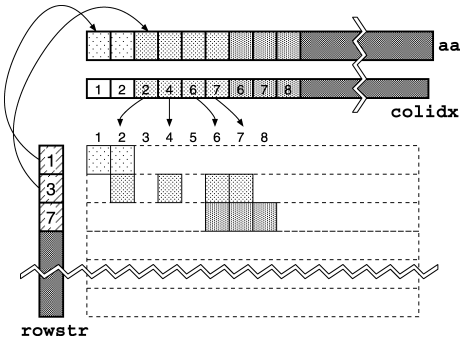


図 4 Compressed Row Storage 形式
Fig. 4 Compressed Row Storage format.

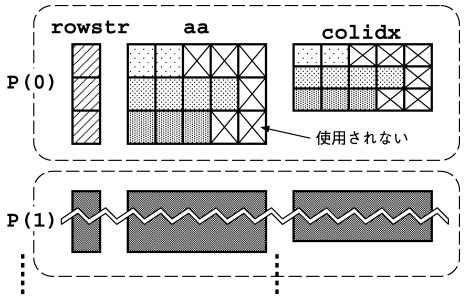


図 5 従来手法における CRS 形式の分散
Fig. 5 Distribution of CRS format in the conventional method.

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（**図 4**）．

5.1 従来手法

NPB3.0 alpha 版と Nishitani らの実装では，下のように，配列 aa と colidx の次元を拡張して疎行列の一次元目（行方向）に対応付け，その次元をブロック分散することを行っている（**図 5**）．このとき，配列 b および c はブロック分散される．

```
!HPF$ DISTRIBUTE (*,BLOCK) :: aa, colidx
!HPF$ DISTRIBUTE (BLOCK) :: b, c

!HPF$ INDEPENDENT
      DO i=1, N
       DO j=1, rowstr(i+1)-rowstr(i)+1
        c(i) = c(i) + aa(j,i)*b(colidx(j,i))
       END DO
      END DO
```

この方法では，データ構造の変更をともなう Fortran 文の改変が必要になる．また，行列ベクトル積の実行の直前に，配列 b に対する ALL_GATHER 通信が必要になる．ALL_GATHER 通信では，全対全の送受信が行われる．

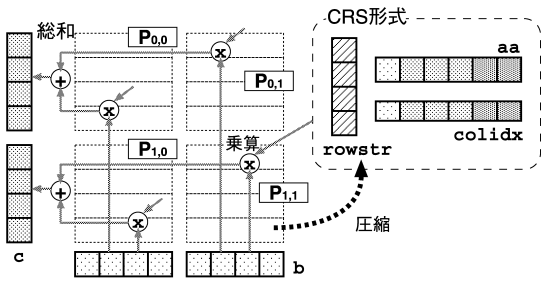


図 6 二次元分散による CG の実装
Fig. 6 Implementation with 2-dimensional distribution.

5.2 HPF/ES 版の実装

CG を並列化する方法はいくつか提案されているが，通信量の点で二次元分散を利用する方法が最も効率が良いとされており[16]，MPI 版でもこの方法が採用されている．

二次元のプロセッサ配列の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．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し，一次元目と二次元目の方向には複製されるように，分散する（**図 6**）．たとえば，図 6 において，プロセッサ $P_{0,0}$ と $P_{1,0}$ は，ともに配列 b の前半部を保持している．これらの配列に対する分散指示文は次のようになる．テンプレート t が元の疎行列に対応している．

```
!HPF$ TEMPLATE t(N,N)
!HPF$ DISTRIBUTE t(BLOCK,BLOCK)
!HPF$ ALIGN b(j) WITH t(*,j)
!HPF$ ALIGN c(i) WITH t(i,*)

!HPF$ ALIGN rowstr(i) WITH t(i,*)
      DIMENSION aa(nz/nprocs)
      DIMENSION colidx(nz/nprocs)
```

CRS 形式を構成する aa，colidx，rowstr の 3 つの配列は「ローカルデータ」として扱う．aa および colidx は，あらかじめ分散後のサイズで宣言しておき，明示的な分散は指定しない．疎行列は実行時に擬似乱数によって生成されるため，各プロセッサにおける分散後の正確なサイズは実行前には不明である．しかし，本実装では，シリアル版のサイズをプロセッサ数で等分した値（nz/nprocs）を分散後のサイズとしておけば十分であることが分かっている．rowstr は，疎行列の一次元目へ整列するように分散を指定する．

ここで，ローカルデータとは，各プロセッサが個別に保持するデータのことである．HPF プログラムの実行時に，通常のデータ（グローバルデータ）の実体が概念上はただ 1 つしか存在しない（複製が存在する

場合でも，それらの一致制御は処理系が保証する）のに対し，ローカルデータの実体は各プロセッサ上に存在し，各プロセッサはそれらを独立して定義・参照するため，プロセッサによって異なる値をとりうる．これらのローカルデータの初期化や設定はローカル手続きで行う．ただし，このようなローカルデータの存在は，厳密には HPF の規格に従っていない．

本実装では，ローカル手続きの中で，疎行列の非零要素を生成する処理を行う．生成される非零要素が他プロセッサに割り当てられる要素であった場合，これを捨て，自プロセッサに割り当てられる要素のみを CRS 形式に保持する．非零要素を生成するローカル手続きのコードを下に示す．

```
      EXTRINSIC (HPF_LOCAL)
     >     SUBROUTINE sparse(...)

      firstrow = ... ; lastrow = ...
      firstcol = ... ; lastcol = ...

      DO i = 1, N
       DO nza = 1, arow(i)
        j = acol(nza, i)
        IF (j < firstrow .OR.
     >      j > lastrow) CYCLE
        DO nzrow = 1, arow(i)
         jcol = acol(nzrow, i)
         IF (jcol < firstcol .OR.
     >       jcol > lastcol) cycle

         jj = j - firstrow + 1
         DO k=rowstr(jj), rowstr(jj+1)-1
          ...
          colidx(k) = ...
          a(k) = ...
         END DO

        END DO
       END DO
      END DO

      END
```

図 6 を見ると，行列ベクトル積の実行において各プロセッサの行う処理は次のようになる．各プロセッサは，自分が保持する b の複製と，CRS 形式で表現されたローカルな aa に対して行列ベクトル積を実行する．次に，各プロセッサで得られた c の部分和に対し，疎行列の二次元目の方向に総和処理を行い，結果 c を得る．

HPF/ES 版において，行列ベクトル積を実行するコードを次に示す．

```
!HPF$ TEMPLATE t2(N,np_col)
!HPF$ DISTRIBUTE t2(block,block)

!HPF$ INDEPENDENT, REDUCTION(c)
      DO ii=1, np_col
!HPF$  ON HOME(t2(:,ii))

!HPF$  INDEPENDENT
       DO i=1, N
!HPF$   ON HOME(ttt(j,ii)), LOCAL(b,rowstr)
        DO j=rowstr(i), rowstr(i+1)-1
         c(i) = c(i) + aa(j)*b(colidx(j))
        END DO
       END DO

      END DO
```

ローカルデータである rowstr の値によって上下限が決定される DO j ループの内側が，各プロセッサごとの「ローカルな」処理である．追加された外側 DO ii ループは，疎行列の二次元目の方向の総和処理をコンパイラに指示するためのダミーである（各プロセッサは DO ii ループの 1 回の繰返しのみを実行する）．

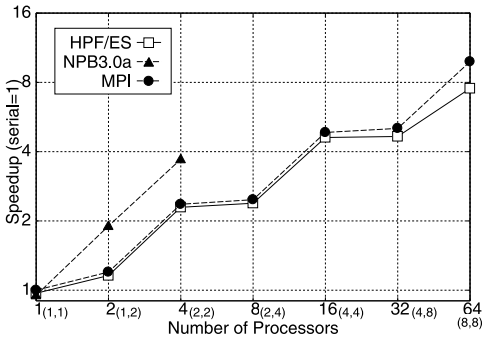
本手法では，配列 c に対する総和処理にともなう通信（総和通信と呼ぶ）を除いて通信を必要とせずに行列ベクトル積を実行することができる．ただし，b と c の分散が異なるため，行列ベクトル積の後で，c を参照して b を更新するときに通信（転置通信と呼ぶ）が必要となる．転置通信では，プロセッサ $P_{i,j}$ と $P_{j,i}$ との間で 1 対 1 の送受信が行われる．

5.3 評　　価

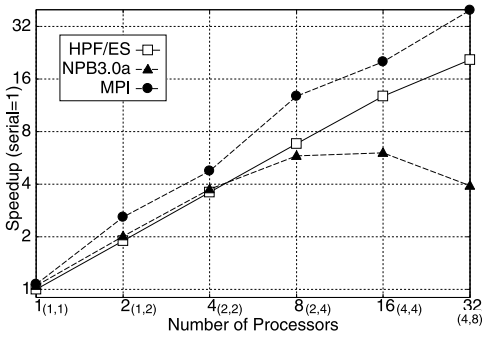
図 7 にベンチマーク CG の評価の結果を示す．グラフの縦軸はシリアル版の性能を 1 とする相対性能である．横軸の括弧内に示した数字は，二次元プロセッサ配列の構成を示す．

ES では HPF/ES と MPI の性能がほぼ同等であるのに対し，PC クラスタでは HPF/ES の性能は MPI の 50〜90%程度にとどまっている．次のような理由が推測されるが，これを実測によって確認することは今後の課題である．

まず，ベンチマーク対象部分に関し，HPF/ES 版のプログラムに対して HPF/ES が生成した中間 Fortran プログラムと，MPI 版のプログラムは，総和通信と転置通信を除きほぼ等価であった．MPI 版では，総和通信および転置通信は，1 対 1 通信（SEND-RECV）を人手でスケジュールすることによって実装されている．一方，HPF/ES 版では，それらの通信は，コンパイラ・ランタイムから呼ばれる MPI 集団通信によって実行される．そのような集団通信は，MPI ライブラリの実装が通信ハードウェアに対し十分に最適化されていれば，人手による 1 対 1 通信と同等かそれを上回る性能を発揮できるが，最適化が不十分であればそ



(a) ES



(b) PC クラスタ

図 7 CG の評価結果
Fig. 7 Evaluation result of CG.

れには及ばない．ES 上の MPI ライブラリがベンダによって十分に最適化されているのに対し，PC クラスタ上の MPI ライブラリの最適化は十分ではなかったものと思われる．

また，ES において，HPF/ES と MPI の速度向上が階段状になるとともに，PC クラスタに比べて速度向上の値が小さくなるのは，次の理由による．

HPF/ES 版および MPI 版では，元の疎行列の二次元目も分散される．二次元目が分散されると，この次元方向に対応する，行列ベクトル積の `DO j` ループのループ長が短くなり，ベクトル実行の効率が低下する．すなわち，プロセッサ数が増加するにつれて，各プロセッサが処理すべき仕事の量自体が減少する（並列化による台数効果が大きくなる）一方で，ベクトル実行の効率が低下してしまうため，全体として速度向上の値は小さくなる．また，ベクトル実行の効率が，二次元プロセッサ構成の一次元目と二次元目の寸法の増減に関して非対称的な挙動を示すことが，速度向上の値に階段状の影響を及ぼす．

行列ベクトル積以外の計算は，`b` の分散に従って並列化されるため，プロセッサ構成の二次元目の寸法に等しい並列度しか持たない．これは，性能向上を悪化させる要因となるはずだが，本研究で用いた評価環境

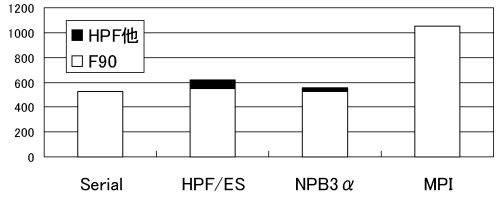


図 8 CG の各実装におけるプログラム行数
Fig. 8 Number of the program lines in each implementation of CG.

では行列ベクトル積の計算と通信が支配的であったために，実際にはほとんど影響がなかった．

NPB3.0 alpha は，ES の 8 プロセッサ以上では実行することができなかった．これは，疎行列の初期化を行う手続きで使用される作業配列が分散されておらず，使用されるメモリサイズがプロセッサ数に比例して大きくなるためである．ES 上の 1〜4 プロセッサの実行で，HPF/ES 版および MPI 版の性能を上回るのは，ベクトル化される `DO j` ループのループ長が長く保たれるためである．前述のように，元の疎行列の二次元目も分散する HPF/ES 版および MPI 版では，プロセッサ数が増えるとともにベクトル長が短くなる．また，PC クラスタにおいて，1〜8 プロセッサの実行では HPF/ES 版の性能にほぼ近いが，16〜32 プロセッサでは急速に性能が低下する．これは，実行時間がプロセッサ数に比例する ALL_GATHER 通信のオーバヘッドの影響によるものと考えられる．

図 8 から，MPI 版のプログラム行数はシリアル版の 2 倍以上になっているのに対し，HPF/ES 版では 10〜15%程度の HPF 指示文などが追加されているだけで，シリアル版とあまり変わっていないことが分かる．NPB3.0 alpha 版は，追加された HPF 指示文などを除けば，シリアル版とほぼ同じプログラム行数であり，HPF/ES 版よりもわずかに少ない．これは，NPB3.0 alpha 版では，HPF による並列化の際に，配列 `aa` と `colidx` の出現を置き換えるだけで，新規の Fortran 文を追加する必要がほとんどなかったためである．

6. 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．**図 9** に，MG における階層的実行（V サイクル）を示す．

MG を実装する際には，V サイクルの中で参照される階層的データの分散をどのように実現するかがポイントとなる．

シリアル版および MPI 版では，V サイクルは，下

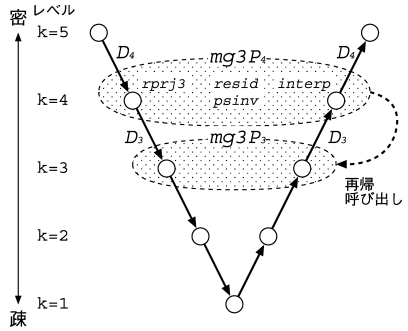


図 9 MG における V サイクル
Fig. 9 V-cycle in MG.

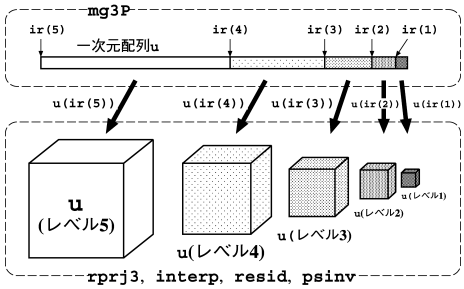


図 10 シリアル版 MG における階層的データの実装
Fig. 10 Implementation of hierarchical data in serial MG.

に示すように，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,r,k)

      DO k = lt, 2, -1
       j = k-1
       CALL rprj3(r(ir(k)),r(ir(j)),k)
      END DO

      CALL psinv(r(ir(1)),u(ir(1)),1)

      DO k = 2, lt
       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)
      END DO

      END
```

ここでは，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的データとして利用している（**図 10**）．このような引数渡しは，Fortran プログラミングにおいては一般的なテクニックだが，分散された実引数と仮引数の形状が異なることは HPF 規格合致ではない．したがって，これらの階層的データを分散し，HPF によって MG を並列化するためには，Fortran 文の改変を含む何らかの工夫が必要である．

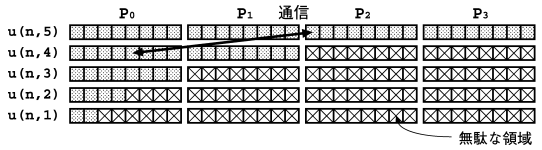


図 11 NPB3.0 alpha 版 MG における階層的データの実装
Fig. 11 Implementation of hierarchical data in NPB3.0 alpha MG.

6.1 従来手法

- NPB3.0 alpha 版の実装
  階層的データに階層（レベル）を表す次元を付加する．**図 11**（簡単のため，一次元配列で表記）に示すように，無駄な（実際には使用されない）領域が大きいことに加えて，階層的データ間のやりと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 u(*,*,BLOCK,*)
```

- Nishitani らの実装
  各階層に属する階層的データを別々の配列として宣言し，IF 文で階層ごと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．各階層的データに対して，後述する最適な分散を直接に指定できるため，無駄な領域や不要な通信は発生しないが，階層の数によって，データの宣言や手続き呼び出しを変更しなければならない煩雑さがある．

```
      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,...)
```

6.2 HPF/ES 版の実装

まず，手続き mg3P を，ある階層の処理だけを行うように書き直し（図 9 の網掛け部分），これを下のように再帰的に呼び出すことによって，V サイクルの階層的手続き呼び出しを実現する．

```
      RECURSIVE SUBROUTINE mg3P(u,r,k)

      IF (k > 1) THEN
         CALL rprj3(r,r2,k)
         CALL mg3p(u2,r2,k-1)
         CALL interp(u2,u,k)
         CALL resid(u,r,k)
      END IF

      CALL psinv(r,u,k)

      END
```

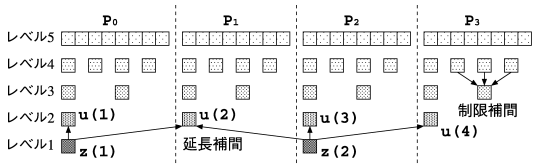


図 12 HPF/ES 版 MG における階層的データの実装
Fig. 12 Implementation of hierarchical data in HPF/ES MG.

次に，各階層的データ（手続き mg3P の引数である配列 u および r など）を分散する方法について考える．

階層的データが，**図 12**（簡単のため，一次元配列で表記）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とりにおいて通信は発生しない．このような分散を実現するためには，各階層的データをレベルに応じたストライドで整列させる必要がある．

V サイクルのレベル k における手続き mg3P のインスタンスを $mg3P_k$，レベル k の階層的データを D_k とする．また，L_{top} を V サイクルの初期レベル（最も密な階層のレベル）とする．

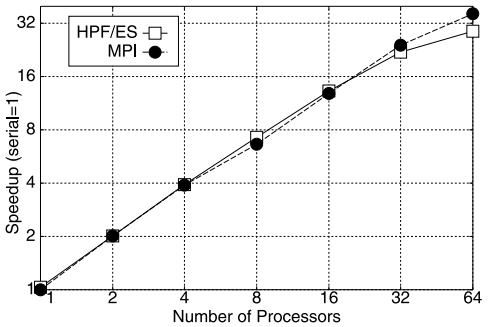
$mg3P_k$ は，$mg3P_{k+1}$ から自分のレベル k と D_k を引数として受け取る．このとき，下のように，D_k（仮引数 u および r）を，$D_{L_{top}}$ に対してストライド $2^{L_{top}-k}$ で整列させる．

```
!HPF$ TEMPLATE t(N)
!HPF$ DISTRIBUTE (BLOCK) :: t

      DOUBLE PRECISION u(n), r(n)
!HPF$ ALIGN (i) WITH
!HPF$>     *t(i*(2**(lt-k))-m) :: u, r

      DOUBLE PRECISION, ALLOCATABLE ::
     >                          u2(:), r2(:)
!HPF$ ALIGN (i) WITH
!HPF$>     t(i*(2**(lt-k+1))-m2) :: u2, r2
```

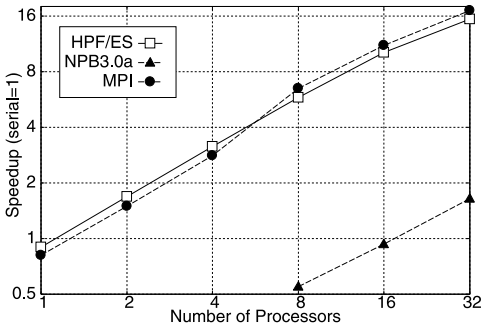
さらに，D_{k-1}（上の u2 および r2）の領域を割り付けるとともに，ストライド $2^{L_{top}-k+1}$ の整列を指定し，$mg3P_k$ から再帰的に呼び出される $mg3P_{k-1}$ へ引数として渡す．

以上により，階層的データに対して，階層間のやりとりにおいて通信が発生しない最適な分散を実現できる．加えて，無駄な領域が割り付けられることや，プログラムが階層の数に依存することもない．

本手法の利点は，手続き mg3P が，2 つのレベルの階層的データのみを扱えばよいことにある．HPF および Fortran では，「ポインタの配列」を扱うことができないことから，複数の配列を番号付けして管理するには図 10 のような方法をとらざるをえない．これに対し，再帰呼び出しを利用する本手法では，仮引数として渡される D_k と，ローカルな割付け配列として宣言される D_{k-1} の 2 つのレベルの階層的データのみを扱えばよいため，それぞれを三次元配列として宣言して別々の名前を付けることにより，実引数と仮引数の形状を一致させることができる．



(a) ES



(b) PC クラスタ

図 13 MG の評価結果
Fig. 13 Evaluation result of MG.

6.3 評　　価

図 13 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の性能を 1 とする相対性能である．NPB3.0 alpha は，上述のように無駄な領域が大きいために，ES では実行することができず，PC クラスタでも 1〜4 プロセッサでは実行できなかった．

HPF/ES は，ES と PC クラスタのいずれの場合にも，MPI とほぼ同等の性能を示している．NPB3.0 alpha は，本来は不要な通信が発生しているために，PC クラスタ上で実行できた場合でも，HPF/ES および MPI に比べて性能は低い．

ES の高並列（32〜64 プロセッサ）時に HPF/ES の効率がやや低下するのは，疎なグリッドの格子点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が例外的な処理を行うためである．たとえば，図 12 において，プロセッサ P1 は，レベル 2 のデータ u(2) を更新するために，両隣のプロセッサからレベル 1 のデータ z(1) および z(2) を受信する必要がある．通

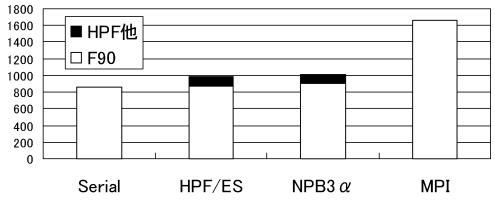


図 14 MG の各実装におけるプログラム行数
Fig. 14 Number of the program lines in each implementation of MG.

常，HPF では，このような通信は，参照するデータにシャドウ領域を宣言し，REFLECT 指示文を実行することによって実現できる．しかし，プロセッサ P_1 には，z の要素が割り当てられておらず，したがってシャドウ領域が存在しないため，この通常の方法を適用することはできない．そこで，u と同じサイズで，シャドウを持つ作業配列 tmp を介して，下のように u を更新する．

```
!HPF$ SHADOW tmp(1:1)

!HPF$ INDEPENDENT
      DO i=1, m
!HPF$  ON HOME(z(i)), LOCAL
       tmp(i*2-1) = z(i)
      END DO

!HPFJ REFLECT tmp

      ...

!HPF$ INDEPENDENT
      DO i=1, m-1
!HPF$  ON HOME(u(2*i)), LOCAL
       u(2*i) = u(2*i) +
     >  0.5d0 * (tmp(2*i+1) + tmp(2*i-1))
      END DO
```

MPI 版ではこの例外処理をうまく記述できるため効率の低下が小さく抑えられる．

図 14 から，MPI 版のプログラム行数はシリアル版の 2 倍近くになるのに対し，HPF/ES 版では 10〜15%程度の HPF 指示行などが追加されているだけで，シリアル版とあまり変わっていないことが分かる．HPF3.0 alpha 版のプログラム行数は HPF/ES 版とほぼ同じである．

7. 考　　察

7.1 各ベンチマークの評価結果

- LU

現在の HPF の言語仕様には，プライベート変数の使用または集計演算以外に起因するループ運搬依存の存在を表明する手段は含まれないため，プログラマがウェーブフロント計算の並列化をコンパイラに指示することはできない．しかし，HPF の言語仕様の自然な拡張である PIPELINE 節によってサポートできるという意味において，パイプライン並列化は HPF の潜在的な機能の 1 つだと考えられる．実際には，ウェーブフロント計算に自動的にパイプライン並列化を適用することは，コンパイラ技術としてはそれほど高度なものではなく，事実いくつかの HPF コンパイラはすでにこの機能を備えている[8),17)]．しかし，複雑なループに対して自動的にパイプライン並列化を適用するのは難しい場合もあることから，我々が提案している PIPELINE 節のような機能は有用である．

また，HPFX プリプロセッサを利用する LU の実装では，マッピング問合せ手続きの呼び出しと通信スケジュールの生成が頻発することが，スケーラビリティを悪化させる要因であった．これらのオーバヘッドは，パイプライン並列化の機能を HPF コンパイラがサポートすれば回避できることから，各 HPF コンパイラによるパイプライン並列化のサポートが望まれる．

パイプライン並列化の機能を持つコンパイラを利用できる場合またはコンパイラが PIPELINE 節のような機能をサポートしている場合には，ウェーブフロント計算を並列化するには，ハイパープレーン法よりパイプライン法を用いる方がプログラミングは容易であるといえる．しかし，ハイパープレーン法の方がベクトル処理との相性が良いため，ES のようなベクトル計算機上で並列度があまり高くない場合には，ハイパープレーン法を用いた方が有利である．

- CG

HPF/ES 版の実装で用いた二次元分散に基づく方法は，通信量の点で効率が良く，プロセッサ数が増加しても速度向上の低下は小さい．しかし，一方の次元方向の並列度が大きくなるにつれてベクトル実行の効率が低下するという性質を持つため，ベクトル計算機よりもスカラ計算機により向いているといえる．これに対し，配列の次元拡張をともなう従来手法では，ベクトル実行の効率は高いが，通信のオーバヘッドが大きいため，プロセッサ数が増加したときの速度向上の低下が著しい．なお，HPF/ES 版の実装において，二次元プロセッサ配列の形状を変更することで従来手法と同様の並列化を実現することもできる．この変更はプログラムの改変を必要とせず，パラメータを変更するだけで実現できる．

一方，HPF の外来機構に基づくローカル手続きの機能を用いれば，次のようなプログラミング技法を用いることが可能になる．

– MPI 手続きの呼び出しによる通信の高速化
– SPMD モデルに基づく並列化

– 既存の Fortran ライブラリの利用

この機能の過度な使用は，HPF の利点であるシンプルなプログラミングを損なう原因になり必ずしも好ましくはないが，適度な使用はプログラムを高速化するのに役立つ．たとえば，ローカルデータを用いる HPF/ES 版の CG の実装は，本来の HPF の並列実行モデルから外れるものではあるが，HPF で並列化することが難しいとされる不規則なデータ構造（CRS 形式）を扱うプログラムを並列化するのに効果的であった．そのような意味において，このプログラミング手法は，HPF の潜在能力の一端を示すものだと考えることができる．

- MG

6 章で述べたように，MG を HPF で並列化するには，整合配列や割付け配列などの，サイズが動的に決まる（コンパイル時に定まらない）データのマッピングを静的に指定できること，さらに，そのマッピングのパラメータ（整列のストライドやオフセットなど）として，コンパイル時に値が定まらない変数を含む式（宣言式）を指定できることが必要であった．より一般に，マルチグリッド法などの動的なアプリケーションを並列化するには，サイズやマッピングが動的に変わりうる「動的データ」を効率良く扱う必要がある．HPF の言語仕様は，そのための機能として，公認拡張仕様の動的再マッピングなどを備えているが，HPF/ES を含め，多くの HPF コンパイラで，動的データを使用することは大幅な性能低下につながる場合が多い．これは，コンパイラが，データ（配列）のサイズやマッピングに関する知識を利用した最適化を多く行うためである．そのような知識が得られない場合には，最適化が不十分な効率の悪いコードを生成する．この問題は，公認拡張仕様の `RANGE` 指示文や HPF/ES の独自拡張である `ASSERTION` 指示構造[4] を用いたりプログラムを改変することである程度は回避できるが，コンパイラの解析能力を強化し，データのサイズやマッピングに関する情報のフロー解析を行うことによって解決されることが望ましい．

7.2 並列言語のプログラミングモデル

HPF でプログラムを並列化する際には，Fortran 文自体は基本的に変更する必要がない．本研究でも Fortran 文の改変を行わないことを基本方針として掲げた．この意味するところは，HPF プログラムにおける配列およびループのインデックスは逐次プログラムのそれと変わらない，すなわち HPF によるプログラミングはグローバルであるということである．我々は，これが HPF の最大の利点であり欠点であると考えている．

グローバルなプログラミングでは，プログラマは，行わせたい処理の全体を記述すればよい．データや計算を各プロセッサに分配するために実際にインデックスをローカル化する（SPMD プログラムを生成する）仕事は処理系によって行われるため，プログラマの負担は小さい．一方で，プロセッサごとのローカルなデータや計算を直接に記述することはできない．したがって，通信や計算のスケジューリングといった，並列実行の効率に関わる要因を細かく制御することができないため，プログラムの最高の性能を得ることは難しい．

それに対し，MPI や Co-Array Fortran（CAF）[18] のプログラミングはローカルである．ローカルなプログラミングでは，労力と引き換えに，より効率の良いプログラムを得ることができる．

将来の HPF またはその他の並列言語には，グローバルなプログラミングを基本としつつ，それを損なうことなく，特定の箇所ではローカルなプログラミングによって並列実行の効率を高めることができるような機能が必要だと考えられる．

5 章や 7.1 節で述べたように，ローカル手続きの機能を用いれば，HPF にローカルなプログラミングを導入することができるが，より簡便なインタフェースが望まれる．たとえば，Unified Parallel C（UPC）[19] では，分散を指定されたグローバルなデータに対し，`for` ループによってローカルな計算を，`upc_forall` ループによってグローバルな計算を，それぞれ記述することができる．

8. おわりに

NPB の LU，CG，MG を HPF で実装し，最新の HPF コンパイラである HPF/ES で並列化した．その際に，LU では `PIPELINE` 節によるパイプライン並列化が，CG ではローカル手続きの利用が，MG では動的な配列に対するマッピング指定が，それぞれ重要であった．ES および PC クラスタ上でそれらの性能を測定した結果，HPF の機能を最大限に活用してプログラムを開発するとともに，それに対し効率の良い並列コードを生成できる優れたコンパイラを利用すれば，多くの場合で，MPI に匹敵する高い性能をより小さな手間で得られることを確認できた．MPI に比べて性能の低下が見られたいくつかの場合では，その原因を特定し，処理系の改良などの対策によって問題を解決しうることを示した．

現在，米国防総省高等研究計画局（DARPA）の

High Productivity Computing Systems（HPCS）Program では，IBM，Sun，Cray の各社によって，X10[20]，Fortress[21]，Chapel[22] という並列言語の開発が進められている．また，将来，Fortran 標準に Co-Array Fortran の仕様が取り込まれることが決まっている[23]．このように，MPI によらない，言語としての並列化手段を模索する動きが再び高まりつつある中で，並列言語のデファクト標準として認知されている HPF の評価を行ったことは，将来の並列言語の可能性を計るという観点からも意義があると考えられる．

今後の課題として，各章であげたもののほかに，さらに多くの HPF 処理系，プラットフォーム，アプリケーションやベンチマークに対して評価を行い，HPF の有効性と能力を検証することがあげられる．

謝辞 本研究は，平成 16 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 Version 2.0 (1997).
2) Sakagami, H. et al.: 14.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator, *Proc. SC2002* (2002).
3) Habata, S. et al.: Hardware System of the Earth Simulator, *Parallel Computing*, Vol.30, No.12, pp.1287–1313 (2004).
4) Yanagawa, T. and Suehiro, K.: Software System of the Earth Simulator, *Parallel Computing*, Vol.30, No.12, pp.1315–1327 (2004).
5) High Performance Fortran Forum: HPF/JA 言語仕様書，High Performance Fortran 2.0 公式マニュアル，財団法人高度情報科学技術研究機構（編），pp.275–375, シュプリンガー・フェアラーク東京 (1999).
6) Bailey, D. et al.: THE NAS PARALLEL BENCHMARKS, Technical Report NAS-94-007, Nasa Ames Research Center (1994).
7) Frumkin, M. et al.: Implementation of NAS parallel benchmarks in high peroformance fortran, Technical Report NAS-98-009, Nasa Ames Research Center (1998).
8) 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, pp.769–787 (2002).
9) Uno, A.: Software of the Earth Simulator, *J. Earth Simulator*, Vol.3, pp.52–59 (2005).
10) Ishikawa, Y. et al.: RWC PC Cluster II and SCore Cluster System Software — High Performance Linux Cluster, *Proc. 5th Annual Linux Expo*, pp.55–62 (1999).
11) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Weley, Redwood City, CA (1996).
12) Lamport, L.: The Parallel Execution of DO Loops, *Comm. ACM*, Vol.17, No.2, pp.83–93 (1974).
13) 村井 均，岡部寿男：地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価，*Proc. SACSIS2004*, pp.389–396 (2004).
14) Murai, H. and Okabe, Y.: Pipelined Parallelization in HPF programs on the Earth Simulator, *Proc. 6th Intl. Symposium on HPC* (*ISHPC-VI*) (2005).
15) Barrett, R. et al.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd edition, SIAM, Philadelphia (1994).
16) Lewis, J.G. et al.: Distributed Memory Matrix-Vector Multiplication and Conjugate Gradient Algorithm, *Proc. SC'93*, pp.484–492 (1993).
17) Gupta, M. et al.: An HPF compiler for the IBM SP2, *Proc. SC'95* (1995).
18) Numrich, R.W. and Reid, J.: Co-array Fortran for parallel programming, *ACM SIGPLAN Fortran Forum*, Vol.17, No.2, pp.1–31 (1998).
19) El-Ghazawi, T. et al.: *UPC: Distributed Shared Memory Programming*, John Wiley and Sons (2005).
20) Charles, P. et al.: X10: An Object–oriented Approach to Non–Uniform Cluster Computing, *Proc. OOPSLA 05* (2005).
21) Allen, E. et al.: The Fortress Language Specification, version 0.707 (2005). http://research.sun.com/projects/plrg/fortress0707.pdf
22) Callahan, D. et al.: The Cascade High Productivity Language, *Proc. 9th Intl. Workshop on High-Level Parallel Programming Models and Supportive Environments* (*HIPS 2004*), pp.52–60, IEEE Computer Society (2004).
23) Numrich, R.W. and Reid, J.: Co-arrays in the next Fortran Standard, *ACM SIGPLAN Fortran Forum*, Vol.24, No.2, pp.4–17 (2005).

(平成 17 年 10 月 4 日受付)
(平成 18 年 2 月 14 日採録)

村井　均（正会員）

1996 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年日本電気（株）入社．C&C 研究所等を経て，現在，第一コンピュータソフトウェア事業部に勤務．2002 年から 2005 年にかけて，海洋科学技術センター（現，海洋研究開発機構）へ出向，地球シミュレータセンターに勤務．並列処理および並列化コンパイラ技術の研究開発に従事．2002 年ゴードン・ベル賞受賞．

岡部　寿男（正会員）

1988 年京都大学大学院工学研究科情報工学専攻修士課程修了．同大学工学部助手，大型計算機センター助教授等を経て，2002 年より京都大学学術情報メディアセンター教授．博士（工学）．並列・分散アルゴリズム，インターネットワーキング等の研究に従事．電子情報通信学会，システム制御情報学会，日本ソフトウェア科学会，IEEE, ACM，EATCS 各会員．