HPF処理系Parallel FORTRANによるNAS Parallelベンチマークの並列化

太田 寛[†] 西谷康仁[††]
小林 篤[†††] 布広永示[††]

分散メモリ型並列スーパコン用の言語として，HPF（High Performance Fortran）が提案されているが，まだ実用化の域に十分に達してはいない．その理由として，指示文の不足，処理系の最適化機能の不足，さらに様々な計算のHPFによる実装方法が必ずしも明らかでないなどの問題があげられる．並列スーパコンの代表的ベンチマークであるNAS Parallel Benchmarks（NPB）に対しても，これまでHPFによる効率的な実装は部分的にしか行われていなかった．本研究ではNPBの全8本を対象として，上記問題を具体的に示し，それらを解決するための，処理系の必要機能およびHPFによるベンチマークの実装方法を明らかにする．言語仕様上の機能としては，いくつかの拡張指示文のサポートが必要であり，最適化機能に関しては，ガードのループ外追い出し，およびDOACROSSループの粒度調節が効果がある．これらの機能は，我々の開発したHPF処理系Parallel FORTRANに実装されている．Parallel FORTRANを用いて，我々が実装したHPF版全8本を並列化し，並列スーパコンSR2201により性能評価を行った．プロセッサ16台時に，HPF版の実行時間は，人手並列化版であるNPB2.1と比較して1.7倍以内であった．

Parallelization of the NAS Parallel Benchmarks by an HPF Compiler "Parallel FORTRAN"

Hiroshi Ohta,[†] Yasuhito Nishitani,[††] Atsushi Kobayashi[†††] and Eiji Nunohiro[††]

HPF (High Performance Fortran) has been proposed as a language for distributed-memory parallel supercomputers, but it is not yet widely accepted for practical use. This is because the directives are not sufficient, compiler optimizations are not yet mature, and efficient implementations in HPF are not well understood for various computations. These problems also apply to the NAS Parallel Benchmarks (NPB), a well-known benchmark set for parallel supercomputers. Only a part of the NPB has been efficiently implemented in HPF. In this study, for all the eight benchmarks of the NPB, we show the instances of the problems and the solutions for them. We reveal necessary features of a compiler and the way of HPF implementations. For language spcification, some extended directives are necessary. For compiler optimization, guard elimination and DOACROSS granurality adjustment are effective. These features are implemented in "Parallel FORTRAN," an HPF compiler we have developed. We have parallelized our HPF implementations of all the eight benchmarks of the NPB by Parallel FORTRAN and evaluated them in a parallel supercomputer SR2201. In 16 processors, the execution time of HPF code is within a factor of 1.7 compared to NPB2.1, the hand-parallelized version of the NPB.

1. はじめに

分散メモリ型の並列スーパコン（以下並列機）は，スケーラブルな高性能計算機システムとして期待されているが，一方でプログラミングが困難という問題がある．プログラミングを容易にするために，Fortran言語にデータ分散などの指示文を追加したHPF（High Performance Fortran）が提案されている．1994年までにHPF1[1]，1997年にHPF2[2]の仕様が制定され，各社が処理系を開発しているが，まだ実用化の域に十分に達してはいない．その理由としては，指示文が不足していて十分な記述ができないこと，処理系の最適化が不十分で並列化されたプログラムの性能が良くないことなどがあげられる．さらに，ユーザが行いたい

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi, Ltd.
†† 株式会社日立製作所ソフトウェア開発本部
Software Development Center, Hitachi, Ltd.
††† 日立東北ソフトウェア株式会社
Hitachi Tohoku Software, Ltd.

様々な計算に対して，HPFによる効率的な実装方法が必ずしも明らかでないという問題もあげられる．

本研究では，並列機の代表的ベンチマークであるNAS Parallel Benchmarks（NPB）を対象として上記問題を具体的に示し，それを解決するための，処理系の必要機能およびHPFによるベンチマークの実装方法を明らかにする．

NPB[3]は，米国NASAのNAS計画において作成されたベンチマークであり，計算流体力学のアプリケーションに基づいて作られた8本のプログラムから構成される．それらの一覧を**表1**に示す．

現在NPBには，NPB1とNPB2の2種類があるが，本研究は，NPB1の規則に基づくものである．NPB1は，元々単にNPBと呼ばれていたもので，“pencil and paper”形式のベンチマークである．この形式は，ベンチマークの仕様のみを規定し，その規定に従う限り，詳細アルゴリズムや実装法は自由というものである[3]．問題サイズとして，Class A，Class B，Class Cの3段階が設定されており，この順に扱う配列データが大きくなる．本研究の評価においてはClass Aを用いた．

なおNPB2は，1995年に公開されたものであり，メッセージパッシングライブラリMPI[4]を用いて人手並列化されたソースコード形式である[5),6)]．

これまでに，多くの並列機に対してNPBの性能測定結果が報告されているが，それらはほとんど人手並列化によるものである[7)]．HPF等の言語で記述しコンパイラによって並列化した例もいくつか報告されているが，数は少なく，しかも部分的なもの（NPB全8本のうち一部分のみ）である[7)~9)]．NASA自身もNPBのHPF版を作成する計画を表明しているが，現時点ではそのHPF版ソースコードは公開されていない．

本研究では，NPB全8本をHPFで記述し，我々の開発したHPF処理系Parallel FORTRANで並列化した．以下2章，3章では，この並列化における処理系の必要機能およびHPFプログラムの記述方法を述べる．これらの内容はNPBだけに特有のものではなく，他の多くの問題にも応用できるものである．4章では，並列機SR2201を用いた本HPF版の性能評価を示す．5章では，関連研究との比較を述べる．

表1 NPB一覧
Table 1 List of NPB.

名称	内容
EP (Embarrassingly parallel)	モンテカルロ法
MG (Multigrid)	マルチグリッド法
CG (Conjugate gradient)	共役勾配法
FT (3-D FFT PDE)	フーリエ変換
IS (Integer sort)	整数ソート
LU (LU solver)	流体アプリケーション
SP (Pentadiagonal solver)	流体アプリケーション
BT (Block tridiagonal solver)	流体アプリケーション

2. HPF処理系の必要機能

本章では，NPBの効率的並列化のために，HPF処理系が備えるべき機能を述べる．これらの機能はすべて，我々が開発したHPF処理系Parallel FORTRANに実装されている．Parallel FORTRAN（以下，本処理系）は，HPFソースプログラムを通信ライブラリを含むFortran90のSPMD（Single Program Multiple Data-stream）プログラムに変換するトランスレータである．その概要は文献10)，11)に述べられている．

2.1 拡張指示文

本処理系では，HPFの仕様を補完するためにHPF1に対して独自の拡張仕様を設けている．現在サポートされている拡張仕様は，配列のオーバラップ領域サイズを指示するOVERLAP指示文，ループ中のリダクション演算を指示するREDUCTIONオプション，および，当該手続きが他プロセッサのデータを参照しないことを指示するLOCAL_PURE指示文である．前2者についてはHPF2で類似の仕様が導入されたので，それらとの比較も述べる．

(1) OVERLAP指示文

本拡張仕様は，NPBの中ではMG，LUの2本に対して有用である．隣接計算型のプログラムを分散メモリ並列機で実行するときには，隣接点のデータを受信するためのバッファとして，各プロセッサのローカル配列の端にオーバラップ領域を設ける手法がよく用いられる[12)]．OVERLAP指示文はこの領域のサイズを指示するものである．

たとえば2次元配列aの第1次元をブロック分散して，下側に2要素分，上側に1要素分のオーバラップ領域を設けたいときには，次のように指示する．

```
!HPF$ distribute a(block,*)
!PFD$ overlap a(2:1,:)
```

ここで，!PFD$は拡張指示文のプレフィックスであり，“Parallel FORTRAN Directive”の意味である．

OVERLAP指示文は，複数手続きにまたがって参照される配列に対して特に効果がある．本指示文がないとき，HPF処理系の解析に基づいて確保されるオーバラップ領域のサイズは，手続き間で異なる可能性がある．すると，手続き呼び出し時に，配列を別の領域にコピーするなどの余分な処理を行わなければならない．OVERLAP指示文によって，ユーザはオーバラッ

プ領域サイズを手続きにまたがって一致させることができ，余分なコピー処理を回避できる．

なお，HPF2[2]でも，オーバラップ領域サイズを指示するSHADOW指示文が提案されている．HPF2の初期の案ではDISTRIBUTE指示文の一部として指示する仕様であったが，これには個々の配列ごとに異なるサイズが指示できないなどの問題があった．現在では修正されて本OVERLAP指示文とほぼ同等なものになっている．

(2) **REDUCTIONオプション**

本拡張仕様は，NPBの中ではEPに対して有用である．REDUCTIONオプションはINDEPENDENT指示文のオプションとして使用する．本オプション付きのINDEPENDENT指示文は，直後のループの各イタレーションが，総和などのリダクション計算を除けば独立であることを指示する．すなわち，各イタレーションを各プロセッサで独立に実行した後で，大域演算用の通信を行ってリダクションの結果を求めてよいことを示す．たとえば，変数Sに対する総和を除けばループが独立である場合，次のように指示する．

```
!PFD$ independent,reduction:sum(S)
                                ↑
                          総和計算対象変数
```

HPF2[2]でも，同様のREDUCTIONオプションが提案されているが，本処理系のものはリダクションの種類を指示する点が異なる．たとえば，上の例では総和を表す“sum”という指示が含まれている．一方，HPF2ではリダクションの種類を記述する代わりに，リダクション対象変数が特定の形の文（リダクション文）にしか現れてはならないという制約が設けられている．本処理系では，ユーザがリダクション計算であると分かっている限り，任意の文にリダクション対象変数が現れてよく，記述の自由度が高いという利点がある．

(3) **LOCAL_PURE指示文**

本拡張仕様は，NPBの中ではEP，FTの2本に対して有用である．HPFでは手続き呼び出しを含むループの分散を補助するために，PURE手続きが規定されている☆．PURE手続きはその呼び出しによる副作用のない手続きである．しかし，PURE手続き内で他プロセッサのデータの参照は許されており，そのための通信が必要となる可能性がある．このような手続き呼び出しを含むループ分散しようとすると，メッセージパッシング型の並列機では，send/receive対の生成が困難となる．

そこで本処理系では，他プロセッサのデータを参照しないPURE手続きとして，LOCAL_PURE手続きを規定している．LOCAL_PURE手続きはプロセッサ内で閉じて処理できる．したがって，数学関数のような組込み手続きと同様に扱うことができ，ループ分散の障害とはならない．

LOCAL_PURE手続きを利用するには，PURE手続き本体の宣言部，および呼び出し側の当該PURE手続きに対するインタフェースブロック内に，次の形式のLOCAL_PURE指示文を挿入する．

```
!PFD$ local_pure
```

2.2 最適化機能

本処理系では，効率的並列コードを生成するために各種の最適化を行っている[10),11)]．その中でNPBで特に効果のあるものを以下に述べる．

(1) **ガードのループ外追出し**

NPBのMGには次のようなループが現れる．

```
do i2=1,n
  do i1=1,n
    u(i1,i2,1) = u(i1,i2,n-1)
  enddo
enddo
```

ここで，配列uは第3次元ブロック分散とする．この2重ループの制御変数はいずれも分散次元の添字になっていないので，いずれのループも分散されない．このとき，Owner-Computes Ruleに従って単純に並列化すると，左辺配列のオーナプロセッサだけが代入文を実行するように，IF文で代入文にガードを掛けることになる．すると，ループ内の代入文実行のたびに，毎回IF文のオーバヘッドが掛かることになる．

実は上記のループでは，左辺配列の分散次元添字はループ不変であるから，オーナプロセッサもループ不変である．そこで本処理系では，ガードのIF文をループの外側に追い出し，IF文のオーバヘッドを2重ループ全体で1回だけとする．また，右辺のオーナは左辺のオーナと異なる可能性があるので通信が必要となるが，これについてもループの外側で一括して転送するコードを生成する．

上記のようなループは，一般に配列の端だけに対する処理を行うときに現れるので，本最適化はNPB以外のプログラムでも効果がある．

(2) **DOACROSS多重ループの粒度調節**

NPBのLUには，i, j, kの3方向にループ運搬依存のある次のようなループが現れる．

```
do k=2,nz-1
```

☆ Fortran95にも採用された．

```
      do j=2,ny-1
      do i=2,nx-1
        do m=1,5
          do l=1,5
            v(m,i,j,k) = v(m,i,j,k) - omega
     $       *(ldz(m,l,k,j,k)*v(l,i ,j ,k-1)
     $        +ldy(m,l,k,j,k)*v(l,i ,j-1,k )
     $        +ldz(m,l,k,j,k)*v(l,i-1,j ,k ) )
          enddo ! do l
        enddo   ! do m
          :
      enddo     ! do i
      enddo     ! do j
      enddo     ! do k
```

本処理系では，運搬依存のあるループに対して DOACROSS 型の並列化を行う．すなわち，あるプロセッサで定義された配列の値を，次々に隣接プロセッサへ転送することによって，パイプライン式に並列処理できるようにする．

コンパイラがループ運搬依存に対する通信を挿入するときは，運搬依存のレベルに対する位置に挿入するのが基本である[13]．しかし，この原則に従うと，一般に粒度が不適当になることがある．すなわち，細かい通信が大量に発生したり，逆に通信位置が最外側にあるために処理の並列性が失なわれてしまうなどの問題が生じることがある．

実は，上のループでは，i, j, k の 3 ループは完全交換可能[14]であり，ループの順序を任意に入れ換えたり，通信を任意の位置に挿入したりできる．本処理系では，完全交換可能な DOACROSS ループに対して，以下のような粒度調節を行っている．

上記のような 3 重以上のループに対しては，分散しないループを 1 つ選んで最外側に移動し，すべての通信をその内側に挿入する．また 2 重ループに対しては，タイリングを適用する．すなわち，分散しないループをストリップマイニングで 2 重化し，さらに，それが内側ループであれば外側に移動する．通信は移動後の最外側ループの内側に挿入する．タイリングの粒度は実行時間が最小になるように決定する[15]．

3. 各プログラムの HPF による実装方法

本章では，8 本のプログラムの各々に対して，今回行った HPF による実装の特徴点を述べる．

3.1 EP

EP は，モンテカルロ法のプログラムである．乱数のペアを発生させ，それを平面上の点の座標と見なす．そして，各点が平面上に設けた 10 個の領域のうちどれに属するかを求め，各領域に属する点の個数を求める，というものである．

処理の概略は，次のとおり．

```
      PARAMETER (NN=2**8, NK=2**16)
      REAL*8 X(2*NK),Q(0:9),T1,T2,T3
      計時開始
      DO 150 K=1, NN
          :
        DO 120 I=1,100
            :
          T3=RANDLC(T1,T2)    ! 乱数の種を生成
            :
120     ENDDO
      CALL VRANLC(2**NK,X,T1)
                  ! 配列 X の全要素に乱数を設定
      DO 140 I=1, NK
          :
        L=...       ! 乱数配列 X から L の値を計算
        Q(L)=Q(L)+1.0          !カウントアップ
140     ENDDO
150   ENDDO
      計時終了
```

DO 150 のイタレーションの各々で，2*NK 個の乱数を発生させて配列 X に格納する．さらに乱数を 2 個ずつペアとして NK 個の点の座標を求め，各点が属する領域の番号 L を計算し，配列要素 Q(L) をカウントアップしている．最終的に，配列 Q の各要素に，各領域に属する点の個数が求められる．

このプログラムを並列実行するのに最も自然かつ効率的な方法は，DO 150 のループを各プロセッサで分散して実行し，プロセッサごとに点の個数を求めて配列 Q の各要素に格納し，最後に Q の要素ごとに全プロセッサにわたるグローバル和を求めるというものである．以下では，このようなアルゴリズムを HPF で実装するときの課題と対策を述べる．

(1) **分散の基準となる配列がない**

Parallel FORTRAN では原則として Owner-Computes Rule (OCR) に従って処理の分散を行う．ところが，上のプログラムでは，DO 150 ループ内に，ループインデックス K を添字に持つような配列が存在しないため，OCR に基づいたループの分散ができない．

この課題の対策として，ダミー配列 DUMMY(NN) を設け，それに対して分散指示を行い，ループ内に DUMMY(K)=0.0 というダミーの代入文を挿入した．これにより，ダミー配列を基準とする OCR に従って

ループが分散される．ループボディが大きいため，この代入文によるオーバヘッドはほとんど無視できる．

(2) 手続き呼び出し

ループ内に，ユーザ定義手続きRANDLC, VRANLCの呼び出し（下線）が含まれる．呼び出しの副作用に関する情報が不明な限り，コンパイラはループを分散できない．

この課題に対しては，2.1節で述べたParallel FORTRANのLOCAL_PURE指示文を利用して，手続き呼び出しの副作用や他プロセッサデータの参照がないことが呼び出し元で分かるようにした．

(3) プライベート配列，配列要素への総和

配列XはDO 150ループの各イタレーションにつきプライベートである．しかし，これをコンパイラが知るためには，手続き間解析等の高度な解析を必要とする．また，配列Qの10個の要素に対して総和計算が行われている．しかしこれを認識するためには，少し手の込んだパターンマッチングが必要になる．

これらの課題に対しては，それぞれ，HPFで規定されているNEWオプション，およびParallel FORTRAN拡張仕様のREDUCTIONオプションを利用した．すなわち，DO 150ループに対して，次の指示を挿入した．

```
!PFD$ independent, new(X), reduction:sum(Q)
                    ↑              ↑
              プライベート変数    総和計算対象変数
```

3.2 MG

MGは，マルチグリッド法によって3次元の立方体領域内の偏微分方程式の解を求めるプログラムである．マルチグリッド法とは，格子点の密度を（1次元あたり）2倍ずつ変化させながら差分法の収束計算を行うものである．格子点の1次元あたり個数の最大値は，Class Aの場合256であり，最小値は2である．

処理の概略は，次のとおり．

```
      parameter (lm=8,nit=4)
      計時開始
      1辺が 2^lm の格子点での隣接計算
      do 20 it=1,nit
        do 50 k=lm,2,-1
         1辺が 2^k の格子点から 2^(k-1) の格子点への射影
 50     enddo
        1辺が2の格子点での隣接計算
        do 100 k=2,lm
         1辺が 2^(k-1) の格子点から 2^k の格子点への補間
         1辺が 2^k の格子点での隣接計算
100     enddo
 20   enddo
      計時終了
```

MGは基本的には隣接計算なので，格子点（配列）をブロック分散するのが自然な並列実行方法である．どの次元を分）するかについてはいくつか候補があるが，今回の実装では外側の2個の次元をブロック分散することにした．以下では，実装上のいくつかの特徴を述べる．

(1) 格子点数の少ない場合の重複実行

プログラム実行中に，格子点の数は最小で $2^3 = 8$ になる（1辺が2の立方格子）．格子点の数が少ないときにそれを分散すると，計算時間より通信時間の方が長くなり，かえって効率が落ちてしまう．そこで，格子点数が少ないときには，配列を分散せずに，処理を全プロセッサで重複実行するようにした．

(2) 射影，補間処理

格子点の密度を変えるときに，細かい格子点から粗い格子点への射影処理，その逆の補間処理が必要である．細かい格子点1個おきに粗い格子点が対応する．HPFでは，配列要素を1個おきに整列させる記述ができるので，一見これを使えばよいように思えるが，実はそのような整列関係を記述すると，一般には配列やループの上下限が複雑になったり，通信範囲の計算が複雑になったりし，効率的な並列化ができない場合が多い．

そこで今回の実装では，元の格子点における分散次元を「折り畳んで」表現することにした．すなわち，格子点上の1個の次元に対して，配列の方は2個の次元を用いて表現する．元の格子点のインデックスと配列のインデックスとを**図1**のように対応させる．このような表現により，密度の異なる格子点の間でも，配列の外側次元のインデックスは一致する．この外側次元で配列を整列させることにより，HPF処理系は効率の良い並列コードを生成できる．

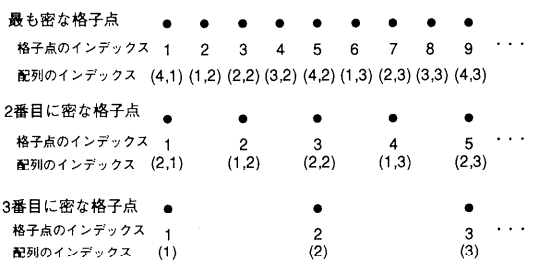


図1 MGでの格子点と配列のインデックスの関係

Fig. 1 Relationship between grid points and array indices in MG.

3.3 CG

CG は，共役勾配法（Conjugate Gradient 法）によって N×N 疎行列の固有値を求めるプログラムである．疎行列とベクトルの積の計算 $y = Ax$ が CG の核ループである．

CG での課題は，疎行列を表現するために，インデックス配列による参照が必要になることである．HPF では，インデックス配列が現れると不規則な通信が発生して大幅な性能低下を引き起こしやすい．

そこで，配列表現を工夫して，インデックス配列が分散次元の添字に現れないようにした．元の疎行列 A の各行を圧縮して，次の 3 組の配列で表現する．

`aa(N,MAXROWNZ):`	行列 A の各非ゼロ要素の値．
`colidx(N,MAXROWNZ):`	`colidx(i,j)` は非ゼロ要素 `aa(i,j)` の存在する列のインデックスを表す．
`rownz(N):`	行列 A の各行ごとの非ゼロ要素の個数．

ここで，MAXROWNZ は各行ごとの非ゼロ要素の最大個数である．例として，**図 2** に 8×8 の疎行列の表現を示す．

この表現法によると核ループは次のようになる．

```
do  i = 1,N
  do  k = 1, rownz(i)
    y(i) = y(i) + aa(i,k)*x(colidx(i,k))
  enddo
enddo
```

配列 y，aa，colidx，rownz は第 1 次元でブロック分散し，x は分散しない（全プロセッサに複製する）．こうすれば，i ループが分散され，しかも，インデックス配列を添字に持つ配列 x は分散されていないので，通信は必要ない．

なお，プログラムの他の箇所では，配列 x もブロック分散にしておく方がよいので，実際には核ループの前で，ブロック分散から複製への再分散を行っている．配列 x は 1 次元なので，この再分散のオーバヘッドはプログラム全体から見れば小さい．ただし，この再分散はスケーラビリティがないので，プロセッサ台数が多くなると，オーバヘッドが目立ってくる可能性がある．

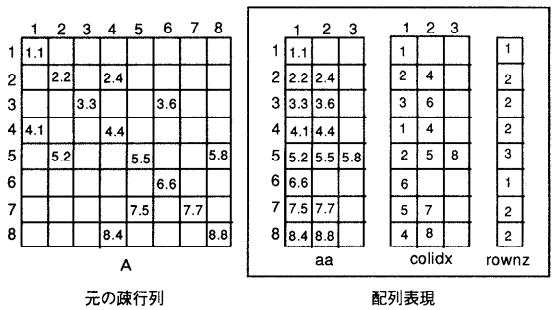


図 2 CG での疎行列の配列表現
Fig. 2 Array representation for a sparse matrix in CG.

3.4 FT

FT は，3 次元の高速フーリエ変換（FFT）である．3 次元配列の I，J，K の各方向について順次，1 次元 FFT を行う．各方向の 1 次元 FFT 処理は，その方向のみに依存関係があり他の 2 方向については独立である．したがって，各方向の 1 次元 FFT において，配列が FFT 方向について分散されていなければ，計算は各プロセッサで独立に実行でき，通信は不要である．

そこで，配列の分散は，基本的には I 方向のブロック分散とし，I 方向 FFT のときだけ J 方向のブロック分散になるように再分散した．

1 次元 FFT の部分はサブルーチンになっている．このサブルーチンは分散したいループ内で呼び出されるので，通常の場合はループ分散の障害となる．しかし，このサブルーチンは 2.1 節で述べた LOCAL_PURE 手続きの条件を満たすので，LOCAL_PURE 指示文を利用することによって，ループの分散を可能とした．

3.5 IS

IS は，N 個の整数（以下キーという）のソートである．ソートとはいっても，計時区間内の処理は，実際にキーを並べ替えるのではなく，キーのランク（下から何番目か）を求めるというものである．NPB で規定されている処理は，次のとおりである．

```
parameter (N=2**23,IMAX=10)
乱数生成ルーチンにより，N 個のキーを生成
計時開始
do i=1,IMAX
  いくつかのキーの値を変更
  全キーのランクを求める
  いくつかのキーのランクを検証
enddo
計時終了
全キーをランクに従って並べ替え，ソートされていることを検証
```

下線で示した，ランクを求める部分が IS の中心処理である．ランクを求めるアルゴリズムは任意のものを用いてよい．キーを格納する配列の初期配置は，各プロセッサに均等なブロック分割と定められている．キーのとりうる値の範囲は 0 から Bmax までである．ここで Bmax = N/10000 である．したがって，キー

の値には重複が多い．

本研究では，ランクを求めるアルゴリズムとしてradixソートの一種を用いた．この方法では，あらかじめキーの値のとりうる範囲をいくつかの区間に分割し，各区間に対して，その区間に含まれるキーを格納するための入れ物（バケットと呼ぶ）を設けておく．処理は3ステージから成る．第1ステージで，キーの値に基づいて，対応するバケットにキーを入れる．このとき，各バケット内のキーの数をカウントしておく．第2ステージで，各バケット内でのキーのランクを求める．第3ステージで，自身より下の（より小さいキーを格納している）バケット内のキーの数だけゲタを履かせることにより，全体の中でのランクが求められる．

図3 に，具体的なHPFでの実装方法を示す．本実装では，以下の配列を用いた．

`key(NROW,NBUK)`: キー

`key_buk(BUKSIZE,NBUK,NBUK)`: キーを振り分けるためのバケット

`accum(NBUK,NBUK)`: 自身より下のバケット内のキー数の積算

`rank_buk(BUKSIZE,NBUK,NBUK)`: キーのランク

ここで，NBUKは区間の数である．配列を分散する都合上，NBUKはプロセッサ数の倍数となるようにしておく．NROW=N/NBUKである．BUKSIZEはバケットのサイズであり，バケットに格納されるキーの数より大きくとっておかなければならない．配列key_bukの第2次元のインデックスがバケット番号を表す．配列key_bukの第3次元は第1ステージを分散実行するためにわざと設けた次元である．配列accumはゲタを求めるために用いる．第1次元がバケット番号を表し，第2次元は第1ステージの分散実行のためにわざと設けた次元である．

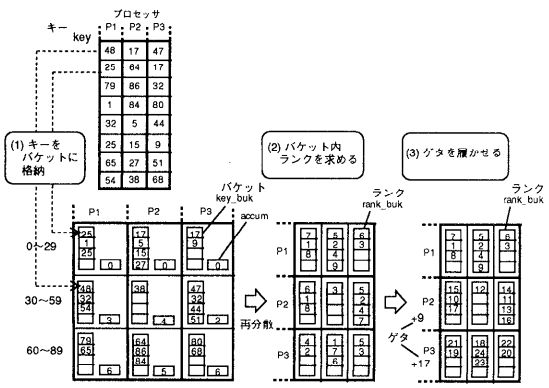


図3 ISのアルゴリズム
Fig. 3 Algorithm for IS.

第1ステージは，key，key_buk，accumを最終次元で分散しておくと，各プロセッサ内で独立に実行できる．第2，第3ステージは，key_buk，accum，rank_bukを第2次元で（すなわちバケットごとに）分散しておくと，やはり独立に実行できる．結局，第1ステージと第2ステージの間で配列key_buk，accumを再分散するだけで，後はまったく通信はない．

3.6 LU

LUは，3次元のSSOR法（Symmetric Successive Over-Relaxation法）である．この方法の特徴部分は，2.2節ですでに示したDOACROSS型ループである．このループに対しては，配列は2個の次元でブロック分散するのが望ましい．なぜなら，1個の次元で分散するのと比べて，次の2点が有利だからである．

- 切断面の面積が小さいのでデータ転送の総量が少ない．
- 全プロセッサが処理を開始するまでに要する時間が短い．

また，3個の次元で分散すると，パイプラインを稼動させるための非分散ループがなくなってしまう．したがって，今回は配列をj，kの2個の次元でブロック分散した．

このループに対する処理系の最適化は，2.2節ですでに述べたとおりである．すなわち，分散しないiループを最外側に移動し，運搬依存にともなうすべての通信をその内側に挿入する．

3.7 SP

SPの特徴部分は，3次元のADI法（Alternative Direction Implicit）法である．この方法では，i方向，j方向，k方向にループ運搬依存のあるフェーズが，交互に現れる．ある方向に運搬依存のあるフェーズでは，他の2方向については独立に実行できる．したがって，データ分散としては，基本的には最外側のk方向でブロック分散し，k方向に運搬依存があるフェーズだけは，再分散を用いてj方向ブロック分散とする．これにより，再分散以外は通信なしで分散実行できる．

3.8 BT

BTもSPと同様に3次元のADI法である．したがって，データ分散方法などはSPと同様である．

4. 評　　価

4.1 各プログラムの性能

作成したHPF版NPBをParallel FORTRANで並列化し，並列機SR2201上で性能評価した．問題サイズはClass Aである．通信ライブラリはMPIを用

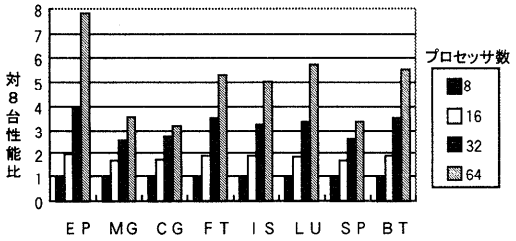


図4 SR2201 上での HPF 版 NPB 性能
Fig. 4 Performance of the HPF-version of NPB on SR2201.

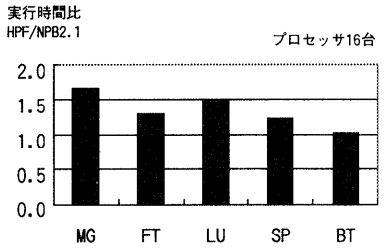


図5 HPF 版と NPB2.1 の実行時間比
Fig. 5 Execution time ratio of HPF-version and NPB2.1.

いた.

測定結果を**図4** に示す. 各プログラムについて, プロセッサ数8台時の性能を1としたときの相対性能を示している. 8台時を基準としたのは, これ以下の台数では1台あたりの使用メモリ量が大きくなるため, いくつかのプログラムが測定できないからである.

図4によると, 全プログラムにつき, 台数の増加に従って性能も向上している. 対8台の性能向上比を全8本について平均すると, 次のようになる.

16台性能/8台性能 = 1.84 (台数2倍)
32台性能/8台性能 = 3.18 (台数4倍)
64台性能/8台性能 = 4.93 (台数8倍)

これらの値は, 文献 16) に基づいて計算した人手並列化の場合の平均値とほぼ同等であり, 相対性能 (スケーラビリティ) に関しては良好な結果が得られたといえる.

次に, 絶対性能を評価するために, HPF 版と NASA 提供の NPB2.1[6] との比較を**図5** に示す (SR2201, 16台時). NPB2.1 は, MG, FT, LU, SP, BT の5本に対する, MPI 通信を用いた人手並列化版である. 図5から分かるように, HPF 版の実行時間は NPB2.1 の 1.0 倍から 1.7 倍の間に収まっている. 今後の処理系の最適化機能強化により, 人手版の性能にさらに近づいていくものと考えている.

4.2 処理系機能の効果

2章で述べた HPF 処理系の必要機能の各々につき,

表2 処理系機能の効果
Table 2 Effects of compiler features.

機能	対象	機能非サポート時の影響
OVERLAP 指示文	LU	実行時間 8%増, 使用メモリ 12%増
REDUCTION オプション	EP	主要ループ分散不能
LOCAL_PURE 指示文	EP, FT	主要ループ分散不能
ガードのループ外追出し	MG	実行時間 10 倍以上増
DOACROSS ループの粒度調節	LU	実行時間 73%増

NPB の並列化に与える効果を評価する. 以下の評価において, プロセッサ数は16台とした.

まず, 拡張指示文について述べる. OVERLAP 指示文は MG, LU の2本に対して効果があるが, 今回は LU について, OVERLAP 指示文を用いない HPF 版を作成して評価を行った. その結果, OVERLAP 指示文を用いない場合は, 用いた場合と比較して, 実行時間が 8%, 使用メモリ量が 12%増加することが分かった.

REDUCTION オプションは EP に対して効果があり, LOCAL_PURE 指示文は EP, FT に対して効果がある. これらの拡張指示文を用いない場合, EP, FT は主要ループが分散不能となり, 並列化による性能向上はまったく得られなくなってしまう.

次に, 最適化機能について述べる. ガードのループ外追出しは MG に対して効果がある. MG について, 本最適化を適用しなかった場合, すなわち, ループ内でオーナの判定と通信を行った場合の実行時間を測定した. その結果, 最適化適用時には1分以内で終了していたものが, 10 分以上経過しても終了しないことが分かった.

DOACROSS 多重ループの粒度調節は LU に対して効果がある. LU について, 本最適化を適用しなかった場合, すなわち, ループ交換を行わず, 通信を運搬依存のレベルに対する位置に挿入した場合の実行時間を測定した. その結果, 実行時間は, 最適化適用時と比較して 73%増加することが分かった.

以上の結果を**表2** にまとめて示す.

5. 関連研究

分散メモリ型 (含分散共有メモリ) の並列機に対して, コンパイラによって NPB の並列化を行った研究としては, 富士通の VPP Fortran を用いて AP1000 で評価したもの[8], Stanford 大の SUIF コンパイラを用いて DASH で評価したもの[9], および日本電気の Cenju-3 用 HPF 処理系によるもの[17]が知られている. さらに NASA から, APR 社および PGI 社の HPF コンパイラを用いて, HP 社の Exemplar, IBM

社のSP2，Cray社のT3Dの各マシンで評価した結果が報告されている[7]．

これらの報告は，8本のプログラムの一部分のみを並列化したものであり，全8本をコンパイラにより並列化した報告はこれまでにない．特に，IS，LUの2本については，従来のいずれの報告にも含まれていない．本研究では，これら2本を含む全8本の並列化を達成した．なお，各報告における性能の比較については，測定マシン環境などが異なるので，ここでは言及を差し控える．

なお，集中共有メモリ型並列機に対してのコンパイラによるNPBの並列化としては，文献18)および19)がある．これらはHPFではなく，指示文なしプログラムの自動並列化を狙ったものである．前者は，Illinois大のPolarisコンパイラを用いてSGI社のChallengeで評価したものであり，LU，SPの2本についての結果が含まれている．後者は，SUIFコンパイラを用いてDEC社のAlpha-Serverで評価したものであり，NPB全8本についての結果が示されているが，必ずしもすべてについてはスケーラビリティが得られていない．

6. おわりに

本研究では，並列スーパコンの標準的ベンチマークであるNAS Parallel Benchmarksを対象として，HPF処理系による並列化を行い，処理系の必要機能およびHPFによる実装方法を明らかにした．結論は以下である．

処理系の言語仕様上の機能としては，いくつかの拡張指示文のサポートが必要である．また処理系の最適化機能に関しては，ガードのループ外追出し，および，DOACROSSループの粒度調節が効果がある．これらの機能は，我々の開発したHPF処理系Parallel FORTRANに実装されている．

HPF版の実装方法に関しては，全8本につき実装の特徴点を示した．特に，MG，CG，ISについては，HPFに適したデータ構造やアルゴリズムを提案した．実装で用いた代表的なプログラミングテクニックは以下である．

(1) 再分散により，プログラム内の各部分に最適な分散を適用する（FT，IS，SP，BT），
(2) 小さい配列の処理においては，重複実行により通信オーバヘッドを回避する（MG），
(3) インデックス配列等の複雑な添字は，非分散次元のみに現れるようにする（CG）．

Parallel FORTRANによって全8本を並列化し，並列スーパコンSR2201により性能評価を行った．スケーラビリティに関しては，プロセッサ8台を基準として，台数を2倍，4倍，8倍にしたときの性能向上が，8本の平均で，それぞれ1.84倍，3.18倍，4.93倍であった．絶対性能に関しては，プロセッサ16台時に，NASA提供の人手並列化版であるNPB2.1と比較して，HPF版の実行時間は1.7倍以内であった．

一方で，HPF処理系にはまだ改善の余地が多く残されている．たとえば，今回MGで用いた複雑な配列表現は，本来は処理系の変換技術の強化によって解決されるべきものである．また，HPF版と人手並列化版との間には依然として無視できない性能差が残っており，よりいっそうの最適化強化が必要である．さらに今後，様々な実アプリケーションへの適用を通して，処理系技術およびユーザによる利用技術を蓄積することにより．HPFは実用化へ近づいていくであろう．

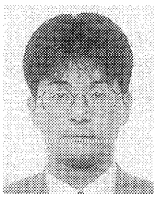
謝辞 NPBの並列化について多くの助言をいただいた（株）日立製作所汎用コンピュータ事業部の米村崇氏，HPF版NPBの性能チューニングにご尽力いただいた同ソフトウェア開発本部の根岸清氏および日立東北ソフトウェア株式会社の黒澤隆氏，そしてParallel FORTRAN開発に携わった（株）日立製作所ソフトウェア開発本部および同システム開発研究所の方々に感謝いたします．

参考文献

1) IIigh Performance Fortran Forum: *High Performance Fortran Language Specification Version 1.1 DRAFT*, Rice University, Houston, Texas (1994).
2) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0*, Rice University, Houston, Texas (1997).
3) Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V. and Weeratunga, S.: The NAS Parallel Benchmarks, RNR Technical Report, RNR-94-007, NASA Ames Research Center (1994).
4) Snir, M., Otto, S., Hess-Ledermann, S., Walker, D. and Dongarra, J.: *MPI: The Complete Reference*, MIT Press (1996).
5) Bailey, D., Harris, T., Shaphir, W., van der Wijngaart, R., Woo, A. and Yarrow, M.: The NAS Parallel Benchmarks 2.0, Report NAS-95-020, NASA Ames Research Center (1995).
6) Saphir, W., Woo, A. and Yarrow, M.: The

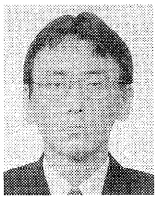
NAS Parallel Benchmarks 2.1 Results, ReportNAS-96-010, NASA Ames Research Center (1996).
7) Saini, S. and Bailey, D.H.: NAS Parallel Benchmark (Version 1.0) Results 11-96, ReportNAS-96-018, NASA Ames Research Center (1996).
8) 金城ショーン，進藤達也：VPP Fortran を用いた NAS Parallel Benchmark の並列化と AP1000 を用いた評価，情報処理学会研究報告，Vol.94, No.68, pp.27–32 (1994).
9) Tseng, C.-W., Anderson, J.M., Amarasinghe, S.P. and Lam, M.S.: Unified Compilation Techniques for Shared and Distributed Address Space Machines, *ICS'95*, ACM, pp. 67–76 (1995).
10) 吉川 聡，根岸 清，佐藤真琴，太田 寛，黒澤 隆，小林 篤，会田一弘，布広永示：High Performance Fortran トランスレータの機能概要，情報処理学会研究報告，Vol.96, No.81, pp.51–56 (1996).
11) 佐藤真琴，太田 寛，布広永示：HPF トランスレータ Parallel FORTRAN の開発と評価，情報処理，Vol.38, No.2, pp.105–108 (1997).
12) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley (1996).
13) Hiranandani, S., Kennedy, K. and Tseng, C.-W.: Evaluating Compiler Optimizations for Fortran D, *J. of Parallel and Distributed Computing*, Vol.21, pp.27–45 (1994).
14) Wolf, M.E. and Lam, M.S.: A Loop Transformation Theory and an Algorithm to Maximize Parallelism, *IEEE Trans. Parallel and Distributed Systems*, Vol.2, pp.452–471 (1991).
15) Ohta, H., Saito, Y., Kainaga, M. and Ono, H.: Optimal Tile Size Adjustment in Compiling General DOACROSS Loop Nests, *ICS'95*, ACM, pp.270–279 (1995).
16) Saini, S. and Bailey, D.H.: NAS Parallel Benchmarks Results 12-95, Report NAS-95-021, NASA Ames Research Center (1995).
17) 蒲池恒彦，末広謙二，草野和寛，妹尾義樹：Cenju-3 における HPF 処理系の開発と評価，情報処理，Vol.38, No.2, pp.109–113 (1997).
18) Blume, W., Doallo, R., Eigenmann, R., Grout, J., Hoeflinger, J., Lawrence, T., Lee, J., Padua, D., Paek, Y., Pottenger, B., Rauchwerger, L. and Tu, P.: Advanced Program Restructuring for High-Performance Computers with Polaris, Technical Report, 1473, Univ. of Illinois at Urbana Champaign, CSRD (1996).
19) Hall, M.W., Anderson, J.M., Amarasinghe, S.P., Murphy, B.R., Liao, S.-W., Bugnion, E. and Lam, M.S.: Maximizing Multiprocessor Performance with the SUIF Compiler, *Computer*, Vol.29, No.12, pp.84–89 (1996).

(平成 8 年 9 月 13 日受付)
(平成 9 年 3 月 7 日採録)



太田 寛（正会員）

1962 年生．1987 年東京大学大学院理学系研究科地球物理学専門課程修了．同年，(株) 日立製作所入社．システム開発研究所勤務．入社以来，論理型言語の研究を経て，並列化コンパイラの研究に従事．並列処理ソフトウェア全般，並列アーキテクチャに興味を持つ．電子情報通信学会，ACM，IEEE 各会員．



西谷 康仁（正会員）

1971 年生．1994 年東京大学工学部電子工学科卒業．同年，(株) 日立製作所入社．現在，同社ソフトウェア開発本部勤務．並列化コンパイラの研究開発に従事．

小林 篤（正会員）

1963 年生．1987 年北海道大学理学部数学科卒業．同年日立東北ソフトウェア (株) に入社．日立製作所中央研究所で並列化コンパイラの研究後，同社ソフトウェア開発本部にて自動ベクトル化コンパイラ，並列化コンパイラ，HPF トランスレータの開発に従事．

布広 永示（正会員）

1957 年生．1985 年日本大学大学院生産工学研究科数理工学専攻博士課程中退．工学博士．現在，(株) 日立製作所ソフトウェア開発本部勤務．数値解析の分野において特に丸め誤差解析，並列処理向き言語，開発支援システムなどに興味を持つ．日本応用数理学会会員．