ハイパフォーマンス
コンピューティング 62－10
（1996．8．28）

HPF処理系による NAS Parallel Benchmarksの並列化

太田　寛，西谷　康仁*，小林　篤**，　布広　永示*

(株)日立製作所　システム開発研究所
*(株)日立製作所　ソフトウェア開発本部
** 日立東北ソフトウェア株式会社

NAS Parallel Benchmarks (NPB) は並列型スーパーコンの代表的ベンチマークとして広く用いられている。これまでに，様々なマシンでの性能測定結果が報告されているが，それらの多くは人手並列化によるものであった。コンパイラによる並列化の報告は少なくまた部分的なものであった。本報告では，NPBの全8本をHPFで記述し，日立で開発したHPFトランスレータによって並列化した。各ベンチマークにつき，用いたソース記述法，コンパイル技術，評価などについて述べる。実機上での評価の結果，良好なスケーラビリティが得られた。

Parallelization of NAS Parallel Benchmarks by an HPF Compiler

Hiroshi OHTA, Yasuhito NISHITANI*, Atsushi KOBAYASHI**, Eiji NUNOHIRO*

Systems Development Laboratory, Hitachi Ltd.
* Software Development Center, Hitachi Ltd.
** Hitachi Tohoku Software Ltd.

NAS Parallel Benchmarks (NPB) are widely used as one of the standard benchmarks for parallel supercomputers. Performance results have been reported for various machines, but most of them are using hand-parallelized code. Reports of compiler-parallelization are few and partial. In this study we have implemented all eight benchmarks of NPB in HPF and parallelized them by an HPF compiler developed in Hitachi. We will describe the implementation technique, compiler technology, and evaluation. The results of the evaluation by SR2201 show pretty good scalability.

1. はじめに

分散メモリ型の並列スーパーコンは，スケーラブルな高性能計算機システムとして注目されているが，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，HPF(High Performance Fortran)[1]が提案され，各社がコンパイラを開発しているが，まだ実用化の域に十分に達してはいない。HPFが実用的言語として認知されるには，主要なベンチマークやアプリケーションが容易に記述でき，また効率的に並列化できることが必要であろう。

このような背景の下に本研究では，並列型スーパーコンの代表的ベンチマークとして広く用いられているNAS Parallel Benchmarks (NPB)[2]の，HPFコンパイラに

よる並列化を行った。これまでに，多くの並列型スーパーコンに対して 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[3]。コンパイラによる並列化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[4][5][6]。

今回，NPB 全 8 本を HPF で記述し，日立で開発した HPF トランスレータ(以後，本処理系と呼ぶ)で並列化した。本報告では，この並列化の際の，ソース記述法，並列化のためのコンパイル技術，そして，並列機 SR2201 を用いた性能評価などについて述べる。

2. NPB の概要

ＮＰＢ(NAS Parallel Benchmarks)は，米国の NASA Ames Research Center の NAS 計画において作成されたベンチマークである。NPB は，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。

現在 NPB には，NPB1 と NPB2 の 2 種類がある。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ソースコードの書き方は自由というものである[2]。一方 NPB2 は，1996 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 を用いて人手並列化されたソースコード形式である[7]。ソースコードに対しては僅かな変更しか許されない。本報告内容は，NPB1 の規則に基づくものである。

NPB では，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。

3. 各ベンチマークの並列化方法

本節では，8 本のベンチマークの各々に対して，ベンチマークの特徴，HPF ソースの記述法，コンパイル技法などを詳述する。

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，次の通り。

```
      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，プロセッサ毎に点の個数を求めて配列 Q の各要素に格納し，最後に Q の要素毎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，このような並列化を HPF で行うときの課題と解決方法を述べる。

(1) 分散の基準となる配列がない

本処理系では原則として Owner-Computes Rule(OCR)にしたが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った。そして，ループ内に DUMMY(K)=0.0 というダミーの代入文を挿入し，これを基準とする OCR にしたがってループが分散されるようにした。

(2) 手続き呼出し

ループ内に，ユーザ定義手続き RANDLC, VRANLC の呼出し(下線)が含まれる。これらの内容が不明な限り，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，HPF で規定されている PURE 手続きと，本処理系の拡張指示文である LOCAL_PURE 指示文を利用した。HPF の PURE 手続きは，手続き呼出しの副作用がないことを示す。さらに LOCAL_PURE 指示文は，分散されたデータの参照がないこと，すなわち，各プロセッサ内で閉じて処理できることを指示する。RANDLC や VRANLC はこれらの条件を充たしている。

このときこれらの手続きは，数学関数のような組み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。

(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なわれている。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，および本処理系の拡張指示である REDUCTION オプションを利用した。これらは，ループの各イタレーションが独立に実行できることを示す INDEPENDENT 指示文のオプションとして用いられる。NEW オプションは，変数がプライベートであることを表し，REDUCTION オプションは，変数に対しリダクション計算 が行なわれていることを表す。具体的には，挿入した指示文は次の通りである。

```
  拡張指示文のプレフィクス
    ↓
!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ベンチマークである。マルチグリッド法とは，格子点の密度を 2 倍ずつ(1 次元当り)変化させながら差分法の収束計算をおこなうものである。格子点の 1 次元当り個数の最大値は，Class A の場合 256 であり，最小値は 2 である。

処理の概略は，次の通り。

```
   parameter (lm=8,nit=4)

   計時開始
   一辺が 2^lm の格子点での隣接計算
   do 20 it=1,nit
     do 50 k=lm,2,-1
       一辺が 2^k の格子点から 2^k-1 の格子点への射影
50    enddo
      一辺が 2 の格子点での隣接計算
      do 100 k=2,lm
        一辺が 2^k-1 の格子点から 2^k の格子点への補間
        一辺が 2^k の格子点での隣接計算
        一辺が 2^k の格子点での隣接計算
100   enddo
      一辺が 2^lm の格子点での隣接計算
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。どの次元を分散するかについてはいくつか候補があるが，今回は最外側次元のみをブロック分散することにした。必ずしもこれが最適とは限らないが，様々な分散を試す前の出発点という程度の意味である。以下では，この並列実行方法を HPF で実装するときの課題と解決方法を述べる。

(1) 格子点数の少ない場合

プログラム実行中に，格子点の数は最小で 2^3=8 になる。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) 境界条件設定処理

各フェーズの計算終了後に，周期的境界条件を充たすため立方体の端から端へのコピー処理がある。例えば配列の第 3 次元についてのコピーは，次のようなループで記述される。

```
do i2=1,n
  do i1=1,n
    u(i1,i2,1) = u(i1,i2,n-1)
    u(i1,i2,n) = u(i1,i2,2)
  enddo
enddo
```

配列インデックスの 2 から n-1 までが実際の格子点に対応し，1 と n はコピー用のバッファである。

配列を第 3 次元で分散したとき，格子点の端から端へのコピーは，両端のプロセッサ同士の通信となる。通信される配列要素数は n^2 である。効率的な処理のためには，これらの配列要素が一括して転送されなければならない。

また，代入は両端のプロセッサのみで行なわれるので，代入文に対して if 文によるガードが掛かるが，その if 文がループ内に挿入されるとオーバーヘッドが大きい。if 文はループの外に出して，ループ全体がガードされるべきである。

これらの課題に対して，今回は人手によって上記ループを分割し，代入文を 1 個ずつ含む 2 個の 2 重ループにした。そして，コンパイラには，そのような 2 重ループに対してループ全体にガードを掛け，通信を一括して行う機能を組み込むことによって対応した。

(3) 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，一見これを使えば良いように思えるが，実はそのような整列関係を記述すると，一般には配列の上下限が複雑になったり，通信範囲の計算が複雑になり，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。（初めの配列インデックスが(1,1)でなく(4,1)や(2,1)になっているのは実装上の都合である。）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コンパイラは効率の良い並列コードを生成できる。

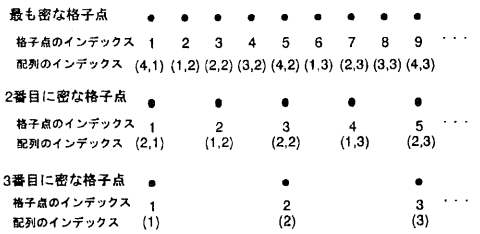

図 1　MGでの格子点と配列のインデックスの関係

(4) 配列拡張領域

MGは隣接計算なので，隣接点のデータを受信するために配列オーバーラップ領域が必要である。複数手続きにまたがって参照される配列に対しては，手続き間でオーバーラップ領域のサイズが一致している必要がある。この目的のためには，本処理系の拡張指示文であるOVERLAP指示文を利用した。

3.3 CG

CGは，共約勾配法(Conjugate Gradient法)によってN×N粗行列の固有値を求めるプログラムである。粗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，粗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。しかし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，間接参照が分散次元の添字に現れないようにした。元の粗行列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の各行毎の非ゼロ要素の個数。
```

ここで，MAXROWNZは各行毎の非ゼロ要素の最大個数である。例として，図 2に8×8の粗行列の表現を示す。

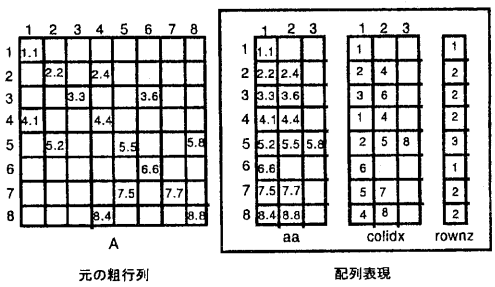

図 2　粗行列の配列表現

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(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良いので，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ー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無いので，プロセッサ台数が多くなると，オーバーヘッドが目立ってくる可能性がある。

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方向については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。I 方向を選んだのは，プログラムの他の箇所と関連した実装上の都合である。

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 EP の所で述べた LOCAL_PURE 手続きの条件を充たすので，拡張指示文の LOCAL_PURE 指示文を利用することによって，ループの分散が可能となる。

3.5 IS

IS は，N 個の整数(以下キーと言う)のソートである。ソートとは言っても，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。ランクを求めるアルゴリズムは任意のものを用いて良い。

本報告では，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取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 2 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。そして，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。

このアルゴリズムが HPF で並列化できるように，以下の表現を用いた。

```
key(NROW,NBUK)  …  キー
key_buk(BUKSIZE,NBUK,NBUK)  …  バケット
rank_buk(BUKSIZE,NBUK,NBUK)  …  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 は N/NBUK であり，BUKSIZE はバケットのサイズである。配列 key_buk，rank_buk の第 2 次元のインデックスはバケット番号を表す，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。

第 1 ステージは，key, key_buk 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 ステージは，key_buk, rank_buk を第 2 次元で(すなわちバケット毎に)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 を再分散するだけで，後は全く通信は無い。

3.6LU

LU の特徴部分は，3 次元の SOR 法(Successive Over-Relaxation 法)である。この方法では，i, j, k の 3 方向にループ運搬依存のあるループが現れる。すなわち，次のようなループである。

```
do k=2,nz-1
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  )
$        +ldx(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。

上記ループを，DOACROSS 型並列実行する場合には，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。

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。

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。しかし，この原則に従うと，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，完全交換可能性を判定して，i ループを最外側に移動し，全ての通信をその内側に挿入するという最適化を行っている。

なお，配列 v について，複数手続きにまたがってオーバーラップ領域サイズを一致させるために，MG のときと同様に OVERLAP 指示文を利用した。

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，再分散以外は通信無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. 性能評価

本処理系により並列化したNPBを，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PARALLELWARE(Express)[1]を用いた。

測定結果を図3に示す。各プログラムについて，測定可能な最小台数の場合と比較したときの性能向上比を示している。 これ以下の台数では，メモリ量の制限のため実行できなかった。

図3によると，MGを除いてかなり線形に近い性能向上となっている。 対8台の性能向上比を，MGを除く7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 ＝ 2.00 (台数2倍)
32台性能／8台性能 ＝ 3.61 (台数4倍)
64台性能／8台性能 ＝ 6.02 (台数8倍)

これらの値は，人手並列化の場合とほぼ同等であり，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言って良いであろう。

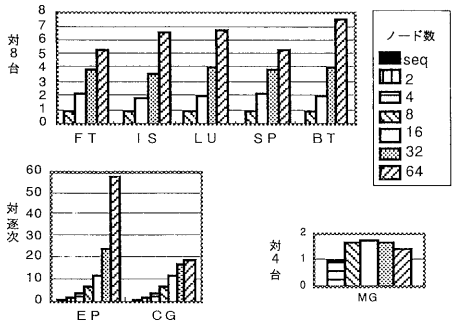

図 3 SR2201 上での NPB 測定値

MGの性能向上が良くない原因は，通信の割合が多いことである。今後，2次元分散の採用などによって，通信の割合を減らすチューニングを行っていく予定である。

なお，絶対性能に関しては，HPF版は人手並列化版と比べて全般に数倍性能が悪い。この最大の原因は，HPF版ではノード内演算の高速化のためのを全く行っていないことである。こちらについても，今後，チューニングを進めていく。

5. おわりに

並列型スーパーコンの標準的ベンチマークであるNAS Parallel BenchmarksのHPF版を作成し，日立で開発したHPFトランスレータによって，全8本を並列化した。並列機SR2201により性能評価を行った結果，良好なスケーラビリティが得られた。

今後の主な課題は，ノード内演算高速化に向けたチューニングを行い，絶対性能を向上させることである。

謝辞

NPBの並列化について多くの助言を頂いた(株)日立製作所汎用コンピュータ事業部の米村氏，また本HPF処理系の開発に携わった同ソフトウェア開発本部の方々に感謝致します。

参考文献

[1] High Performance Fortran Forum, "High Performance Fortran Language Specification (Version 1.1 DRAFT)," Rice University, Houston, Texas, 1994.

[2] D. Bailey, J. Barton, and T. Lasinski, eds., "The NAS Parallel Benchmarks," NASA Technical Memorandum 103863, NASA Ames Research Center, 1993.

[3] S. Saini and D. H. Bailey, "NAS Parallel Benchmarks Results 12-95," Report NAS-95-021, NASA Ames Research Center, 1995.

[4] 金城ショーン, 進藤達也, "VPP Fortranを用いたNAS ParallelBenchmarkの並列化とAP1000を用いた評価," 情報処理学会研究報告, 94-HPC-52, pp.27-32, 1994.

[5] C.-W. Tseng, J. M. Anderson, S. P. Amarasinghe, and M. S. Lam, "Unified Compilation Techniques for Shared and Distributed Address Space Machines," ICS'95, pp.67-76.

[6] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, P. Tu, "Advanced Program Restructuring for High-Performance Computers with Polaris," Technical Report 1473, Univ. of Illinois at Urbana Chanpaign, CSRD, 1996.

[7] D. Bailey, T. Harris, W. Saphir, R. Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," Report NAS-95-020, NASA Ames Research Center , 1995.

[1] PARALLELWAREは、新日本製鉄（株）の登録商標です。
PARALLELWAREの米国での製品名称は"Express"です。
Expressは、米国でのParasoft Corporationの商標です。