計算機アーキテクチャ 137－7
ハイパフォーマンス
コンピューティング 80－7
（2000．3．2）

並列ベクトル計算機 VPP 上の HPF の性能評価

安田計*, 岡部寿男*, 安岡孝一†, 沢田篤史†, 金澤正憲†
*京都大学大学院情報学研究科
†京都大学大型計算機センター

分散メモリ型並列ベクトル計算機 Fujitsu VPP800 では、並列計算を行うための Fortran 言語として、データ並列型言語 HPF と、データ並列に一部 SPMD 的要素を取り入れた言語である VPP Fortran が提供されている。HPF は、高い可搬性をもつデータ並列型言語として今後の期待が大きい。一方 VPP Fortran は、きめ細かいデータ転送を明示的に記述できることが特徴であり、処理速度の面でも実績がある。そこで両者の記述性や処理速度の比較を行い、HPF の有効性について評価した。

Performance of High Performace Fortranon on VPP parallel vector supercomputers

Kei Yasuda*, Yasuo Okabe*, Koichi Yasuoka†, Atsushi Sawada†, Masanori Kanazawa†
*Guraduate School of Infomatics, Kyoto University
†Data Processing Center, Kyoto University

There are two parallel Fortran languages available on Fujitsu VPP800, distributed-memory vector supercomputers. One is HPF, which is a data-parallel language, the other is VPP Fortran, which is also a data-parallel language but is partially based on the SPMD execution model. HPF has been paid attention as a standardized next-generation data-parallel language, while VPP Fortran is more descriptive in low-level data transfer and is known to generate efficient object codes. In this paper, we have compared these two languages using some benchmark programs, and evaluated the effectiveness of HPF.

1 はじめに

分散メモリ型並列計算機上で並列実行を可能にするためのプログラミングは一般的に煩雑である。ベクトル計算機や共有メモリ型並列計算機では自動ベクトル・並列化機能により従来の Fortran77/90 で書かれたプログラムにほとんど手を加えることなしにベクトル実行や並列実行を行うことが可能であるのに対し、分散メモリ型並列計算機では配列データの各プロセッサへの割付方式やプロセッサ間のデータ転送などを記述する必要が生ずる。そこで、PVM や MPI などのメッセージ通信ライブラリが多くの並列計算機に実装されいるが、これらのライブラリはデータ転送の記述を明示する必要があり、一般のユーザには利用することが難しい。そのためメッセージ通信ライブラリに比べて扱い易いデータ並列型言語が注目されている。

本大学大型計算機センターで運用されている Fujitsu VPP800 には VPP Fortran[5] と HPF (UXP/V HPF)[6] という 2 つの並列計算用 Fortran 言語が存在する。これらの言語は従来の Fortran をベースに並列実行のための機能を付加する形態をもつデータ並列型言語である。

HPF(High Performance Fortran) は Fortran 言語の拡張として作成された純粋なデータ並列型言語であり、それを VPP800 上で利用可能にしたものが UXP/V HPF である。国際的に標準化が行われており、他機種との互換性 (可搬性) が優れているという面で今後の期待も大きい。

それに対して VPP Fortran は VPP 用に Fortran 言語を独自の仕様に基づいて一部制限並びに拡張した言語である。データ並列に SPMD 実行モデル

	UXP/V HPF	VPP Fortran
並列方式	単純なデータ並列	データ並列 + SPMD 実行方式 (Spread-barrier 方式)
データの扱い	すべて global	global+local の使い分け
指示文の記述法	先頭に !HPF$	先頭に !xocl
do ループの並列化方法	do ループに対し independent 文 + on home 文で並列可能情報を提供 (owner compute rule に基づく自動並列化)	spread do 文で並列化指示
分割配列の動的再配置	○ (ただし現時点では×)	×
broadcast 転送	×	○
compile された実行ファイルの処理速度	遅い	速い
可搬性	○	×

表 1: HPF と VPP Fortran の特徴

の概念を採り入れた実行方式を用いることにより処理速度の向上を図っている。また、データ転送機能などの並列実行を最適化するための指示文も多く用意されており、きめ細かな記述が可能であるという特徴をもつ。

そこで本研究では PGI(Portland Group Inc.)[2] のベンチマークプログラムを用いることにより両言語の記述性や処理速度の比較を行い HPF の有効性について評価した。

2 HPF と VPP Fortran

表 1に HPF と VPP Fortran の特徴を記す。[1] 並列計算用言語に重要なことは次の 2 点である。

1. 分散メモリや通信を意識させずにプログラミングをできるようにすること
2. ハードウェアの性能を十分引き出して処理速度を高めることができること。

特に分散メモリ型並列計算機において 2 を実現するためには適切なデータの分散配置と処理の分割を行うことで、データアクセスの局所性を高め、プロセッサ間通信を減少させることが重要になる。しかしそのためには複雑なプログラミングが必要となりそのためには 1 と矛盾すると言う現象が起きる。

2.1 HPF

HPF[3] は、1991 年に活動を開始した HPF-F(HPF Forum) によって、当時未だ存在していなかった" 並列計算用高級言語の世界標準" を作成することを目的に、1993 年に初期バージョンである HPF1.0 が制定された。HPF では、変数をすべての PE で参照可能なグローバル変数とすることで、分散メモリであることを意識させない変数参照を可能としている。また、HPF 指示文は!HPF$を prefix とする文にすることによって従来の Fortran との親和性を高めている。また、処理の分割を処理系が自動的に行なうことによって、ユーザはデータの分割を記述するだけで並列処理のプログラムを記述できるような仕様となっている。

しかしそれを実現させるためには、処理系の自動並列化・最適化機能の実現がある程度前提となっており、UXP/V HPF はこれらの機能が現時点では不十分で、最適化のための指示行を更に付加する必要がある。ところがその並列実行を最適化するための指示文も不十分で、きめ細かな記述を行なうことができない面も多い。1997 年に制定された HPF2.0 では shadow,on,resident 文などの処理の分割に関する記述が、さらに 1998 年に日本の組織である JAHPF(Japan Association for High Performance Fortran) で制定された HPF/JA 1.0 では full shadow,reflect などの通信最適化に関する記述は拡張されたがそれでも不足の観は否めない。

2.2 VPP Fortran

VPP Fortran は VPP 用に作成された独自の Fortran 言語拡張である。以下のような特徴がある。

ローカル変数の使用　VPP Fortran ではすべての PE から参照可能なグローバル変数のほかに、他の PE からは参照できないが高速アクセスが可能なローカル変数を用いている。これらを使い分けることにより適切なデータ分散や処理分散を可能とする。

ちなみにローカル変数ともグローバル変数とも宣言されなかった変数はすべての PE のローカルメモリに同じ大きさで変数がコピーされ、個々の PE で独立して高速アクセスできるようになる。この変数のことを重複ローカル配列と呼ぶ。またこの時、各 PE 間の値の整合性を保つために unify 転送や broadcast 転送が提供されている。

SPMD 実行方式を採り入れた並列実行機能　VPP Fortran では spread do 構文を用いることで SPMD 実行方式方式を採り入れた並列実行を行うことができる。これにより処理速度の高速化が可能となる。

これらの機能によって、VPP Fortran ではきめ細かな記述を可能とし、定評ある処理速度を実現している。

2.3　本研究の目的

本研究で目指すところは、HPF のユーザへの親和性と VPP Fortran の処理性能を兼ね備えた言語の開発であり、そのために本稿では両言語の記述性並びに処理速度の評価を行うことで課題を整理した。

3　行列積計算の並列化

両言語によるベンチマークプログラムの実装を行う。PGI[2] の提供する PGI benchmark を使用した。これは 6 つの Fortran90 で書かれたベンチマークプログラムを PGI の提供する pghpf で動作するように書き換えたものである。そこで 6 つのうち行列積演算を行う matmul というプログラムを用いた。

3.1　内積型行列積演算 (ijk 型)

matmul というベンチマークプログラムでは、m 行 n 列の行列 a と n 行 p 列の行列 b の積 c

$$c = ab$$

を求める演算を行なっている。行列 c の演算部分のコードは以下の通りである。

```
do i = 1,m
 do ii = 1,p
   arow(ii)=a(i,ii)
 enddo
 do j = 1,n
  do k = 1,p
    c(i,j)=c(i,j)+arow(k) * b(k,j)
  enddo
 enddo
enddo
```

行列積演算は多重ループのインデックスの違いで 3 種類に分類されこのプログラムは内積型と呼ばれる。ここで配列 c の演算式では

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

を求めており、配列 $a \cdot b \cdot c$ はあらかじめ列方向に分割指定されている。配列 c 演算時の配列の分割方法を示した図 1から、$a(i,k)$ と $b(k,j)$ の積を求めるときにプロセッサ間通信が発生することがわかる。そこで、重複ローカル配列 $arow$ に配列 a の内容をコピーすることによって、配列 c の演算時における通信を取り除いている。

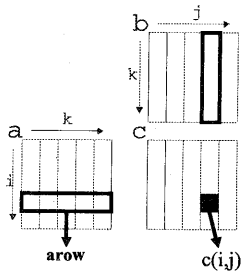

図 1: 内積型演算

UXP/V HPF には、PGI 社の pghpf に実装されている do ループの owner-computes rule に基づく自動並列化機能が実装されていないこと、並びに CPU 時間の計測関数が実装されていないことから、オリジナルの PGI HPF benchmark プログラムの do ループに対し independent・on home 文を付加し、CPU 時間計測関数の代わりにシステム実時間計測関数を用いるという変更を行うことで、UXP/V HPF 用に実装を行ない、さらに VPP Fortran でも同様の実装を行なった。

しかし、このプログラムにおいても配列 a から $arow$ へのコピー時にプロセッサ間通信が発生するので処理速度は低下する。そこで、VPP Fortran には unify 転送と言う独自のデータ転送方式があり、こちらを用いた方が通信時間を短縮できることから、unify 転送を用いたプログラムも作成した。unify 転送とは、図2のように重複ローカル配列の各演算担当 PE のデータをすべての PE へ一括転送を行うことで通信の高速化を図った方式である。

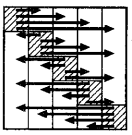

図 2: unify 転送

3.2 外積型 (kji 型)

次に多重ループのインデックスを入れ替えることで外積型の演算も行なうことにした。以下に行列 c の演算部分におけるコードを示す。

```
do k = 1,p
 do ii = 1,m
   acolumn(ii) = a(ii,k)
 enddo
 do j = 1,n
  do i = 1,m
    c(i,j)=c(i,j)+acolumn(i) * b(k,j)
  enddo
 enddo
enddo
```

各配列の分割された様子を図3に記す。

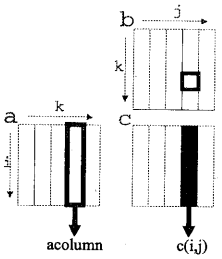

図 3: 外積型演算

内積型と同じように、配列 c の演算時において $a(i,k)$ と $b(k,j)$ の間に通信が発生するが、今回は a の演算担当 PE が 1 つに集中しており、配列 a のコピー内容も変わることからそれを明確にするためコピーするための重複ローカル名を $arow$ から $acolumn$ に代えて同様の処理を行なっている。今回もこの方法を用いた UXP/V HPF 用プログラムと VPP Fortran 用プログラムを作成した。また、内積型のときと同じように、このプログラムにおいても配列 a から $acolumn$ へのコピー時にプロセッサ間通信が発生する。そこで、今回は broadcast 転送と呼ばれる VPP Fortran 独自のデータ転送方式を用いて最適化を行なう。

broadcast 転送は図4のように重複ローカル配列の指定された PE のデータをすべての PE へ一括転送を行うことで通信の高速化を図った方式である。このとき送信側の PE から他の PE へ放送しているように見えることから broadcast と呼ぶ。

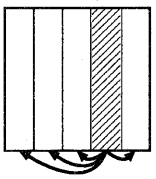

図 4: broadcast 転送

以上のことから、実装を行ったプログラムの一覧を以下にまとめる。

1. 行列積演算内積型
   - (a) 逐次型 Fortran 版
   - (b) UXP/V HPF 版
   - (c) VPP Fortran 版
   - (d) VPP Fortran 版 (unify 転送あり)
2. 行列積演算外積型
   - (a) 逐次型 Fortran 版
   - (b) UXP/V HPF 版
   - (c) VPP Fortran 版
   - (d) VPP Fortran 版 (broadcast 転送あり)

また参考として、HPF/V HPF 版 (1.b・2.b) に小修正を加えたものを逐次型 Fortran である UXP/V Fortran[4] でも評価を行うこととした。それが (1.a) と (2.a) である。

4 処理速度の評価

4.1 内積型

まず200行200列の行列積演算を行った。その結果が表2である。

言語	方式	演算時間 (msec)	処理速度 (MFLOPS)
F90	(1.a)	22.193	719.124
HPF	(1.b)	2903.258	5.497
VPP	(1.c)	407.589	39.157
	(1.d)	12.453	1281.607

表 2: 内積型行列積演算の性能評価結果 (200 × 200)

CPU数は逐次型Fortran以外はすべて10CPUで行った。演算時間は演算1回あたりの時間である。同様の条件で2001行2001列の行列積演算も行った。その結果を以下の表3に記す。

言語	方式	演算時間 (msec)	処理速度 (MFLOPS)
F90	(1.a)	5618.558	2851.267
HPF	(1.b)	318449.493	50.306
VPP	(1.c)	40458.595	395.961
	(1.d)	549.447	29156.576

表 3: 内積型行列積演算の性能評価結果 (2001 × 2001)

VPP Fortranにおいては重複ローカル配列*arow*へのコピー時にunify転送を行ったものの方が行なわなかったものに比べて処理速度は30～70倍以上改善されている。また、HPFはVPP Fortranのunify転送を用いたものより200～500倍遅く、VPP Fortranのunify転送を行なわなかったものに対しても8倍程度遅い。また、逐次型Fortranの処理速度を越えることができたのは、VPP Fortranでunify転送を用いた時のみとなっていた。

4.2 外積型

内積型と同様に200行200列と2001行2001列の行列積演算を10CPUで行った。結果を以下の表4にまとめて記す。

内積型に比して軒並み処理速度が低下しているが、内積型と同様にVPP Fortranにおいては重複ローカル配列*acolumn*へのコピー時にbroadcast転送を行った方が処理速度は7倍近く上がっている。また、HPFはVPP Fortranのbroadcast転送を用いたものより40倍近く遅く、VPP Fortran行なわなかったものよりも6倍遅い。また、逐次型Fortranに対してはVPP Fortranのbroadcast転送を用いたものが2001 × 2001の演算時に半分の処理速度を出せただけで、かなり開きがある。

配列の大きさ	言語	方式	演算時間 (msec)	処理速度 (MFLOPS)
200 × 200	F90	(2.a)	6.067	2630.758
	HPF	(2.b)	2916.386	5.473
	VPP	(2.c)	535.846	29.785
		(2.d)	84.062	189.860
2001 × 2001	F90	(2.a)	4153.413	3857.071
	HPF	(2.b)	329533.691	48.614
	VPP	(2.c)	52826.569	303.257
		(2.d)	8281.910	1934.337

表 4: 外積型行列積演算の性能評価結果

5 考察

5.1 逐次型 Fortran との比較による考察

本実験では内積・外積という違う転送方式を用いる2種類の行列積演算の評価を行ったが、HPFは10CPUを用いて計算しても、逐次型Fortranに比して格段に遅い。また、VPP Fortranでは逐次型Fortranに比べて処理速度が速かった事例を見かけたが、すべてunify転送やbroadcast転送といった、通信最適化のための命令を記述した場合だけであり、そのような記述がない場合は逐次型Fortranに比べて大分差がある。

そのため並列化処理を効率良く行なうためには、細かな命令も記述できるようにする必要性があるが、HPFでは記述性において不足している部分が多い。今回VPP Fortranで用いたunify・broadcast転送に関しても記述するための方法はUXP/V HPFには用意されていない。

5.2 HPF によるデータ転送の記述

unify転送を行うための命令文をUXP/V HPFでは持ち合わせていない。しかし、HPF/JA 1.0

仕様に存在する full shadow 命令を用いれば full shadow 本来の意味とは差異があるが、実現可能である。full shadow 指示は隣接する PE のデータを一時的に保持する領域を確保するための命令である shadow を拡張したもので、保持領域は隣接する PE だけではなくすべての PE のデータに拡げたもののことである。shadow 領域に他の PE のデータを保持させるための命令が reflect 文である。full shadow を用いて、重複ローカル配列配列 *arow* で unify 転送を指示するためのコードを図 5 に示す。

```
!HPF$ align with a(*,:),shadow(*) :: arow
...
!HPF$ independent
      do i = 1,m
!HPF$  on home(a(i,:)),local(a,arow) begin
         do ii = 1,p
           arow(ii) = a(i,ii)
         enddo
!HPF$  end on
      enddo
!HPF$ reflect( arow )
```

図 5: full shadow を用いた unify 転送の実現

ここで重複ローカル配列である *arow* を full shadow 指定したグローバル配列として宣言し、unify 転送を行いたい場所に reflect 文を挿入することで実現する。

しかし、broadcast に相当する命令は未だ存在しない。そこで本研究では、新たに broadcast 転送を行うための拡張 reflect を HPF の拡張仕様として提案する。拡張 reflect は on 文に対し clause として用いるものとして定義する。図 6に使用例を示す。

```
      do i = 1,m
!HPF$ on home(a(:,i)), local(a)
!YHV$&, reflect(arow)
!HPF$&begin
       do ii = 1,p
         arow(ii) = a(i,ii)
       enddo
!HPF$ end on
      enddo
```

図 6: 拡張 reflect を用いた broadcast 転送の実現

ここで !YHV$ とは独自の HPF 言語拡張のための prefix である。on 文に reflect 節を挿入することで home 節で指定されている配列 (例の場合 a) の存在する PE 上で演算を行い、その結果を他の全 PE 上に反映させる。HPF/JA の reflect と区別するために、この拡張 reflect は clause でのみ用いるものとし文としては用いないものとする。これによって broadcast 転送の命令を実現させる。

これらの拡張を用いることにより HPF でもきめ細かな記述が可能となり、より効率のよい並列化を行うことができるようになる。

6 おわりに

本稿では HPF と VPP Fortran との記述性ならびに処理速度の比較を行い HPF の有効性を示した。HPF の言語仕様は HPF2.0 が 1997 年に、そして HPF/JA 1.0 が 1998 年に制定されたが更なる改良の余地も多く、HPF コンパイラも未だ開発途上の段階にある。しかし並列言語の国際標準化の意義は大きく、今後の期待も高い。また当研究では独自の HPF 拡張言語を提案した。これは、HPF に大胆な変更を加えることなく、より精細な記述を認めており、現在の HPF に対して新たな概念を導入することで更によいものにできるのではないかという提案を行っている。

なお、この独自の HPF 拡張言語の処理系も設計され、VPP800 上で VPP Fortran への translator と言う形で実装されている。

参考文献

[1] 岩下英俊: "HPF からみた VPP Fortran", 情報処理学会誌第 38 巻 2 号,pp114-120,1997

[2] http://www.pgroup.com/

[3] High Performance Fortran Forum: "High Performance Fortran 2.0 公式マニュアル", シュプリンガー・フェアラーク東京,1999

[4] UXP/V Fortran 使用手引書 V20 用,J2U5-0410-02,1999

[5] UXP/V Fortran/VPP 使用手引書 V20 用,J2U5-0430-02,1999

[6] UXP/V HPF 使用手引書 V20 用,J2U5-0450-01,1999