J. Plasma Fusion Res. Vol.82, No.12 (2006) 843-846

連載コラム High Performance Fortran で並列計算を始めよう

5．作ってみよう HPF プログラム(3)

岩下英俊，林 康晴[1)]，石黒静児[2)]
富士通 ソフトウェア事業本部，[1)]NEC 第一コンピュータソフトウェア事業部，[2)]核融合科学研究所

（原稿受付：2006年11月7日）

超入門「作ってみよう」編はこれで最後となりました．今回は，HPF プログラムの中での関数とサブルーチンの考え方を解説します．

5.1 手続呼出しの種類

関数とサブルーチンを合わせて手続と呼びます．関数は式の中で使われて，式を評価するときに呼び出されます．また，サブルーチンは CALL 文で呼び出されます．これらの実行を手続呼出しといいます．

HPF の手続呼出しでは，並列実行中かそうでないかによって，呼び出せる手続の種類が違う，という点に注意が必要です．なお，並列化のネストについては，今回は考えないことにします[1]．

5.1.1 呼び出す側の状態

Fig. 1 に示すとおり，HPF プログラムの実行中には大きく 2 種類の状態があると理解してください．

- **冗長実行部**（Fig. 1 の A の部分）では，並列実行ループの外側を実行中です．重複変数への読み書きなど，全プロセッサで同じ動作を行うことが多いのでこのように呼びます．WRITE 文の出力は 1 回だけ行われるようにするなど，プロセッサ間で協調した実行が行われることもあります[2]．
- **並列実行部**（Fig. 1 の B の部分）では，並列実行ループの本体を実行中です．この範囲内で呼び出された手続の実行もまた，並列実行となります．冗長実行とは異なり，並列実行中はプロセッサ間で干渉し合うことはありません．プロセッサごとに独立に実行を進めます．

冗長実行中の手続呼出しは，冗長実行しているすべてのプロセッサに対して同時に起こります[3]．呼び出された手続の中で，並列実行することも可能です．

一方，並列実行中の手続呼出しは，プロセッサごとに独立に起こります．呼出しのタイミングも回数もプロセッサごとに違いますから，プロセッサごとに独立な手続しか呼び出すことはできません．

INDEPENDENT 指示文で並列化を指示されていない DO ループの本体は，コンパイラがループを自動並列化すれば並列実行部になり，しなければ冗長実行部になります．コンパイラは，並列化できると断定できない限り，並列化しない方を選択します．ほとんどの場合，外部手続呼出しを含むループは自動では並列化されません．

5.1.2 呼び出される手続の種類

今度は，呼び出される手続の側から考えてみます．HPF の手続には，大きく 2 種類あると考えてください．

＊グローバル手続（Fig. 1 の sub1）は，複数プロセッサで同時に呼び出されて，冗長実行で実行を開始することを想定した手続です．手続中で並列化を行うことができます．HPF では，すべてのプログラムはデフォルトではグローバルです．HPF コンパイラは，すべてのグローバル手続をできるだけ自動的に並列化します．ここで注意が必要なのは，INDEPENDENT 指示文で並列化を指示した DO ループの中でグローバル手続を呼ぶ場合（Fig. 1 の sub2 と同じ場所で sub1 が呼ばれる場合）です．呼ばれ側手続内と呼出

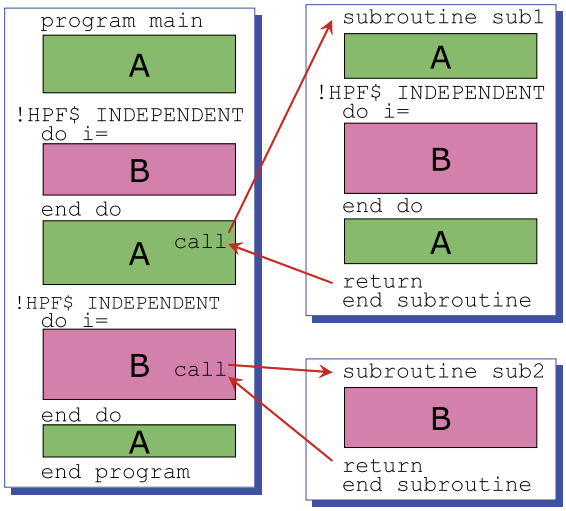


Fig. 1 手続呼出しの場所による違い．

1 ここでは全プロセッサによるフラットなループ処理の分担だけを考えます．一般には，プロセッサの仮想的な多次元構成を使えば，多次元並列化も可能です．
2 文法的に正しいプログラムである限り，「指示文を無視した逐次実行と同じ結果」が自動的に保障されます．
3 同時刻であるとは保障されませんが，プロセッサ間のデータ依存関係は保障されます．

Let Us Start Parallel Processing Using High Performance Fortran ! 5. Let Us Make HPF Program (3)
IWASHITA Hidetoshi, HAYASHI Yasuharu and ISHIGURO Seiji *corresponding author's e-mail: iwashita.hideto@jp.fujitsu.com*

し側の両方で並列化されると，誤った実行を起こす場合があります．5.3節でもう少し説明します．

＊ローカル手続（Fig.1 の sub2）は，プロセッサごとに呼び出されて，各々のデータだけを使って，他のプロセッサとは独立に実行されることを想定した手続です．つまり，それぞれのプロセッサについての逐次実行の手続だと思ってください．

このグローバルとローカルの違いを，**言語モデル**といいます．言語モデルにはもう一つ，必要なデータを1プロセッサに集めてそのプロセッサだけで実行する，シリアルモデルがあります．言語と言語モデルの組合せはいろいろ考えられますが，実用的には以下の2つで十分です．

- 冗長実行部で通常（グローバルモデル）の HPF 手続を呼び出す方法．デフォルトの動作です（5.2節）．
- 並列実行部で Fortran 手続を呼び出す方法．お勧めは，ローカルモデルの Fortran 手続です（5.3節）．

5.2 グローバル手続の呼出し

グローバル手続の間のデータの受渡しでは，COMMON 変数を使う方法と，引数で渡す方法がよく使われます．COMMON 変数の分散は，手続間で同じに宣言されていなければなりません．一方，引数の分散は，実引数（呼出し側）と仮引数（呼ばれ側）で必ずしも一致していなくても構いません．一致していないとき，HPF では処理系が自動的に通信を行って正しい実行を保証します．この仕組みを手続呼出し時の**自動再マッピング**と呼んでいます．

5.2.1 自動再マッピングの例

前回4.2節で議論した行列積のプログラムの中で，最も効率が良かった2番目の方法を Fig.2 に示しました．仮引数の配列 a，b，c のうち，b と c は DISTRIBUTE 指示文により2次元目で BLOCK 分散されています．a は分散の指示がないので，全プロセッサで全配列をもつ重複変数となります．

```
      subroutine mmul022(a,b,c,m,n,l)
      real a(m,l),b(l,n),c(m,n)
!HPF$ DISTRIBUTE (*,BLOCK) :: b,c

      c=0.0
      do j=1,n
       do i=1,m
        do k=1,l
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
        end do
       end do
      end do
      return
      end
```

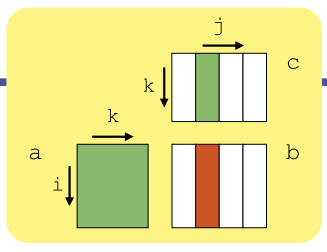


Fig. 2 プログラム例6：行列積（通信のない版）．

このサブルーチンを呼び出す側の実引数の分散はいろいろ考えられますが，2つの例を Fig.3 に示します．(a)は理想的なケースで，実引数と仮引数の分散が完全に一致しています．この場合，呼出し側の実引数 x，y，z は，コピーされることなくそのまま仮引数a，b，cとなります．(b)は実引数と仮引数の分散が一致していない場合ですが，このような呼出しも HPF では許されています．この場合，サブルーチン mmul022 の呼出し時には，実引数の配列 x，y，z から，仮引数a，b，cのために確保された領域に対して，それぞれ通信を伴うコピーが実行され，復帰時にはこの逆方向にコピーされます．これが自動再マッピングです．a，b，c に対応する領域がどこにいつ確保されるかは，実装に依存します．静的な領域かもしれませんし，スタックに積まれたり，動的に確保されるかもしれません．

(b)のプログラム中にある INTERFACE 構文は，呼出し側に呼ばれ側の仮引数の情報を伝えるための Fortran90の書式で，**明示的引用仕様**の一つです．自動再マッピングが行われる呼出しでは，明示的引用仕様が必要になります[4]．(a)のように，自動再マッピングを行わなくてよい場合には省略することもできます[5]．明示的引用仕様には，Fig.4 の(a)で示した INTERFACE 構文を使う方法や，(b)で示したモジュールを使う方法などがあります．どちらも，サブルーチン xxx の呼出しについての明示的引用仕様を書いています．前者の場合，xxx には手を加えず，xxx の宣言部を呼出し側の INTERFACE 構文の中にコピーします．後者の場合，手続 xxx を MODULE 構文で囲んでモジュール mmm の中の手続とし，呼出し側では USE 文でモジュール mmm を引用します．USE 文は INCLUDE 文を高級にしたようなもので，この宣言があると呼出し側プログラムのコンパイル時にモジュールの宣言部が参照されます．

```
      real x(n1,m),y(m,n2),z(n1,n2)
!HPF$ DISTRIBUTE (*,BLOCK) :: y,z
      ...
      call mmul022(x,y,z,n1,n2,m)
      ...
```

(a) 引数の分散が一致する場合

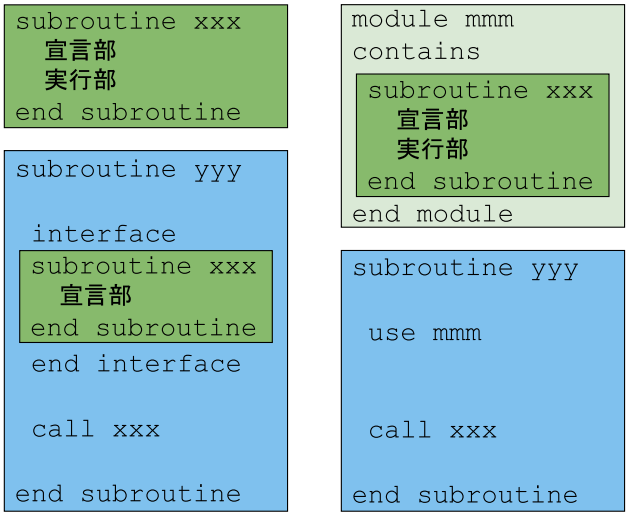
```
      real x(n1,m),y(m,n2),z(n1,n2)
!HPF$ DISTRIBUTE (BLOCK,*) :: x,y,z
      interface
       subroutine mmul022(A,B,C,m,n,l)
       real A(m,l),B(l,n),C(m,n)
!HPF$  DISTRIBUTE (*,BLOCK) :: B,C
      end interface
      ...
      call mmul022(x,y,z,n1,n2,m)
      ...
```

(b) 引数の分散が一致しない場合

Fig. 3 プログラム例6を呼び出すプログラムの例．

4 文法上は必要とされていますが，処理系によっては省略できます．再マッピングを呼ばれた手続の最初と最後に行う方式と，呼出し側で手続呼出しの直前と直後に行う方式がありますが，後者の実装では明示的引用仕様は省略できません．

5 形状引継ぎ配列を使用する場合など，Fortran90 の規約で明示的引用仕様が省略できない場合には省略できません．



(a) INTERFACE 構文を使う方法　(b) モジュールを使う方法

Fig. 4　明示的引用仕様の書き方.

5. 2. 2　手続を跨ぐデータの分散

大きな配列の再マッピングは，実行時に大きなコストがかかります．頻繁に呼び出される手続については，再マッピングを最小限にする工夫が必要です．そのためには以下のような方法が考えられます．

1．仮引数の分散を，それを頻繁に呼び出す手続の実引数に一致させる．または，実引数の分散を仮引数の分散に一致させる．— 例えば姫野ベンチHPF版[1]では，まず最も重要なサブルーチン jacobi の性能が出るような分散を考え，主プログラムと初期化ルーチンの分散はそれに合わせました．

2．自動再マッピングをうまく活用する．— 例えば ADI 法のように，サブルーチン sub1, sub2, sub3 を順番に繰返し呼び出すアルゴリズムで，sub3 だけで引数の分散が違うのであれば，主プログラムの分散は sub1 と sub 2 に合わせて，sub3 の呼出しで自動再マッピングを使うことができます．

5. 3　ローカル手続の呼出し

HPF で手続の記述言語 (HPF か Fortran か) と言語モデルを指定するには，**EXTRINSIC 接頭辞**を使います．これは指示文ではなく，SUBROUTINE 文，FUNCTION 文，MODULE 文などを拡張する書式となっています．構文は最後の Fig. 7 にまとめています．

Fortran で記述されたローカル手続の使用例を Fig. 5 に示します．呼出し側 (a) と呼ばれ側 (b) は別々にコンパイルすることができます．(a) のコンパイル時には，コンパイラはサブルーチン `foolocal` が Fortran のローカル手続であることを明示的引用仕様から知って，19行目の呼出しについて適切な引数の受渡しを行うコードを生成します．(b) のコンパイル時には，コンパイラは 1 行目の EXTRINSIC 宣言に従い，サブルーチン `foolocal` を Fortran プログラムだと思ってコンパイルします．

```
 1       program main
 2       parameter (n=100)
 3       real a(2,n), c(n)
 4 !HPF$ DISTRIBUTE a(*,BLOCK)
 5 !HPF$ DISTRIBUTE c(BLOCK)
 6
 7       interface
 8         EXTRINSIC('Fortran','LOCAL')
 9      &    subroutine foolocal(x,r)
10         real x(2),r
11         end subroutine
12       end interface
13
14 C    ---------------------
15 C      Here input a
16 C    ---------------------
17 !HPF$ INDEPENDENT
18       do i=1,n
19         call foolocal(a(:,i),c(i))
20       enddo
21 C    ---------------------
22 C      Here output c
23 C    ---------------------
24       stop
25       end program
```

(a) HPF の主プログラム

```
1       EXTRINSIC('Fortran','LOCAL')
2      & subroutine foolocal(x,r)
3       real x(2),r
4       r = sqrt(x(1)**2 + x(2)**2)
5       return
6       end subroutine
```

(b) Fortran のサブルーチン

Fig. 5　プログラム例 7 ：ローカル手続の呼出し.

並列ループ内での手続呼出しでは，引数の自動再マッピングが起こると通信のコストが大きいので，実引数の指定の方法に気をつける必要があります．同図 (a) の19行目の呼出しのように，部分配列 `A(:,i)` などを使って，受け渡す配列の範囲を絞り込み，手続内で必要なデータはすべて同一のプロセッサ上にあるようにしてください[6]．そうすれば，コンパイラはそれに合わせてループ処理の分担を決めて，ループを並列化します．

EXTRINSIC 接頭辞は言語仕様拡張なので，これを使うとそのプログラムが普通の Fortran でコンパイルできなくなるという不便があります．この例のように，入出力や分散配列の参照がない手続なら，デフォルトのグローバル手続として扱っても問題はないので，むしろその方をお勧めします[7]．(a)，(b) ともSUBROUTINE文からEXTRINSIC接頭辞を削除すれば，サブルーチン foolocal はグローバル手続と解釈されます．

5. 4　HPF の制限事項

HPF は Fortran95仕様をほぼカバーしていますが，分散配列では原理的にカバーできないものがあります．それは

6　fhpf を使う場合．V1. 4. 2以降の版をダウンロードしてご利用ください．

7　ローカル手続として宣言する必要があるかないかの線引きは実装に依存しています．なお，INDEPENDENT 指示された DO ループ内で呼び出される手続は，グローバル／ローカルにかかわらず，呼出し間にデータ依存があってはなりません．

以下のような，古い Fortran から引き継いだ仕様です．

- EQUIVALENCE 文や COMMON 文などによって，複数の名前の変数が同じメモリ領域を共有することがあります（記憶列結合）．
- 多次元配列でも配列要素の並びの順序が決まっています（順序結合）．例えば，形状(2,3)の配列 A なら，その要素は `A(1,1)`, `A(2,1)`, `A(1,2)`, `A(2,2)`, `A(1,3)`, `A(2,3)` の順に並んでいます．

これらの性質は，HPF では分散メモリ上に配列がばらばらに配置されるので維持することができません．HPF の分散配列には，以下のような制限が付きます．

- EQUIVALENCE 文中に指定できません．
- COMMON 文で宣言する場合，形状と分散がすべての出現で一致していなければなりません．
- 手続呼出しの引数にする場合には，実引数と仮引数の形状が一致していなければなりません．
- 実引数が配列要素（例えば `A(1,2)`）のとき，仮引数を配列にすることは禁止されています．
- 大きさ引継ぎ配列（`A(n,*)` など）にはなれません．

既存の Fortran プログラムからの並列化では，こういった制限事項に直面して，HPF としてコンパイルするのが難しい場合がありますが，それが並列化しなくてもよい手続なら，以下のどちらかの方法で扱うこともできます．

- EXTRINSIC 接頭辞付きの明示的引用仕様か処理系固有のオプションを使って，Fortran 手続としてコンパイルします．
- Fortran コンパイラを使ってオブジェクトやアーカイブにして，HPF プログラムと結合します．HPF の呼出し側では，Fortran 手続の呼出しであることを EXTRINSIC 接頭辞で明示します．

後者の方法は，既存の Fortran 用のライブラリなどを HPF から呼び出す場合にも使えます．

5.5 並列化の方法

Fortran プログラムからスタートして，HPF でどうやって並列化するか，その戦略を紹介して「作ってみよう」全3回の終わりとします．このとおりやればどんなプログラムでも並列化できる，とまでは言えませんが，一つの方法として参考にしてください．

1．プログラム全体の手続の呼出し関係を掴んでください．呼出しグラフを書いてみるのがよいでしょう．
2．主要変数はどれか掴んでください．それらの変数は，どの次元で分散すべきか考えます．分散種別（block や cyclic など）は，いつでも変更できるので仮決めで構いません（ここは MPI には真似のできない HPF の優れたところです）．HPF はデータの分散から並列性を抽出しますので，主要変数がうまく分散できないと並列化が進みません．どうしても分散できないようなら，そのアプリケーションは根本的に分散メモリ上での並列化には向いていないと思われます．
3．主要変数が引数かCOMMON変数として受け渡されている手続は，並列化の対象です．また，Fig. 1 で説明したように，並列ループから直接または間接に呼び出される手続は，逐次実行にすべきです．そうやって手続を，並列化すべきものと，逐次のままにしておくべきものに分けます．
4．あとは，並列化すべき手続について，コストの大きそうなところから順に，主要変数以外の変数の分散や，INDEPENDENT 指示文と ON 指示文の挿入を考えていきます．手続間での分散の不整合は，5.2.2項で説明したような方法で，できるだけ減らします．

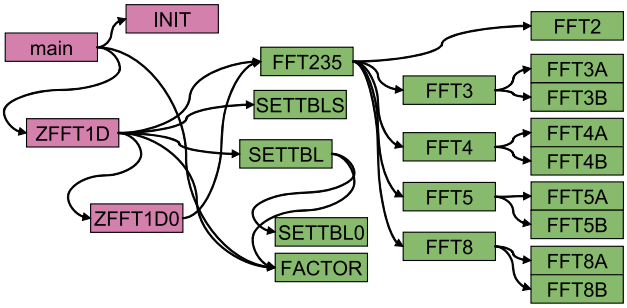


Fig. 6 FFT 1D の呼出しグラフ．

言語と言語モデルの宣言（Fortran仕様の拡張）
EXTRINSIC(*<言語>*, *<モデル>*) SUBROUTINE ...
EXTRINSIC(*<言語>*, *<モデル>*) FUNCTION ...
EXTRINSIC(*<言語>*, *<モデル>*) MODULE ...
<言語> は 'HPF' または 'FORTRAN'
<モデル> は 'GLOBAL' 'LOCAL' または 'SERIAL'
デフォルトは ('HPF', 'GLOBAL')
許される組合せは処理系による．
手続本体と明示的引用仕様に必要．

Fig. 7 今回解説した構文のまとめ．

このような方法で，1000行の Fortran プログラムを HPF で並列化した例を紹介します[2]．プログラムは1次元 FFT ffte-4.0[3]で，OpenMP で書かれていましたが，これは共有メモリ向けの並列言語なので，データの分散に関わる指示は入っていません．Fig. 6は呼出しグラフです．主要変数が A と B であることは容易にわかり，これをトリガに並列性を考えると，同図で赤い色の手続が並列化すべき手続，緑色の部分が逐次実行すべき手続とわかりました．逐次実行の手続には手を加えません．並列化すべき手続は4つで，全体のソース行数の25%程度です．これらについて，自動並列化できないループに ON 指示文を加え，その他いくつかの性能改善を加えた結果，64並列くらいまで性能が上がっていくことを確認しています．

参 考 文 献

［1］HPF 推進協議会（http://www.hpfpc.org/）．
［2］岩下英俊，岡部寿男，杉崎由典，青木正樹：LINPACK と FFT による HPF コンパイラ fhpf の生産性の評価，情処研究報告（HOKKE2006），Vol.2006, No.20, pp.67-72 (2006).
［3］FFTE: A Fast Fourier Transform Package (http://www.ffte.jp/).