FORTRAN90/SX コンパイラの特徴と使用方法

日本電気株式会社　第一コンピュータソフトウェア事業部　吉田幸正

概 要

Fortran90 言語は、1994 年に JIS Fortran 規格として制定された Fortran 言語です。

本文では、Fortran90 言語仕様 及び SX-4 で動作する FORTRAN90/SX コンパイラの特徴と使い方についてご紹介致します。

１．　はじめに

FORTRAN 言語は、1966 年に最初の規格(通称 FORTRAN66)、1978 年に二番目の規格(通称 FORTRAN77)が出版され、科学技術計算向け言語として広く使われておりますが、さらに FORTRAN を近代的な言語として発展させる目的で大幅に機能が追加され、1991 年に三番目の規格として Fortran（通称 Fortran90）が国際規格に制定されました。

これに合わせて 1994 年には JIS 規格も改正され、現在に至っております。

(この改正から、"Fortran"のように、F 以外を小文字で表記するようになりました。)

今回の改正では、単に新しい機能を追加するだけではなく、

・近代化した最新の言語へ (モジュールや手続引用仕様など)

・FORTRAN77 よりも高い生産性 (配列処理機能や行内注釈、宣言の形式追加など)

・プログラムの可搬性の向上 (種別型パラメータなど)

・言語進化の概念の導入 (廃止予定事項)

を意図した改善が図られておりますので、従来の FORTRAN とは趣の異なる言語仕様となっています。

ここでは、Fortran90 の言語仕様の特徴 及び Fortran90 規格に準拠したコンパイラ FORTRAN90/SX の特徴や使用方法について、ご説明致します。

2.　Fortran90 の言語仕様

Fortran90 規格では、従来の FORTRAN プログラムの資産を継承するために、FORTRAN77 規格の言語仕様を完全に包含しています。

本節では、FORTRAN77 との違いを中心に、Fortran90 言語の特徴を紹介します。

(1) ソースプログラムの記述形式

FORTRAN77 規格では、行番号を 1 から 5 桁目に、文を 7 から 72 桁目に書くよう定められていました。

Fortran90 規格では、132 文字までのどこに文を書いてもよい書き方が追加され、これを自由形式と呼びます。

従来の書き方も固定形式として許されていますが、固定形式はどちらかというと互換のためですから、新しいプログラムは自由形式で記述することをお勧めします。なお、自由形式では、変数名やキーワードの途中に空白を書くことができませんが、実質的にはほとんど問題とならないでしょう。

FORTRAN90/SX では、入力ソースファイルのサフィックスが.f または.F であるとき、このソースファイルを固定形式であると判断し、サフィックスが.f90 または.F90 であるとき、自由形式であると判断してコンパイル処理を行います。もちろん、コンパイル時オプション -f0 や -f4 を指定することによって、サフィックスとは無関係に、ソースプログラムの形式を指示することもできます。

このほか、Fortran90 では、!以降に注釈を記述できる行内注釈、新しい宣言記述形式などが、導入されています。

例１　　自由形式のソースプログラム

```
A = B * C + D &          ! 行末の&で、行が継続することを示します
  / E - F
```

```
10 FORMAT (I5,F8.5)      ! 文番号はどこにでも書けます。また、
                         ! キーワードに空白を書くことはできません
```

例 2　固定形式のソースプログラム

```
      A = B * C + D
     1    /E - F          ! 6 桁目の文字が、継続行を表します
   10  FO RMA T(I5,F8.5) ! 文番号は 1～5 桁目に書きます
                         ! キーワードの途中に空白が書けます
```

例 3　新しい宣言の記述形式

```
REAL, DIMENSION(10,10) :: A, B   ! REAL A(10,10),B(10,10)と同等
CHARACTER (LEN=20) :: C          ! CHARACTER*20 C と同等
INTEGER, EXTERNAL FUNC           ! INTEGER FUNC
                                 ! EXTERNAL FUNC と同等
COMPLEX, PARAMETER :: D = (1.0, 2.0)
                                 ! COMPLEX D
                                 ! PARAMETER (D=(1.0,2.0))と同等
```

(2) 配列処理機能

ある配列の要素全てに対して同じ計算を行う場合、FORTRAN77 では何重もの DO ループを書かなければなりませんでした。

Fortran90 では、演算や関数の呼び出しに対して配列を指定することにより、その配列全体や配列の一部分に対する処理を、一つの文で記述することができます。

例 1　**FORTRAN77 での記述**

```
      REAL A, B, C, X, Y, Z
      DIMENSION A(10,20), B(10,20), C(10,20), X(100), Y(100), Z(100)
      DO 20 J = 1, 20
        DO 10 I = 1, 10
          A(I,J) = B(I,J) * 2.0 + SQRT( C(I,J) )
10      CONTINUE
20    CONTINUE
      DO 30 I=1,100,4
        X(I) = Y(I) - Z(I)
30    CONTINUE
```

例 2　**Fortran90 での記述**

```
REAL, DIMENSION (10,20) :: A, B, C
REAL, DIMENSION (100) :: X, Y, Z
A = B * 2.0 + SQRT( C )
X(1:100:4) = Y(1:100:4) - Z(1:100:4)
```

例 1 と例 2 のプログラムでは、同じ結果が得られます。

(3) 構造型

構造型(derived type)として、利用者が独自の型を作成することができます。
これを用いて、C 言語などと同様の構造体を表現することもできます。

例 1　構造型の定義

```
TYPE BOOK
  CHARACTER (LEN=20) :: TITLE, AUTHOR
  INTEGER :: YEAR
END TYPE
```

BOOK 型を定義します。BOOK 型は、20 文字の TITLE,AUTHOR 及び 整数の YEAR から成る構造体です。TITLE,AUTHOR,YEAR を、構造型の成分と呼びます。

例 2　構造型の宣言

```
TYPE(BOOK) :: EXAMPLE_F90
```

BOOK 型の変数として、EXAMPLE_F90 を宣言します。

例 3　構造型変数の使用

```
EXAMPLE_F90 % TITLE = 'FORTRAN90'
EXAMPLE_F90 % AUTHOR = 'Name'
EXAMPLE_F90 % YEAR = 1996
CALL SUB(EXAMPLE_F90)
```

成分選択子である%を用いて、個々の成分を参照したり、値を代入したりすることができます。ここでは、 EXAMPLE_F90 の各成分に値を代入しています。

また、サブルーチン SUB の実引数として、構造型変数 EXAMPLE_F90 を使用していますが、サブルーチン SUB 内でこの値を参照する場合には、同じ構造型を定義しておかなければなりません。このような場合には、後述のモジュールの機能を利用すると便利です。

例 4　構造体構成子

```
EXAMPLE_F90 = BOOK('FORTRAN90','Name', 1996) ! 構造体構成子
CALL SUB(EXAMPLE_F90)
```

構造型名(成分の並び)の形式をもつ構造体構成子を用いて、構造型定数を表現することもできます。例４は、例３と同様の意味をもちます。

(4) 配列の動的割り付け

FORTRAN77 では、使用する配列の大きさは、あらかじめ決めておかなければならず、実行中に領域を割り付ける方法がありませんでした。このため、使用する配列の大きさが入力データによって変化するようなプログラムでは、考えられる最大の大きさを宣言するなどしていました。

Fortran90 では、自動割付け配列 及び 割付け配列で、動的に領域を割り付けることができます。

例 1　自動割付け配列

```
SUBROUTINE SUB(N,M)
COMMON /X/ K,L
REAL,   DIMENSION(N,M) :: A
INTEGER, DIMENSION (K,L) ::  B
```

副プログラム内で宣言され、配列の大きさが手続の入口で確定する配列を自動割付け配列と呼びます。

例１の配列 A の大きさは、サブルーチン SUB の引数 N,M の値で、配列 B の大きさは共通ブロック変数 K,L の値で決定されます。これらの配列は、サブルーチンの開始時に動的に割り付けら、サブルーチンの終了時に開放されます。

例 2　割付け配列

```
REAL, ALLOCATABLE, DIMENSION(:,:) :: C
READ(*,*) N,M
ALLOCATE(C(N,M))                    ! 領域の割付け
....
DEALLOCATE (C)                      ! 領域の開放
```

ALLOCATABLE 属性を指定して宣言した配列を割付け配列と呼び、ALLOCATE 文によって、必要な大きさを割り付けることができます。

例２では、入力データ N,M で指定された大きさの配列 C を動的に割り付けています。こ

の配列は、DEALLOCATE 文の実行時に開放されます。

(5) ポインタ

一つの変数で異なる領域を指すことができれば、データの連結リストなどが簡単に表現できますので、「C 言語のポインタが FORTRAN でも使えたら」と思ったかたもいらっしゃるでしょう。

Fortran90 ではこのポインタも使用できるようになりました。ただし、C 言語のようにアドレスだけを意味するポインタではありませんので、思わぬ領域を破壊してしまうといった可能性が少なく、より安全な仕様となっています。

例えば C 言語では、ポインタは“ポインタ型”をもち、領域のアドレスを指定するもので、アドレスの計算などに直接使用できます。

しかし Fortran90 のポインタは、独立した“型”ではなく、必ず実体と結合した状態で使用する“属性”です。また、実体をもつ変数がポインタから指される場合には、TARGET 属性を指定する必要があります。

すなわち、ポインタ属性をもつ変数は、ALLOCATE 文で割り付けるか、TARGET 属性をもつ変数と結合しなければ、参照することはできませんし、許されない変数と結合しようとすれば、コンパイル時エラーとなります。

例 1　Fortran90 のポインタの説明

```
REAL, POINTER :: P1      ! ポインタ属性をもつ変数を宣言
REAL, DIMENSION (:,:), POINTER :: P2
                         ! ポインタ属性をもつ、2 次元配列を宣言
REAL, TARGET :: T1       ! 変数 T1 は、TARGET 属性をもつ
ALLOCATE (P2(10,10))     ! ① ALLOCATE 文
P2(1,1) = 1.234          ! ② 代入文
P1 => T1                 ! ③ ポインタ代入文
P1 = P2(1,1)             ! ④ 代入文
WRITE(*,*) P1            ! ⑤
WRITE(*,*) T1            ! ⑥
```

① ポインタ P2 が指す領域として、10×10 の実数型配列を動的に割り付けます。
② P2(1,1)すなわち P2 が指す領域の(1,1)要素めに、値 1.234 を代入します。
③ ポインタ代入文です。このポインタ代入文を実行すると、ポインタ P1 が、変数 T1 を指します。このような変数 T1 を指示先と呼びます。
④ P1 の指示先に、P2(1,1)の値を代入します。
P1 の指示先は③によって T1 となっていますので、④の代入文の結果、変数 T1 に値 1.234 が代入されます。
⑤ P1 の指示先の値、すなわち 1.234 が出力されます。
⑥ 変数 T1 の値、すなわち 1.234 が出力さます。

この例からおわかりのように、ポインタ変数に代入“=”を使用すると、左辺のポインタが指す先への値の代入となります。ポインタ代入“=>”を使用すると、左辺のポインタと右辺の指示先を結合することができます。

また、ポインタを使用すると、前述のようにデータの連結リストが表現できます。

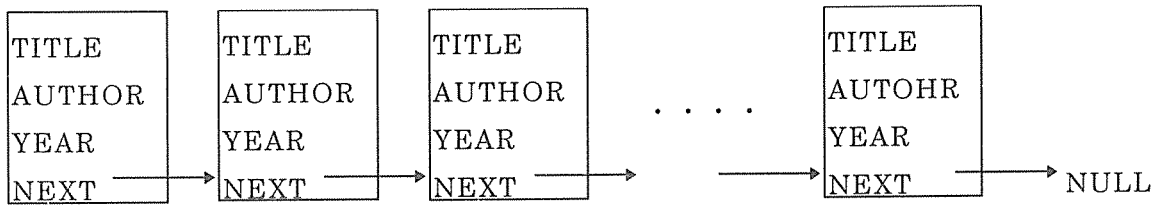
例えば、先の構造型の例を少し変更して以下のようにすることで、100 冊の本のデータの連結リストを作成することができます。

例 2　ポインタを用いた連結リストの作成

```
TYPE BOOK
  CHARACTER (LEN=20) :: TITLE, AUTHOR
  INTEGER :: YEAR
  TYPE(BOOK), POINTER :: NEXT
```

```
END TYPE BOOK
TYPE(BOOK), POINTER :: MY_BOOKS, NEW_BOOK
NULLIFY(MY_BOOKS)
DO I=1,100
   ALLOCATE (NEW_BOOK)
   READ(*,*) NEW_BOOK%TITLE
   READ(*,*) NEW_BOOK%AUTOHR
   READ(*,*) NEW_BOOK%YEAR
   NEW_BOOK%NEXT => MY_BOOKS
   MY_BOOKS => NEW_BOOK
END DO
```

このプログラムの実行の結果、
のような連結リストが作成されます。



(6) モジュール

大域的なデータの定義やデータの隠蔽(パッケージ化)に有効な手段として、モジュールが導入されています。

モジュールには宣言文やモジュール副プログラムが記述でき、これらは USE 文を用いてモジュールの外から参照できます。ただし、PRIVATE 属性を付加することによって、モジュールの外から参照できないようにすることも可能です。

例 1　宣言文だけから成るモジュールの例

```
MODULE DATA_BOOKS
 TYPE BOOK
   CHARACTER (LEN=20) :: TITLE, AUTHOR
   INTEGER :: YEAR
   TYPE(BOOK), POINTER :: NEXT
 END TYPE BOOK
 END MODULE DATA_BOOKS
```

構造型変数の使用の例で、サブルーチン SUB でも同じ構造型の定義が必要であると述べましたが、このように構造型の定義をモジュールに書いておけば、どの手続からでも

```
USE DATA_BOOKS
```

と書くことで、同じ BOOK 型の構造体を参照することができます。同じ構造型を複数の手続で参照する場合には、このようにモジュールで宣言しておくことをお勧めします。

モジュールにはまた、手続を指定することもできます。

例 2　モジュール手続の例

```
MODULE HANDLING_BOOKS
  USE DATA_BOOKS      ! 他のモジュールを USE できます
  TYPE(BOOK), PRIVATE,POINTER :: NEW_BOOK
 CONTAINS
  SUBROUTINE REGISTRATION(MY_BOOKS)
    TYPE(BOOK),POINTER :: MY_BOOKS
    ALLOCATE (NEW_BOOK)
    READ(*,*) NEW_BOOK%TITLE
    READ(*,*) NEW_BOOK%AUTOHR
    READ(*,*) NEW_BOOK%YEAR
```

```
      NEW_BOOK%NEXT => MY_BOOKS
      MY_BOOKS => NEW_BOOK
    END SUBROUTINE REGISTRATION
  END MODULE HANDLING_BOOKS
```

この例は、先ほどの連結リストを作るプログラムをモジュールとして記述したもので、

```
USE HANDLING_BOOKS
```

と書くことによって、サブルーチン REGISTRATION を呼び出して、新しい本を連結リストに登録することができます。ただし、NEW_BOOK には PRIVATE 属性が指定されているので、このモジュールの外では、NEW_BOOK を参照できません。

このようにモジュールは、宣言あるいは手続を大域的に参照できるようにしますので、モジュールを引用するプログラム単位よりも先にコンパイルする必要があります。

また、FORTRAN90/SX でモジュールをコンパイルすると、“モジュール名.mod”という名前の翻訳モジュール情報ファイルが生成され、モジュールの情報を格納します。

このため、モジュールと、そのモジュールを参照(USE)するプログラムを別のソースファイルとして記述した場合には、コンパイルする順序に注意が必要です。

例 3　モジュールのコンパイル

```
sample1.f90
        MODULE MOD1
        ....
        END MODULE MOD1
sample2.f90
        MODULE MOD2
        USE MOD1
        ....
        END MODULE MOD2
sample3.f90
        PROGRAM SAMPLE
        USE MOD2
        ....
        END PROGRAM SAMPLE
```

このようにモジュールの参照がネストしているとき、sample1.f90 → sample2.f90 → sample3.f90 の順番でコンパイルしなければなりません。また、いったんコンパイルしたあとで sample1.f90 だけを変更した場合にも、やはり 3 つのファイルを順にコンパイルし直さないと、sample1.f90 の変更が sample3.f90 のコンパイルに反映されないので、注意が必要です。

(7) 手続の多重定義

手続の多重定義を利用すると、一つの名前(総称名)で、複数の手続を呼び分けることができます。

例　手続の多重定義

```
INTERFACE RATIO                          ! INTERFACE 文
  SUBROUTINE REAL_RATIO(A, B, C)
  REAL :: A, B, C
  END SUBROUTINE REAL_RATIO
  SUBROUTINE INTEGER_RATIO(I, J, K)
  INTEGER :: I, J, K
  END SUBROUTINE INTEGER_RATIO
END INTERFACE RATIO                      ! END INTERFACE 文
REAL :: X, Y, Z
INTEGER :: L, M, N
```

```
CALL RATIO(X, Y, Z)          ! REAL_RATIO が呼び出される
CALL RATIO(L, M, N)          ! INTEGER_RATIO が呼び出される
```

このように INTERFACE 文で総称名を指定しておけば、引数の型によって、一つの名前で複数の手続を呼び出すことができます。(もちろん、サブルーチン REAL_RATIO と INTEGER_RATIO は、それぞれ別に用意しておかなければなりません。)

INTERFACE 文から、END INTERFACE 文までを手続引用仕様宣言と呼びますが、これをモジュールの中に記述しておけば、やはり USE 文を書くだけで総称名が利用できます。

(8)廃止予定事項

Fortran90 規格では、FORTRAN77 規格の仕様は全て使用できるようになっています。

しかし今後も Fortran 言語を発展させていくとき、常に新しい機能を追加するだけでは、いずれ巨大で冗長な言語仕様となってしまうでしょう。

そこで Fortran90 では、言語を進化させるという概念から、時代に合わない古い仕様を廃止する考えを取り入れました。ただし、いきなり廃止したのでは利用者が困ってしまいますから、あまり使われておらず、FORTRAN77 でもよりよい方法が存在したものについて、廃止予定事項としてあらかじめ予告して「将来廃止する可能性があるので、できるだけ使わないように」と推奨しています。

Fortran90 での廃止予定事項としては、以下のものがあります。

- 算術 IF 文
- 実数型 及び 倍精度実数型の DO 変数 及び DO ループ制御式
- DO 構文での端末文の共有 及び END DO 文でも CONTINUE 文でもない端末文
- IF ブロックの外部から END IF 文への飛び越し
- 選択戻り
- PAUSE 文
- ASSIGN 文 及び 割当て GOTO 文
- 文番号割当てによる FORMAT 文の指定
- cH 型編集記述子

```
例          f90 -Wf"-msg b"  test.f90
```

FOTRAN90/SX では、詳細オプション-msg b を指定してコンパイルすると、プログラムの中で廃止予定事項を使用している場合にメッセージが出力されます。

3． FORTRAN90/SX コンパイラの特徴と使用方法

(1) f90 コマンド

FORTRAN90/SX コンパイラを起動する f90 コマンドは、f77 コマンドに準じており、オプションの仕様も基本的には互換性を有しておりますが、コンパイラが異なるため、若干の違いがあります。

たとえば、精度自動拡張オプション"-A"や、.o をファイル単位に出力するオプション"-J"などは、f90 コマンドではサポートしておりません。

f90 コマンドのオプションの詳細につきましては、「FORTRAN90/SX プログラミングの手引」を参照してください。

(2) sxf90 コマンド（クロスコンパイラ）

FORTRAN90/SX では、NEC 48 シリーズのワークステーション上で動作するクロスコンパイラをサポートしており、ワークステーションでコンパイルとリンクを行って、実行可能形式を作成することができます。

クロスコンパイラを起動する sxf90 コマンドは、アセンブラと cpp を呼び出すことができない点を除いて、セルフコンパイラの f90 コマンドとまったく同じオプションが指定できます。

ワークステーションのFortran90コンパイラと区別するために、FORTRAN90/SXクロスコンパイラのコマンド名はf90ではなく、sxf90としてあります。
Sun, SGI, HPのワークステーション上で動作するクロスコンパイラもリリースする予定です。

(3) 最適化機能

従来のFORTRAN77/SXでは、手続のインライン展開機能や自動並列化機能など、最適化機能の一部を最適化プリプロッセサ foppによって行っていましたが、FORTRAN90/SXではコンパイラ内部でこれらの最適化を行います。
また、SX-4アーキテクチャがもつスーパスカラ機能を生かすために、分岐を越えた命令の並べ換えも行っています。

(4) ベクトル化機能

FORTRAN90/SXでも、FORTRAN77/SXと同等の、強力な自動ベクトル化機能をもっています。
すなわち、配列構文に対しても、DOループで記述したときと同様の自動ベクトル化を行います。

例　配列構文のベクトル化

```
REAL, DIMENSION (1000,1000) :: A, B, C
A = B + C                                              ! ①
A(I:J, K:L) = B(I:J, K:L) + C(I:J, K:L)                ! ②
A(1:100,1:900) = B(1:100,1:900) + C(1:100,1:900)       ! ③
```

これらの配列演算は2次元配列なので、内部的には2重のループ構造をもっています。
①は、BとCの配列全体の加算なので、ループを1重化して、配列全体をベクトル化します。
②は、配列の一部分(これを部分配列と呼びます)の演算ですが、一次元目、二次元目ともにサイズが変数で値が不明なので、一次元目をベクトル化します。
③も部分配列の演算ですが、サイズがわかりますので、ベクトル長の長い2次元目をベクトル化します。

(5) 自動並列化

コンパイル時オプションに-Pauto を指定することにより、配列構文に対してもFORTRAN77/SXと同等の自動並列化処理が行われます。

例

```
REAL, DIMENSION (100,100) :: A, B, C, D
A = SQRT(B)
C = A + D
```

このような配列構文に対して -P autoを指定してコンパイルすると、内部的には、

```
CALL SUB$1(A, B, C, D)
```

のようなサブルーチンの呼び出しと、サブルーチン

```
SUBROUTINE SUB$1(A, B, C, D)
REAL, DIMENSION (10000) :: A, B, C, D
!CDIR PARDO FOR
DO I = 1,10000                          ! 並列化されて実行
   A(I) = SQRT(B(I))
   C(I) = A(I) + D(I)
END DO
```

が生成され、ループが一重化されると同時に並列化されて実行されます。(これらは内部的なコードのイメージをFortranソースで表示したものであり、このようなソースプログラムが生成される訳ではありません。)

(6) デバッグ

dbxも、FORTRAN90/SXプログラムに対応した機能が強化されていますので、デバッグにはdbxの使用をお勧めします。

dbxに追加されている主な機能は、以下の通りです。

- print, display, dumpコマンドによって、構造型データやポインタデータの値を表示することができます。
- assignコマンドによって、構造型データやポインタデータの内容を変更することができます。
- nullifyコマンドによって、ポインタ結合を解除することができます。
- whatisコマンドによって、構造体データの型やポインタデータの型、及びPOINTER属性、PRIVATE属性等のデータ属性を表示することができます。

dbxを用いてデバッグするためには、Fortran90プログラムをコンパイルする際、-C debugオプション または -gオプションを指定してください。

例　　f90 -C debug test.f90

dbxの機能の詳細は、「DBX利用の手引」を参照ください。

(7) チューニング

FORTRAN90/SXプログラムのチューニングには、ANALYZER90/SXが使用できます。

ANALYZER90/SXは、ANALYZER-P/SXの実行時間解析機能に加えて、Fortran90で強化されたモジュール、配列演算などの解析機能を追加しています。

例　　fan90　test.f90

ANALYZER90/SXは、fan90コマンドを指定することで利用でき、手続単位、ループ単位、配列演算単位に

- CPU時間
- 実行された命令数
- ベクトル命令実行数
- 浮動小数点データ処理要素数
- メモリアクセス状況

などを測定することができます。

ANALYZER90/SXの機能の詳細は、「ANALYZER90/SX利用の手引」を参照ください。

4．　あとがき

以上、Fortran90言語仕様の特徴とFORTRAN90/SXコンパイラの特徴についてご紹介させて頂きました。

しかしFortran90言語の仕様は、ときとしてFORTRAN言語であるとは思えないほどに強化されておりますので、とても全てを説明することはできません。

さらに詳細につきましては「FORTRAN90/SX言語説明書」「FORTRAN90/SXプログラミングの手引」をご覧頂きたいと思います。

皆様がFORTRAN90/SXを使って頂くうえで、本文が、多少なりともお役に立てれば幸いです。