SX-7C FORTRAN90/SX の自動ベクトル化機能

横谷　雄司 [1]

概　要

SX-7C システムは豊富なベクトル演算命令を有しており，そのハードウェア性能を十分に引き出すためには，コンパイラの自動ベクトル化機能によりプログラムのベクトル化を行い，ベクトル命令によって処理される時間の割合(ベクトル化率)をできるだけ高くする必要がある．本文では，SX-7C システムの FORTRAN90/SX コンパイラがもつ自動ベクトル化機能について，その特長と性能向上の観点を紹介する．

1. はじめに

SX-7C システムは 8 個の CPU をもつノードを複数結合した「スケーラブル・パラレル・スーパーコンピュータ」であり，そのハードウェア性能を十分に引き出すためには，

・ 複数の CPU を効率的に使用する，**並列化**

と共に，

・個々の CPU の中で効率的に計算を行う，**ベクトル化**

が大変重要となります．

並列化につきましては別稿に譲り，本稿では，FORTRAN90/SX コンパイラがもつ**自動ベクトル化機能**についてご紹介致します．

2. 自動ベクトル化機能

SX-7C システムでのベクトル化技法は，SX-7 システムの場合と基本的には同じですが，今回はスーパーコンピュータを初めて使われる方にもご理解いただけるよう，ベクトル化の基本概念からご紹介させていただきたいと思います．

2.1 ベクトル化の基本概念

通常の演算命令は，一度に一組のデータに対する演算処理を行います．(これを，ベクトル命令と対比させるために**スカラ命令**と呼びます．)

[1] 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

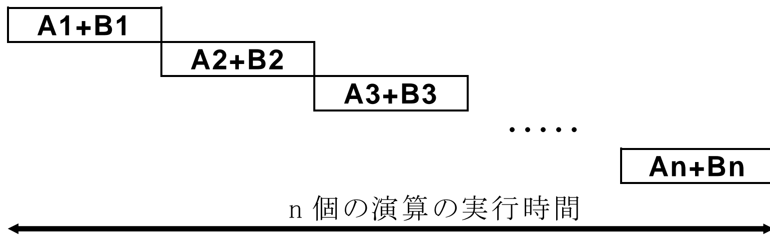

図 1 スカラ命令(スカラ加算)の実行イメージ

これに対して，**ベクトル命令**は，複数の組のデータに対する演算処理を一つの命令で一度に行うことができます．

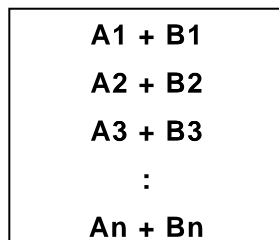

図 2 ベクトル命令(ベクトル加算)の実行イメージ

ループ中で計算される行列の要素など，規則的に並んだ配列データ(ベクトルデータ)に対してベクトル命令を適用することを**自動ベクトル化**と呼び，ベクトル化することによって高速な演算が可能となります．

2.2 自動ベクトル化の例

例えば，次の DO ループは，2 つの配列からのデータのロード，ベクトル加算，メモリへのストアの 4 つのベクトル命令で実行されます．

例 1 配列代入文と，ベクトル化されて生成される命令の概念

```
DO I=1,100
    C(I) = A(I) + B(I)
END DO
        ↓
VR1    ←  配列 A (配列Aからベクトルレジスタにデータをロード)
```

VR2　← 配列 B (配列Bからベクトルレジスタにデータをロード)

VR3　← VR1 + VR2　(ベクトル加算)

配列 C ← VR3　(配列 C に結果をストア)

VRn：ベクトルレジスタ

2.3 ベクトル化の対象範囲

FORTRAN90/SX コンパイラは, Fortran プログラムの以下の範囲を対象として自動ベクトル化を行います.

表 1　FORTRAN90/SX 自動ベクトル化対象

ベクトル化の対象となるループ	配列式, DO ループ, DO WHILE ループ, FORALL ループ, IF 文と GOTO 文によるループ
ベクトル化の対象となる文	代入文, CONTINUE 文, GOTO 文, CYCLE 文, EXIT 文, IF 文, SELECT 構文 (CALL 文, 入出力文　等は不可)
ベクトル化の対象となるデータの型	4 バイト／8 バイトの整数型・論理型 単精度／倍精度の実数型・複素数型 (文字型, 4 倍精度, 2 バイト整数型　等は不可)
ベクトル化の対象となる演算	加減乗除算, べき算, 論理演算, 関係演算, 型変換, 組込み関数 (利用者定義演算, ポインタ代入　等は不可)

詳細につきましては,「FORTRAN90/SX プログラミングの手引　5.1 ベクトル化の条件およびベクトル化の例」を参照してください.

2.4 ベクトル化可能な条件

しかし, 上記の範囲内であれば, どんなプログラムでもベクトル化できる訳ではありません. 次の例を見てください.

例 2
```
DO I=1,99
   A(I) = 2.0          ! 代入文 2-1
   B(I) = A(I+1)       ! 代入文 2-2
END DO
```

図 3 に, このプログラムをベクトル化しない場合, ベクトル化した場合の演算の実行順序を示します.

ベクトル化しない場合の実行順序	ベクトル化した場合の実行順序

```
A(1)=2.0      ! 代入文 2-1
B(1)=A(2)     ! 代入文 2-2
A(2)=2.0      ! 代入文 2-1
B(2)=A(3)     ! 代入文 2-2
                 :
A(99)=2.0     ! 代入文 2-1
B(99)=A(100)! 代入文 2-2
```

```
A(1)=2.0      ! 代入文 2-1
A(2)=2.0      ! 代入文 2-1
                 :
A(99)=2.0     ! 代入文 2-1
B(1)=A(2)     ! 代入文 2-2
B(2)=A(3)     ! 代入文 2-2
                 :
B(99)=A(100)! 代入文 2-2
```

図 3 ベクトル化の有無による実行順序の相違

例 2 のループをベクトル化して実行すると, 全ての要素に対して 2-1 の代入がベクトル命令で実行され, 次に 2-2 の代入がやはり全ての要素に対して実行されます. このため, 配列 B(1)～B(98)の値は全て 2.0 となり, プログラムの意図と異なる結果となってしまいます.

また, N 回目の繰り返しで定義したスカラ変数を N+1 回目の繰り返しで引用する, 例 3 のようなプログラムも, 全ての要素に対して 3-1 の代入が先に実行されるため, ベクトル化すると配列 A(1)～A(100)の値が全てゼロとなってしまいます.

例 3

```
S=0.0
DO I=1,100
   A(I) = S               ! 代入文 3-1 (スカラ変数 S の引用)
   S = B(I)+C(I)          ! 代入文 3-2 (スカラ変数 S の定義)
END DO
```

これらのプログラムのように, ベクトル化によって配列の定義・引用関係に変化が生じてしまう場合には, ベクトル化することができません.

FORTRAN90/SX の**自動ベクトル化機能**は, コンパイラがソースプログラムを解析して, ベクトル命令で実行できる部分を自動的に検出するとともに, 必要ならベクトル化に適合するようにプログラムを変形して, その部分に対してベクトル命令を生成します.

2.5 Fortran95 配列構文のベクトル化

例 2 のプログラムは，FORTRAN77 と同じ DO ループでしたが，Fortran95 配列構文ではどうでしょうか．

```
例 4      A(1:99)=2.0            ! 代入文 2-1
          B(1:99)=A(2:100)       ! 代入文 2-2
```

配列構文は DO ループと異なり，それぞれの文毎に全ての要素を演算することが，規格で定められています．即ち例 4 では，まず代入文 2-1 によって A(1:99)=2.0 を全て実行してから，代入文 2-2 を実行します．これはすなわち図 3 に示した「ベクトル化した場合の実行順序」と同じです．従って Fortan95 配列構文では，例 2 の様に複数の文の前後関係によってベクトル化が阻害されることはありません．(もちろん他の要因でベクトル化できないことはあります．) なお，Fortran95 配列構文については「FORTRAN90/SX 言語説明書」を参照してください．

2.6 ベクトル長

個々のベクトル命令は，演算処理に入る前にある程度の準備処理が必要となります．この準備処理に要する時間を**立ち上がり時間**と呼びます．即ち，実際のベクトル演算に要する時間が余りにも小さいと，立ち上がり時間の影響が大きくなり，ベクトル化による高速化が行えません．

ベクトル化した場合とベクトル化しない場合とで実行時間が等しくなるループの繰り返し数(**ループ長**)を**交叉ループ長**と呼び，立ち上がり時間と交叉ループ長には，図4に示す関係があります．

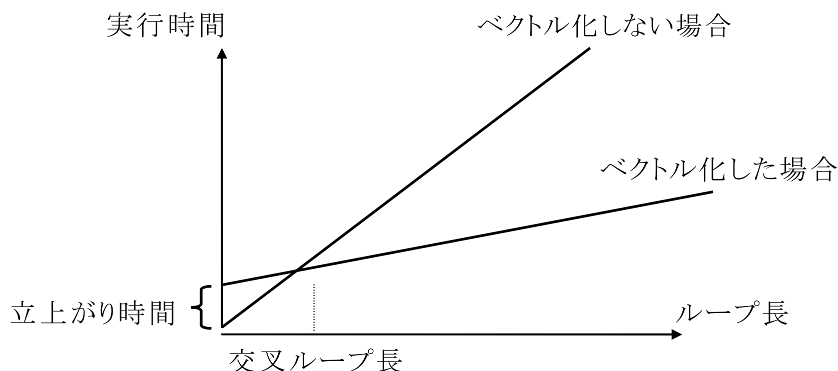

図 4 立上がり時間と交叉ループ長

この図を見ると，ループ長をできるだけ長くした方が，ベクトル化による高速化の効果が大きいことが，おわかりいただけると思います．

FORTRAN90/SX コンパイラの自動ベクトル化では，ループの繰り返し回数が 5 未満の場合には，ベクトル化による効果が少ないと判断し，ベクトル化していません．

3. 拡張ベクトル化機能

前述の通り，FORTRAN90/SX の自動ベクトル化機能は，ベクトル命令で実行可能な部分を検出しますが，そのままではベクトル化できない場合にプログラムを変形してベクトル化したり，プログラムを変形することによってベクトル化の効果をさらに高めたりします．これを**拡張ベクトル化機能**と呼びます．ここでは，FORTRAN90/SX が持つ種々の拡張ベクトル化機能のうち主なものを紹介します．

3.1 文の入れ換え

先ほどの例 2 は，「ベクトル化できない」と書きましたが，二つの文を入れ換えるとベクトル化できることに気づかれたでしょうか？

例 5 ソースプログラム

```
DO I=1,99
  A(I)=2.0        ! 代入文 5-1
  B(I)=A(I+1)     ! 代入文 5-2
END DO
```

コンパイラによる変形

```
DO I=1,99
  B(I)=A(I+1)     ! 代入文 5-2
  A(I)=2.0        ! 代入文 5-1
END DO
```

このプログラムを右のように変形すると，ベクトル化した場合でも配列の定義・引用関係が保存されますので，FORTRAN90/SXコンパイラは自動的に5-1と5-2の文を入れ換えてベクトル化を行います．

3.2 ループの一重化

ループ長が長いほどベクトル化の効果が高いとご説明しましたが，例 6 のようにに多次元配列を 1 次元配列と見なすことができる場合には，多重ループを一重ループに変形することでループ長を拡大すると同時に，外側ループの繰り返し制御の時間を削減します．

例 6

```
DIMENSION A(M,N), B(M,N), C(M,N)
    DO I=1,M
```

```
      DO J=1,N
        A(J,I) = B(J,I) + C(J,I)
      END DO
    END DO
          ↓ コンパイラによる変形のイメージ
    DO IJ=1,M*N
      A(IJ,1) = B(IJ,1) + C(IJ,1)
    END DO
```

3.3 ループの入れ換え

多重ループの場合には，通常は最も内側のループをベクトル化します．しかし，ループを入れ換えることにより定義･引用関係の矛盾が解消されてベクトル化できるようになったり，内側のループよりも外側のループの方がループ長が長く，入れ換えた方が速いと判断した場合には，コンパイラがループを入れ換えてベクトル化を行います．

例 7　配列 A に依存関係がありベクトル化できない

```
  DO J=1,M                              ! 外側ループ
    DO I=1,N                            ! 内側ループ
      A(I+1,J) = A(I,J) + B(I,J)
    END DO
  END DO
     ↓
依存関係がなくなりベクトル化できる
  DO I=1,N                              ! 元の内側ループ
    DO J=1,M                            ! 元の外側ループ
      A(I+1,J) = A(I,J) + B(I,J)
    END DO
  END DO
```

3.4 部分ベクトル化

ループ構造や配列式に，ベクトル化できる部分とベクトル化できない部分が含まれている場合，ベクトル化可能な部分と不可能な部分に分割し，可能な部分だけをベクトル化します．

例 8　4 倍精度の式が含まれているため，ベクトル化できない

```
REAL,   DIMENSION (M) :: A, B, C
REAL(KIND=16), DIMENSION (M) :: Q
A(1:M) = B(1:M) * REAL(C(1:M) * Q(1:M))
```

↓

二つの配列式に分割することで，ベクトル化する．

作業用の配列 WK は，コンパイラが用意する．

```
WK(1:M) = REAL(C(1:M) * Q(1:M)) ！ この配列代入文はベクトル化できない
A(1:M) = B(1:M)*WK(1:M)         ！ この配列代入文をベクトル化する
```

3.5 条件ベクトル化

配列の依存関係がベクトル化に適合しているかどうかコンパイル時に不明であったり，ループ長がコンパイル時に不明で，ベクトル化したほうが速いかどうかわからない場合に，ベクトル化したコードとベクトル化しないコードの両方を生成しておき，プログラムを実行するとき，そのどちらかを選択して実行するものです．

例 9　K が 0 以上であるか，または K が−10 未満ならばベクトル化しても配列 A の各要素の定義参照順序が変わらないのでベクトル化できる

```
  DO I=N, N+10
     A(I) = A(I+K) + B(I)
  END DO
       ↓     コンパイラによる変形のイメージ
IF (K .GE.0 .OR. K.LT.-10) THEN
   A(N:N+10) = A(N+K:N+10+K)+B(N:N+10)   ！ ベクトル化する
ELSE
   DO I=N, N+10                          ！ ベクトル化しない
      A(I) = A(I+K) + B(I)
   END DO
END IF
```

例 10　繰り返し回数がコンパイル時に不明なため，ベクトル化したほうが速いかどうか判断できない場合

```
A(1:N) = B(1:N) + C(1:N)
          ↓        コンパイラによる変形のイメージ
   IF (N .GE. 5)   THEN
      A(1:N) = B(1:N) + C(1:N)                 ! ベクトル化する
   ELSE
      A(1:N) = B(1:N) + C(1:N)                 ! ベクトル化しない
   END IF
```

3.6 マクロ演算の認識

次のようなパターンは，変数や配列要素が繰り返しにまたがって定義・引用されるため，本来はベクトル化できませんが，コンパイラが特別なパターンであることを認識し，専用のベクトル命令を用いることで，ベクトル化を行います．

例 11　総和: ひとつ前の繰り返しで定義したスカラ変数 S の値を，次の繰り返しで引用するため，通常はベクトル化できないが，配列 A の総和を求めるパターンであるとコンパイラが認識して，総和ベクトル命令を用いることでベクトル化する

```
DO I=1, N
   S = S + A(I)
END DO
```

例 12　漸化式: 配列Aのひとつ前の繰り返しで定義された値を次の繰り返しで参照するので，専用のベクトル命令を用いてベクトル化する

```
DO I=1, N
   A(I) = A(I-1) * B(I) + C(I)
END DO
```

備考:もし A(I-1)=A(I)*B(I)+C(I)であれば，前の繰り返しで定義した値を後の繰り返

しでは参照しないため，そのままベクトル化できる．

例 13　最大値，最小値を求める

```
DO I=1, N
    IF (XMAX .LT. X(I)) THEN
      XMAX = X(I)
    END IF
END DO
```

3.7 ループ融合

拡張ベクトル化機能ではありませんが，コンパイラは同じ形状(次元数と各次元のサイズ)を持つ複数の配列式，同じ繰り返し回数を持つ複数のループ構造を一つにまとめてベクトル化します．これを**ループ融合**と呼びます．

例 14　次の二つの配列代入文は形状が一致しているので，下の二重ループと同じように解釈，最適化されてベクトル化されます．

```
A(1:M, 1:N) = B(1:M, 1:N) + C(1:M, 1:N)
D(1:M, 1:N) = E(1:M, 1:N) * F(1:M, 1:N) + S
              ↓           コンパイラによる変形のイメージ
  DO J=1,N
    DO I=1,M
        A(I, J) = B(I, J) + C(I, J)
        D(I, J) = E (I, J) * F(I, J) + S
    END DO
  END DO
```

コンパイラは，同じ形状の配列式・ループ構造が連続していれば融合しますが，間に形状の異なる配列式・ループ構造や，他の文があると融合できません．

高速化のためには，出来るだけ同じ形状の配列式・ループ構造を連続させるようにしてください．

例 15　二つの配列代入文の間に，形状の異なる配列代入文があるために，二つ

の配列式は融合されません．このような場合には，文の順序を入れ換えて，同じ形状の配列代入が連続するように書き換えてください．

下のように書き換えることにより，ループ融合される

```
A(1:M, 1:N) = B(1:M, 1:N) + C(1:M, 1:N)
X(1:L) = 0.0
D(1:M, 1:N) = E(1:M, 1:N) * F(1:M, 1:N) + S
                    ↓
A(1:M, 1:N) = B(1:M, 1:N) + C(1:M, 1:N)
D(1:M, 1:N) = E(1:M, 1:N) * F(1:M, 1:N) + S
X(1:L) = 0.0
```

4. ベクトル化率向上のための手法

4.1 ベクトル化率

プログラムをスカラ命令だけで実行させた場合の実行時間に占める，ベクトル命令で実行可能な部分の時間の割合を**ベクトル化率**と呼びます．

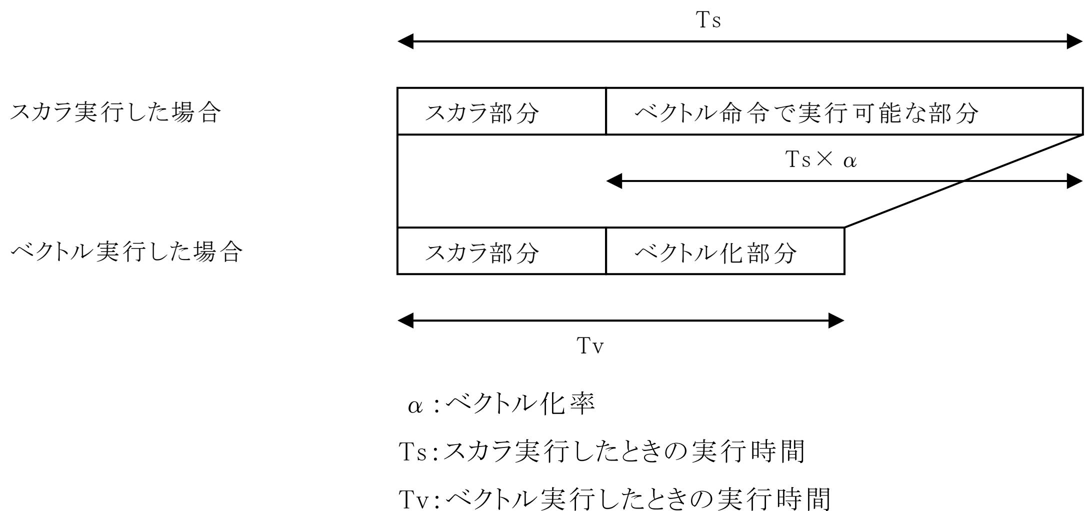

図 5 ベクトル化率

プログラムをベクトル化することにより実行性能が向上しますが，プログラムの一部だけをベクトル化しても，ベクトル化の効果はあまり期待できません．ベクトル化部分の実行時

間が非常に小さくても，ベクトル化率が50%程度では，高々2倍の性能にしかならないことは，図5からもわかると思います．一般にはベクトル化率が90～95%以上ないと，ベクトル化による大きな効果は期待できません．

すなわち，ベクトル化による高速化技法の一つは，ベクトル化率を高め，100%に近づけることにあります．

しかし，一般にベクトル化率を正確に求めることは困難であるため，SX-7Cでは，ベクトル化率に近い値として，プログラム特性情報(proginf)に表示される**ベクトル演算率**を用いています(5.1 proginf情報参照)．ベクトル演算率は，実行された命令の数をハードウェアがカウントすることで，プログラムで処理された全演算数に占める，ベクトル演算命令で処理された数の割合を求めたものです．

SX-7Cでは，この**ベクトル演算率**を高くすることを目標に，チューニングを行ってください．

ここでは，ベクトル化率(ベクトル演算率)を向上させる手法の一つとして，コンパイラ指示行についてご紹介します．

4.2 コンパイラ指示行の挿入による高速化

コンパイラは，プログラムに対して自動的に最適なベクトル命令を生成しますが，例えば変数のもつ値のように，コンパイラがプログラムを解析してもわからない情報があると，必ずしも十分なベクトル化が行われるとは限りません．

このような場合に，コンパイラが知り得ない情報を利用者が与えることにより，ベクトル化の効果を一層促進させるものが，**コンパイラ指示行**です．

コンパイラ指示行は，

```
!CDIR オプション[,オプション]
*CDIR オプション[,オプション](ソースが固定形式の場合のみ)
```

のいずれかの形式で，!CDIR，*CDIRは，1桁から5桁に書かなければなりません．

詳しくは「FORTRAN90/SXプログラミングの手引 3.2 コンパイラ指示行」を参照してください．

以下に，主な指示行のオプションとその使い方について説明します．

a) VECTOR/NOVECTOR

直後の配列式 またはDOループを自動ベクトル化の対象とする(VECTOR)か，対象と

しない(NOVECTOR)ことを指定します.

一般に VECTOR を指定する必要はありませんが，ベクトル長が小さく，ベクトル化しない方が効率の良いことがわかっていような場合に，NOVECTOR を指定します.

例 16　例えば「M は，1 又は 2 にしかなり得ない」ことを利用者が知っている場合，以下の様に NOVECTOR を指定して，ベクトル化を抑止したほうが効率がよい

```
!CDIR NOVECTOR
      A(1:M) = B(1:M) * C(1:M) + D(1:M) * E(1:M) - F(1:M) * G(1:M)
```

b) NODEP

配列の定義･引用関係がコンパイル時に不明で，自動ベクトル化できない場合に，利用者が「定義･引用関係に矛盾が無いから，ベクトル化するように」指示するものです.

例 17　NK の値が正であればベクトル化できる. 条件ベクトル化で NK の値を判断するコードが出力されるが，「NK の値が常に正である」ことを利用者が知っている場合，NODEP を指定することにより無条件にベクトル化される.

```
!CDIR NODEP
         DO I=1, N
            A(I) = A(I + NK)
         END DO
```

例 18　IP(I)の値に重複するものが無ければベクトル化できるが，もし重複するものがある場合にはベクトル化できない. 通常コンパイラにはどちらか判断できず，またこの場合には条件を判断する条件ベクトル化もできない.
もし利用者が，「IP(I)の値に重複するものがない」ことを知っている場合には，NODEP を指定することによってベクトル化することができる.

```
!CDIR NODEP
      DO I=1, N
```

```
      A( IP( I ) ) = A( IP( I )) + B( I )
   END DO
```

c) SELECT(VECTOR)

指定した DO ループをベクトル化することを指示します.

例 19 ループの一重化ができない場合, コンパイラには変数 L,M,N の値がわからないので, 通常最内側ループでベクトル化する.
しかし, 例えば L の値がとても大きく, M や N の値がごく小さいことを利用者が知っているならば, 次のように SELECT(VECTOR)指示行を挿入することにより, 右のように変形されて, ループ長の長い DO I=1,L のループでベクトル化される.

```
!CDIR SELECT(VECTOR)
      DO I = 1, L
         DO J =1, M
            DO K=1,N
               A(K, J, I)=B(K, J, I)+C(K, J, I)
            END DO
         END DO
      END DO
                 ↓ コンパイラによる変形のイメージ
      DO J = 1, M
         DO K =1, N
            DO I = 1,L             ! このループがベクトル化される
               A(K, J, I)=B(K, J, I)+C(K, J, I)
            END DO
         END DO
      END DO
```

上の例で, もし, M の値がとても大きく, L や N の値が小さいならば, DO J=1, M の直前に

```
!CDIR SELECT(VECTOR)
```

の指示行を挿入すればよいことは，もうおわかりのことと思います．

d)　SHORTLOOP

SX システムのベクトル演算命令では，一つの命令で一度に最大 256 要素のデータを処理することができます．それでは，処理したいデータの要素数が 256 個を超える場合にはどうでしょうか？この場合には次のように 256 回ずつの繰り返しを持つループ(例 21 では配列式)に分割してベクトル化を行います．

例 20　繰り返し回数が 256 を超える配列式のベクトル化の概念

```
C(1:1000) = A(1:1000) + B(1:1000)
                 ↓
DO I = 1,1000,256        ! コンパイラが生成するループ
   C(I:MIN(I+255,1000)) = A(I:MIN(I+255,1000) ) +B(I:MIN(I+255,1000))
                              ! この配列代入文は常にベクトル化される
END DO
```

このようにコンパイラが生成するループを，**ストリップマイニングループ**と呼びます．

しかし，ループ長が常に 256 以下であることがわかっていれば，このストリップマイニングループの処理が不要となりますので，このための処理時間を削減できると同時に，レジスタを効率的に使用することができます．

SHORTLOOP は，ループ長が必ず 256 以下であることを指示します．

例 21　変数 M の値が，常に 256 以下であることがわかっているならば，次の指示行を挿入することで，ストリップマイニングループを作成しないようにします．

```
!CDIR SHORTLOOP
      A(1:M) = B(1:M) + C(1:M)
```

FORTRAN90/SX コンパイラのコンパイラ指示行には，ここで紹介した他にもいろいろなオプションが指定でき，ベクトル化を制御することができます．詳細につきましては，「FORTRAN90/SX プログラミングの手引　3.2 コンパイラ指示行」をご参照ください．

5. 性能解析のツール

5.1 proginf 情報

先に述べた，ベクトル演算率を始め，プログラムの実行性能を調べる最も簡単で有効な情報がプログラム特性情報(proginf 情報)です．

proginf 情報は，プログラムの実行時に，

```
setenv F_PROGINF YES
```

または

```
setenv F_PROGINF DETAIL
```

を指定することで，表示されます(東北大学では規定値として DETAIL が設定されています)．

図 6 は，setenv F_PROGINF DETAIL で採取した proginf 情報の例ですが，ベクトル演算率(図 6 の②)，ベクトル長(図 6 の①)ともに非常に大きく，とても効率よくベクトル化されたプログラムでの例を示しています．

もしベクトル演算率が低ければ，コンパイラや後述の簡易性能解析機能の出力情報をもとに，ベクトル化できていないループ構造を抽出し，ベクトル化できるように変形したり，指示行を挿入するなどして，ベクトル演算率の向上を図ります．

次に，平均ベクトル長が短い場合には，2 重以上のループ構造を抽出し，SELECT(VECTOR)指示行を挿入してループ長の一番長いループがベクトル化されるようにすることで，ベクトル長を拡大します．また，ループ長が極端に短いループ構造がベクトル化されている場合には，NOVECTOR 指示行を挿入して，ベクトル化を行わないことも検討します．

```
      ******  Program Information  ******
   Real Time (sec)          :        23.579208        経過時間
   User Time (sec)          :        22.588967        ユーザ時間
   Sys  Time (sec)          :         0.227827        システム時間
   Vector Time (sec)        :        22.044968        ベクトル命令実行時間
   Inst. Count              :      1680849884.        全命令実行数
   V. Inst. Count           :       964802200.        ベクトル命令実行数
   V. Element Count         :    215295433316.        ベクトル命令実行要素数
   FLOP Count               :     95763953865.        浮動小数点ﾃﾞｰﾀ実行要素数
   MOPS                     :      9562.698298        MOPS 値
   MFLOPS                   :      4239.412620        MFLOPS 値
①→VLEN                     :       223.149816        平均ベクトル長
②→V. Op. Ratio (%)         :        99.668514        ベクトル演算率
   Memory Size (MB)         :       240.031250        メモリ使用量
   MIPS                     :        74.410213        MIPS 値
   I-Cache (sec)            :         0.057218        命令キャッシュミス
   O-Cache (sec)            :         0.043932        オペランドキャッシュミス
   Bank (sec)               :         0.006768        バンクコンフリクト時間
   Start Time (date)        :  2002/08/28 16:42:34    プログラムの開始日時
   End   Time (date)        :  2002/08/28 16:42:57    プログラムの終了日時
```

図 6 proginf 情報の出力例

5.2 簡易性能解析機能(ftrace)

proginf 情報では，プログラム全体のベクトル化率(ベクトル演算率)等を知ることができますが，手続き単位の性能を知ることはできません．

これに対して，簡易性能解析機能(ftrace 機能)を使用すれば，手続き単位や指定した範囲の性能情報を表示することができ，この情報を元にプログラムの性能上の問題点を調べることができます．

簡易性能解析機能の使い方を以下に示します．

1) プログラムを -ftrace オプションをつけてコンパイル・リンク

```
% sxf90 -ftrace test.f90
```

2) できた実行ファイルを実行

```
% a.out
```

3) ftrace コマンドを実行

```
% ftrace
```

このようにするだけで，第 7 図に示すように手続き単位の平均ベクトル長，ベクトル演算率などの性能情報が出力されます．

なお，3)の ftrace コマンドを実行する代わりに，2)の実行の前に環境変数 F_FTRACE に YES を設定しておくと，プログラムの実行が終了したときに，性能情報

が自動的に表示されます(東北大学では規定値としてYESが設定されています).

```
*--------------------------*
  FLOW TRACE ANALYSIS LIST
*--------------------------*
Execution : Wed Aug 28 16:42:57 2002
Total CPU : 0:00'22"569

                                                                  ベクトル演算率  平均ベクトル長
手続き名    CALL回数    実行時間                                         ↓           ↙
PROG.UNIT   FREQUENCY   EXCLUSIVE        AVER.TIME    MOPS  MFLOPS   V.OP    AVER      ........   BANK
                        TIME[sec]( % )     [msec]                    RATIO   V.LEN                CONF

calc2             800     7.594( 33.6)       9.492   9778.0   4874.4 99.72   222.3                0.0000
calc1             800     7.053( 31.2)       8.816  10119.8   4642.8 99.76   222.3                0.0000
calc3             798     5.565( 24.7)       6.973   9212.8   3827.9 99.77   222.3                0.0000
shallow             1     2.346( 10.4)    2346.201   8122.2   1998.5 98.87   232.1                0.0067
initial             1     0.007(  0.0)       6.979   6084.6   2188.4 98.93   222.4                0.0000
calc3z              1     0.005(  0.0)       5.479   3930.8      0.0 99.31   222.5                0.0000
-----------------------------------------------------------------------------------------------------tota
l           2401    22.570(100.0)      9.400    9570.7 4243.0 99.67   223.2                       0.0067
```

図 7 ftraceコマンドの出力例

より詳しい使い方については,「FORTRAN90/SXプログラミングの手引 10.3 簡易性能解析機能」をご参照ください.

6. おわりに

以上,FORTRAN90/SX コンパイラの自動ベクトル化機能を中心にご紹介させていただきました.FORTRAN90/SX コンパイラは,他にも本稿でご紹介できなかった種々のベクトル化機能を持っています.詳細につきましては,「FORTRAN90/SX プログラミングの手引」を参照してくださるようお願い致します.

皆様がSX-7CとFORTRAN90/SXを使っていただくうえで,本稿が,多少なりともお役に立てれば幸いです.

参考文献

[1]FORTRAN90/SX 言語説明書　日本電気株式会社　G1AF06

[2]FORTRAN90/SX プログラミングの手引　日本電気株式会社 G1AF07