SX－3Rのバージョンアップによる機能強化について

片山　博　*1　　藤井　等　*2　　久留　央夫　*3　　塩谷　剛　*4

１．はじめに

大阪大学大型計算機センタにおいて、１１月１日（月）よりスーパーコンピュータＳＸ－３のオペレーティングシステムＳＵＰＥＲ－ＵＸをＲ２．２からＲ３．１へバージョンアップしました。今回のバージョンアップに伴い、プログラムの最適化、自動ベクトル化等の従来機能の強化、及び日本語機能（日本語入力、メッセージの日本語化等）等を新規サポートします。

本稿では、

・ＦＯＲＴＲＡＮの最適化
・ＦＯＲＴＲＡＮの自動ベクトル化
・ＦＯＲＴＲＡＮプログラムのチューニング手順
・日本語入力の為の個人環境設定

について記載します。

２．概要

FORTRAN77/SX は、SX-2N の FORTRAN77/SX をベースに、機能・性能の両面で大幅な強化を行った FORTRAN 77 処理系です。以下に、主な強化項目を示します。なお、それぞれの機能については、「SX-3R シリーズの言語プロセッサと開発支援ツール（阪大センターニュース Vol.22 No.3 1992-11 第87号）」または関連説明書をご参照ください。

(1) 言語仕様に関する強化項目

① Fortran 90 配列処理機能のサポート

Fortran 90 で追加された配列構文や配列組込み関数などを使用することができる。これにより、ループを使用しないでも、配列に対する演算を直接記述することができる。

② 行内注釈のサポート

感嘆符（！）で区切ることによって、行の終わりに注釈を記述することができる。

(2) 最適化に関する強化項目

① 手続きのインライン展開機能

関数やサブルーチン副プログラムを呼び出し元に直接展開することができる。これにより、手続き呼び出しに必要な時間が減少するばかりでなく、手続き呼び出しを含むループをベクトル化することが可能になる。

② ループのアンローリング機能

アンローリング、すなわち DO ループの繰り返し回数を 1/n し、かわりに、ループの中身をｎ段に展開する処理を自動的に行うことができる。アンローリングは、最内側のループに対してだけでなく、外側のループに対して行うことも可能である。これにより、ループ中に並列に実行可能な演算が増加したり、同一要素に対する複数の参照を共通式処理することによってメモリ参照の回数を減らすことができる。

③ループ融合機能

同一のパラメータを持つ隣り合う DO ループを融合し、一つのループにまとめることができる。ループ融合には、アンローリングと同様の効果がある。

(3) 自動ベクトル化に関する強化項目

① IF 文と GOTO 文によるループや DO WHILE ループのベクトル化

DO ループばかりでなく、IF 文と GOTO 文によるループや DO WHILE ループも、可能であれば DO ループに変換した後にベクトル化することができる。

② 条件ベクトル化

データの依存関係が不明なループや、ループ長が不明なループに対して、ベクトルコードとスカラコードの両方を生成し、実行時に最適な方を選択して実行させることができる。

(4) その他の強化項目

① 拡張浮動小数点データ形式のサポート

従来より広い数値範囲を持つ浮動小数点形式として、拡張浮動小数点データ形式が使用できる。

② 数値記憶単位を８バイトとするモードのサポート

数値記憶単位の大きさを４バイトではなく８バイトとするモードを利用できる。このモードの場合、整数や論理データの大きさも８バイトとなるため、３２ビットを超える大きな整数値を取り扱うこともできる。

③ プログラム実行や入出力に関する特性情報の出力機能

プログラムの終了時に、ベクトル化率や MFLOPS 情報などのプログラムの実行に関する特性情報や入出力回数や転送データ量などの入出力に関する特性情報を出力させることができる（proginf 機能ならびに fileinf 機能）。

④ IEEE 浮動小数点データファイルに対する入出力機能

IEEE の浮動小数点データ形式を採用しているワークステーションで作成された書式なしファイルに対して、入出力を直接実行することができる。この場合、データの形式変換が自動的に行われる。

SX-3/14R の FORTRAN77/SX では、これら機能の一部を、新たに組み込んだ fopp と呼ばれる最適化プリプロセッサと機能を分担することによって実現しています。

本稿では、始めに SX-2N と SX-3R を対比しながら、SX-3R でのハードウェアの強化点を活かすためのチューニング方法、ならびに FORTRAN77/SX の強化機能を活用してチューニングする方法について説明し、その後で SX-3R 上での開発支援ツールを使ったチューニング方法について説

明します。

３．ＳＸ－３Ｒのハードウェア上の強化点

SX-3R で強化されたハードウェア機能のうち、性能に影響する主な項目には次のようなものがあります。

① 演算パイプラインの多重化

SX-2N は、加算，乗算，論理演算およびシフトの４本のパイプライン × ４セット、計16本のベクトルパイプラインを持っていました。SX-3R では、パイプラインの総数は同じですが、それぞれが多機能パイプラインとなっており、加算またはシフトが実行可能なパイプラインが２本と乗算または論理演算が実行可能なパイプラインが２本の計４本のパイプライン × ４セットという構成になっています。これにより、同時に２本（８個）の加算と２本（８個）の乗算が実行可能となり、演算能力が２倍に強化されています。

② ベクトルレジスタの強化

SX-3R では、ベクトルレジスタの内で、中間結果を保持するために使用されるベクトルデータレジスタ（VDR）の容量が 64 Kバイトから 128 Kバイトに拡大されています。また、ベクトル演算に使用されるベクトルレジスタ回りのパスも強化されています。

③ 新命令の追加

SX-3R では、最大値／最小値やそのインデックスを求めるベクトル命令などが、新たに追加されています。

④ リストベクトル命令の高速化

SX-3R では、SX-2N に比べて、リストベクトルを処理するベクトル命令が大幅に高速化されています。

これらのハードウェア機能を FORTRAN77/SX で十分に引き出すためには、次のようなことが考えられます。なお、FORTRAN77/SX の機能については４章で改めて説明します。

(1) ループ中の演算量を増やすとともにメモリアクセスを減少させる

前述のように、SX-3R ではベクトルパイプライン回りが強化されています。しかし、強化されたパイプラインをフルに活用するためには、ループが少なくとも二つの加算と乗算を含んでいる必要があります。また、それぞれの演算で必要とするデータのうち、メモリから取り出す必要があるものの割合が高いと、メモリからベクトルレジスタへの転送がネックになってベクトルパイプラインが遊んでしまいますので、なるべく多くのデータをベクトルレジスタ上に保持することによって、メモリ参照頻度を少なくする必要があります。

言い換えれば、SX-3R の性能を十分に引き出すためには、なるべく演算量が多く、かつメモリアクセスが軽いループを書く必要があります。メモリアクセスの軽いループとは、次のような条件を満たすものです。

・代入文の右辺が多くの演算を含んでいる。
・同一の配列要素を何度も引用している。

・一度計算した結果を何度も引用している。
・組込み関数を多用している。
・スカラのオペランドが適度に混在している。

ループが上記の条件を満たしていなくても、次のような FORTRAN77/SX の機能を使うことにより、条件を満たすようにチューニングすることが可能な場合があります。

・演算量を増やす　：　アンローリング
・メモリアクセスを減少させる　：　外側ループのアンローリング
　ストリップマイニングの外側への追い出し

(2)最大値／最小値

前述のように、SX-3R では、最大値および最小値を求める命令が追加されました。FORTRAN77/SX では、最大値あるいは最小値を求めるループを認識した場合、この新命令を使って高速に処理されます。なお、ループ長が２５６以下の場合には、ベクトル化指示オプションとして shortloop を指定することにより、さらに高速に実行することができます。

(3)リストベクトル

前述のように、SX-3R では、SX-2N に比べてリストベクトルを処理する命令が大幅に高速化されました。そのため、リストベクトルを多用したプログラムの場合には、大幅な性能向上が期待できます。ただし、通常のロード／ストア命令に比べれば実行性能は落ちるため、ループ長がある程度長ければ、可能な限りリストベクトルを使わない形に修正したほうがよいでしょう。

４．FORTRAN77/SX の機能を活用したチューニング手法

プログラムのチューニングという観点からは、FORTRAN77/SX の次の機能が重要です。

・手続きのインライン展開
・行列積の高速化
・アンローリング
・ストリップマイニングの外側への追い出し
・条件ベクトル化
・その他ベクトル化機能(多重ループ、データ宣言の変更、リストベクトル回避)
・Ｉ／Ｏの最適化（ＤＯ型並びの最適化）
・性能向上支援機能(proginf 機能、支援ツール)

ここでは、これらの機能について説明するとともに、その機能を使用したチューニング例を示します。なお、性能向上支援機能については５章で説明します。

(1) 手続きのインライン展開機能

手続きのインライン展開機能は、サブルーチンや関数副プログラムを、呼び出し側のプログラム内に直接展開するものです。主な効果は、次のとおりです。

① 副プログラムの呼び出しを含むループが、ベクトル化の対象となる。

② 副プログラムを呼び出すための時間が節約できる。

なお、命令列が一か所にまとまるため、ハードウェアの命令キャッシュや命令バッファのヒット率が向上する場合もあります。

図1に、①による性能向上例とその効果を示します。

```
DO I=1,N                              DO I=1,N
  CALL SUB(A(I),B(I),C(I))   →          A(I)=SQRT(B(I)**2+C(I)**2)   →  ベクトル化
END DO                                END DO
    :
SUBROUTINE SUB(X,Y,Z)
X=SQRT(Y**2+Z**2)
RETURN
END
```

約 4 MFLOPS　　→　　約 2.1 GFLOPS (520倍)

図1　インライン展開によりベクトル化可能となるループの例

(2) 行列積の高速化

行列積は、種々の基本的な演算の中でも最も重要な演算の一つです。そこで、FORTRAN77/SX は、行列積を計算している多重ループを認識すると、予め用意した行列積専用の高速ライブラリの呼び出しに自動的に置き換える機能をもっています。そのライブラリの中では、ベクトルパイプラインとベクトルレジスタをフルに活用していますので、ハードウェアのピーク性能に近い性能を容易に得ることができます。図2に、この機能が適用される行列積演算の例と、行列のサイズが256×256の時の性能値を示します。なお、例の中で、JとKの指定の順番についてはどちらを先に指定してもかまいません。

```
DO I=1,N
  DO J=1;N
    A(I,J)=0.0
    DO K=1,N                                    DO I=1,N
      A(I,J)=A(I,J)+B(I,K)*C(K,J)   →             CALL MXV(.....)
    END DO                                      END DO
  END DO
END DO
```

SX-2N…約 700 MFLOPS
SX-3R…約 5.4 GFLOPS(行列積ライブラリ置換あり)
　　約 1.2 GFLOPS(行列積ライブラリ置換なし)

図2　行列積ライブラリ自動呼び出しの例

(3) アンローリング

アンローリングとは、繰り返し数を１／ｎにして、ループの本体の処理量をｎ倍にする処理のことです。ｎをアンローリングの段数といいます。図３は、ｎが２の場合の例です。

```
      DO 10 I=2,511                          DO 10 I=2,511,2
10      A(I,J)=B+C(I,J)/A(I-1,J)      →        A(I,J)=B+C(I,J)/A(I-1,J)
                                          10     A(I+1,J)=B+C(I+1,J)/A(I,J)
```

図３　アンローリングの例

アンローリングは、小さなスカラループに対しては、一般にかなりの効果があります。これは、ループ中の並列に実行可能な演算が増加することにより、スカラのパイプラインの利用効率が向上するためです。図３のスカラループに対して４段アンローリングを施した場合の効果は、次のとおりです。

約 9.5 MFLOPS　　→　　約 12.5 MFLOPS (1.3倍)

一方、ベクトルループの場合には、次のようなデメリットが生じ、かえって性能が低下する場合があります。

・ループの長さが短くなる。
・配列要素の参照が不連続アクセスになり、バンク競合が生じる。

しかし、外側のループのインデックスによるアンローリングでは、上記のデメリットが生じないばかりか、大きな効果が得られる場合があります。

```
DO J=1,1024                              DO J=1,1024,4
  DO I=1,1024                              DO I=1,1024
    A(I,J)=B(I,J)+C(I,K)*D(K,J)   →          A(I,J)=B(I,J)+C(I,K)*D(K,J)
  END DO                                     A(I,J+1)=B(I,J+1)+C(I,K)*D(K,J+1)
END DO                                       A(I,J+2)=B(I,J+2)+C(I,K)*D(K,J+2)
                                             A(I,J+3)=B(I,J+3)+C(I,K)*D(K,J+3)
                                         END DO
                                       END DO
```

約 1.7 GFLOPS　　約 2.0 GFLOPS (1.2倍)

図４　外側ループでのアンローリングの例

この例では、外側ループのインデックス“Ｊ”でアンローリングされているため、最内側ループの長さは変わらず、また配列参照も連続アクセスのままです。一方、演算量は加算と乗算の個数がそれぞれ４個になり、さらに４ヶ所に現れる配列要素 C(I,K) への参照が１回ですむためメモリ参照回数が減ることと相まって、性能が向上します。

(4) ストリップマイニング処理の追い出し

当センターの SX-3/14R では、ベクトル命令で一度に計算できる要素の数（＝ベクトルレジス

タの最大長）は、256 です。したがって、ループの長さがそれより大きい場合、例えば 1,000 の場合には、コンパイラは 232, 256, 256, 256 とループを４分割して計算するコードを生成します。これをストリップマイニング処理と呼びます。

さて、次の二重ループの場合には、ストリップマイニング処理を外側のループの外に追い出すことが可能です。図５は fopp がそのために行う変形を示したものです。この変形により、最内側のループに対するストリップマイニング処理は不要となります。なお、ベクトル化指示行の SHORTLOOPパラメータは、 コンパイラに対して、ストリップマイニング処理を行う必要がないことを指示するものです。

```
DO J=1,N                                  DO I1=0,M-1,MAXVL()  ! MAXVL():最大ベクトル
  DO I=1,M                                  I2=MINO(M-I1,MAXVL())  !      長の取り出し
    A(I)=A(I)+B(I,J)*S    →                 DO J=1,N
  END DO                                *VDIR SHORTLOOP
END DO                                      DO I=1,I2
                                              A(I1+I)=A(I1+I)+B(I1+I,J)*S
                                            END DO
                                          END DO
                                        END DO

   約 1.7Gflops           →          約 3.1Gflops (1.8倍)
```

図５　ストリップマイニングの外側への追い出しの例

効果は次のとおりです。変形後のループにおいて、ベクトル A(I1+I) は外側のループのインデックス“Ｊ”に依存しておらず、またストリップマイニング処理が不要なため、ループ“Ｊ”内では、A(I1+I) に対してベクトルレジスタが固定的に割り当てられ、レジスタへのロードとストアは、ループ“Ｊ”の開始前と終了後に行われます。これにより、ループ“Ｊ”内でのメモリへの参照を大幅に減らすことが可能となり、性能が向上します。

この処理は、コンパイラと fopp が役割を分担しながら協力して行っています。

(5) 条件ベクトル化

FORTRAN77/SX では、配列要素の定義と引用の間の依存関係が不明なためにベクトル化の可否が判断できないループや、ループの長さが不明なためにベクトル化の損得が判断できないループに対して、ベクトルコードとスカラコードの両方を生成するとともに、いずれを実行すべきかを判定する条件文を直前に生成して、実行時には最適な方を実行させることができます。

図６に、条件ベクトル化の例を示します。(a)が依存関係に、(b)がループ長に関するものです。この機能は、プログラムの修正が不要であるという点で便利なものですが、条件判断のための IF 文の実行時間が余分にかかるとか、コードの大きさが大きくなるなどのデメリットもあります。もし、ユーザが依存関係の有無や実際のループ長について明確に判断できる場合には、ベクトル化指示行の nodep あるいは novector を使用して、明示的にベクトル化したり、あるいはベクトル化を抑止すべきです。例えば、ベクトル化指示オプションの nodep 指定によりベクトル化した時の性能と条件ベクトル化した場合の性能を比較すると、ループ長が短い場合は数%性能が低下します。しかし、ループ長がある程度長い場合（数百程度）は、性能はほとんど変わりません。

```
   DO 10 I=1,N
10   A(I)=A(I+K)+B(I)

          ↓

   IF(K.GE.0.OR.K.LE.-N) THEN
     ベクトルコード
   ELSE
     スカラコード
   END IF
```

(a)依存関係型

```
   DO 10 J=1,N
     DO 20 I=J,N
20     A(I,J)=A(I,J)+T*B(I,J)
10 CONTINUE

          ↓

   DO 10 J=1,N
     IF(N-J+1.GT.4) THEN
       ベクトルコード
     ELSE
       スカラコード
     END IF
10 CONTINUE
```

(b)ループ長型

図６　条件ベクトル化の例

(6) 密な多重ループの入れ換え

密な多重ループの場合、FORTRAN77/SX は、それぞれのループのループ長や、配列の参照パターン（連続なアクセスか飛びをもつアクセスか，また飛びはいくつか）などを解析し、有効と判断される場合には、ループの入れ換えを行っています。しかし、場合によってはコンパイラの判断が逆効果になる場合もあるため注意が必要です。

以下の例の場合、FORTRAN77/SX はループを入れ換えることにより配列のアクセスが連続になり、性能が向上すると判断して、ループを入れ換えます。この入れ換えは、N の値が十分に大きい場合には効果が出ますが、非常に小さいときには、ベクトル長が短くなるため、性能が逆に低下します。

```
SUBROUTINE SUB(A,C,N,M)                    SUBROUTINE SUB(A,C,N,M)
REAL A(N,M),C(N,M)                         REAL A(N,M),C(N,M)
  :                                          :
DO I=1,N                                   DO J=1,M
  DO J=1,M                                   DO I=1,N
    A(I,J)=A(I,J)+B*C(I,J)        →            A(I,J)=A(I,J)+B*C(I,J)
  END DO                                     END DO
END DO                                     END DO

(N=1025,M=1024 の時)
        約 1.3Gflops              →              約 1.8Gflops(1.4倍)
(N=33,M=1024  の時)
        約 1.3Gflops              →              約 220Mflops(0.2倍)
```

図７　ループ入れ換えの例

(7) データ宣言の変更

２次元以上の配列の宣言で、若い次元に２のｎ乗の倍数の値が指定された場合、その配列の二次元目以後をループのインデックスとして参照すると、同一のメモリバンクに対するアクセスが

頻発するため、バンク競合によるロス時間が多く発生し性能が大幅に低下します。このようなプログラムの場合、配列の宣言を少し変更することにより、バンク競合を回避することができます。なお、fopp を利用すれば、この変換を自動的に行うこともできます。ただし、手続き間をまたがった無矛盾性の解析は行わないため、手続き間で次元数が異なっている場合等では結果不正になる可能性がありますので注意してください（次のバージョンでは、手続き間をまたがって無矛盾性の解析を行うように強化する予定です）。

```
REAL D(128,128),E(128,128),F(128,128,3),X(128,128),Y(128,128)
DO J=3,126
  DO K=1,128
    F(J,K,1)=F(J,K,1)-YY*F(J-2,K,1)-XX*F(J-1,K,1)*ZZ
    F(J,K,2)=F(J,K,2)-YY*F(J-2,K,2)-XX*F(J-1,K,2)*ZZ
    F(J,K,3)=F(J,K,3)-YY*F(J-2,K,3)-XX*F(J-1,K,3)*ZZ
    X(J,K)=(D(J,K)-XX*Y(J-1,K))*ZZ
    Y(J,K)=E(J,K)*ZZ
  END DO
END DO
```

↓

```
REAL D(129,128),E(129,128),F(129,128,3),X(129,128),Y(129,128)
```

約 60Mflops　→　約 1.4Gflops (23倍)

図８　データ宣言の変更の例

(8) リストベクトルの回避

前述のように、SX-3R ではリストベクトル命令の高速化が図られていますが、通常のロード／ストア命令に比べるとまだ実行性能は落ちるため、可能ならばリストベクトルを使わない形に修正した法がよいでしょう。以下にループを入れ換えることによって、回避する例を示します。

この例の場合、変形前は、インデックス“Ｉ”によってベクトル化されるため、C(IX(I),J) はリストベクトルとして処理されますが、変形後は、インデックス“Ｊ”でベクトル化されるため、C(IX(I),J) はリストベクトルとはならず、等間隔ベクトルとして処理されます。

```
DO J=1,M                                  DO I=1,N
  DO I=1,N                                  DO J=1,M
    A(I,J)=A(I,J)+B*C(IX(I),J)      →       A(I,J)=A(I,J)+B*C(IX(I),J)
  END DO                                    END DO
END DO                                    END DO
```

約 800Mflops　→　約 1.4Gflops(1.8倍)

図９　リストベクトル回避の例

(9) ＤＯ形並びの最適化

入出力並びとして、DO 形並びが指定されている時に、並び中に式が現れた場合、あるいは転送する配列要素が連続アクセスでない場合等には、ループに展開して一要素づつ処理するため性能

が低下します。このような場合、一旦ワーク配列に格納し、そのワーク配列を入出力並びに指定すると高速に処理されます。この変換は fopp でも自動的に行うことができます。

```
REAL X(12800)                                    REAL X(12800),D1(12800
WRITE(1)(X(IX(I)),I=1,12800)          →          J1=0
                                                 DO I=1,12800
                                                   J1=J1+1
                                                   D1(J)=X(IX(I))
                                                 END DO
                                                 WRITE(1)D1
```

処理時間(CPU時間)：約 150ミリ秒　　→　　約 80ﾏｲｸﾛ秒(1900倍)

図１０　ＤＯ形並びの最適化の例

５．支援ツールを使ってのチューニング手順について

ここでは、SX-3R 上で FORTRAN プログラムをチューニングする手順について、proginf 情報および支援ツールを利用して、ご説明します。

(1) proginf 情報の採取

プログラムのチューニングのために、利用者の皆さんに最初にしていただきたいことは、proginf 情報のチェックです。図１１に proginf 情報の出力例と個々の意味を示します。

```
    ******  Program Information  ******
Real Time (sec)       :        12.259569  (経過時間)
User Time (sec)       :        10.483070  (ユーザCPU時間)
Sys  Time (sec)       :         0.276029  (システムCPU時間)
Vector Time (sec)     :         7.786912  (ベクトル命令実行時間)
Inst. Count           :       130767103.  (全命令実行数)
V. Inst. Count        :        48369514.  (ベクトル命令実行数)
V. Element Count      :      4900625096.  (ベクトル命令処理要素数)
FLOP Count            :      2206591494.  (浮動小数点データ処理要素数)
MOPS                  :       475.340020  (１秒あたりの実行演算数)
MFLOPS                :       210.490963  (１秒あたりの浮動小数点データ処理要素数)
VLEN                  :       101.316401  (命令レベルでの平均ベクトル長)
V. Op. Ratio (%)      :        98.346434  (ベクトル演算率)
MIPS                  :        12.474123  (１秒あたりの実行命令数)
Cache (sec)           :         0.860654  (キャッシュミスによるロス時間)
Bank (sec)            :         8.162429  (バンクコンフリクトによる時間)
```

図１１　ｐｒｏｇｉｎｆ情報の出力例

特に調べていただきたい項目は、ベクトル演算率，MOPS 値または MFLOPS 値，平均ループ長およびバンク競合によるロス時間です。もし、ベクトル演算率が不十分な場合には、コンパイラあるいはANALYZER/SX の情報を参考にしながら、さらにベクトル化を進めてください。ベクトル演

算率が高いにもかかわらず MOPS 値または MFLOPS 値が不十分な場合には、ループ長は十分か，またバンク競合の値が異常に大きくないかを確認してください。図１１の例では、バンク競合によるロス時間が多く発生しており、原因の解析が必要であることがわかります。

なお、proginf 情報での平均ベクトル長は、ベクトル命令が処理した平均要素数であり、256を超えることはありませんので注意してください。

(2) ANALYZER/SX と ANALYZER-P/SXを使ってのチューニング

チューニングを支援するバッチ型のツールとしては、ANALYZER/SX と ANALYZER-P/SX の二つがあります。ANALYZER/SX は、SX-2 で提供していた ANALYZER/SX を SUPER-UX に移植したもので、ベクトル化チューニングのために必要十分な機能をもっています。ただし、プログラムの静的解析機能はサポートしていません。ANALYZER-P/SX は、ANALYZER/SX を主として並列処理チューニングの観点から強化したものです。しかし、強化項目の中にはベクトル化チューニングにも有効なものが含まれています。また、プログラムの静的解析機能も備えています。ここでは、ANALYZER/SX を利用してプログラムをチューニングする手順を、出力例を参照しながら説明します。

(a)実行時間解析情報の採取

ANALYZER/SXで最初にしていただきたいのは、ANALYZER/SX の実行時間解析機能を利用して、プログラム単位毎に proginf と同様の情報を採取することです（図１２参照）。この情報をもとに、実行時間の大きいプログラム単位から順に、proginf 情報の時と同様のチェックを行ってください。ここで問題のプログラム単位が見つかった場合には、実行時間解析情報あるいは ANALYZER-P/SX の DO ループ単位解析情報を利用してさらに問題の DO ループを特定していきます。

(b)実行回数解析情報の採取

そのプログラム単位中の個々のループについて、ベクトル化状況やループ長などについて調べてください（図１３参照）。この解析では各ステートメントの実行回数(EXECUTION)と推定実行時間比率(COST)が表示されますので、これらの大きなループやステートメントに注目するとよいでしょう。また、バンク競合が多く発生している場合（実行時間解析リスト中で、BANK CONF. の欄に星印が二つ以上出力されている場合）には、配列の宣言やループ中での配列の参照パターンをチェックし、２のｎ乗の倍数飛びのアクセスになっていないかをチェックしてください。

(c)DO ループ単位解析情報の採取

ANALYZER-P/SX で採取可能な DO ループ単位解析情報は、ループ単位での実行時間、ベクトル化状況、ループ長およびバンク競合などの情報が出力されますので、これらから問題となる DO ループを特定して下さい（図１４参照）。特定できれば、その DO ループに対して実行回

数解析と同様の調査を行って下さい。

この例では、ベクトルデータの飛びが128要素になっているためバンク競合によるロス時間が発生して遅くなっていることが判ります。そして図８に示したデータ宣言の変更により高速化が可能になります。

SUMMARY LIST

```
*------------------------------------------------------------------------*
!  ENVIRONMENT                                                           !
!        ANALYZER/SX REVISION   : REV.019                                !
!        ANALYZED DATE          : Wed Oct  6 10:58:22 1993               !
!                                                                        !
!  PROGRAM TOTAL INFORMATION                                             !
!        EXECUTION ( CPU ) TIME =   0 :  0 ' 11 " 627    (      11627 MSEC)  !
!        INSTRUCTION COUNT      =           129573337                    !
!        VECTOR OPERATION RATIO =  98.37%                                !
!        AVERAGE VECTOR LENGTH  =      101.3                             !
!        MOPS                   =      428.479                           !
!        MFLOPS                 =      189.785                           !
!        BANK CONFLICT          : SEVERE                                 !
!                                                                        !
!  OVERHEAD INFORMATION                                                  !
!        SUBROUTINE CALL FACTOR =  14.74 MICRO SEC                       !
*------------------------------------------------------------------------*
```

ATR.	PROGRAM	FREQUENCY	BANK CONF.	TIME(%)	AV.TIME	V.OP. RATIO	AV.V.LEN.	INSTR.	V.INSTR.	V.ELEMENT	MOPS	MFLOPS
-->	TEST	1		3(0.03)	3	0.74	20.1	80860	30	602	27.39	0.27
FNC	FUN1	14		0(0.00)	0	0.00	0.0	1009	0	0	9.11	0.51
SUB	SUB1	1420		58(0.50)	0	98.39	255.9	2205183	425120	108769200	1914.06	0.00
SUB	SUB2	1		11(0.10)	11	0.00	0.0	493502	0	0	44.23	18.36
FNC	FUN2	100		132(1.14)	1	98.98	256.0	10447304	2880000	737280000	5628.26	3169.33
SUB	SUB3	400	***	5595(49.28)	9	99.32	128.0	7991618	4258400	545226400	147.17	66.04
SUB	SUB4	1		27(0.24)	27	96.26	256.0	840189	76800	19660800	745.16	251.06

図１２　ANALYZER/SX による実行時間解析情報

```
FORMAT LIST        PROGRAM:SUB3

*-------------------------------------------------------------------------------------------------------
------------------------------*
!                          test.f
                                !
!
                                !
!                          EXECUTION CPU TIME    =  0: 0' 5"856 (       5856 MSEC) (  51.66% )
                                !
!                          EXECUTION FREQUENCY   =     91647200                    (  49.97% )
                                !
!                          EXECUTION COST RATIO  =      51.88%
                                !
*-------------------------------------------------------------------------------------------------------
------------------------------*

  LINE    EXECUTION COST(%) TRUE%   LOOP          FORTRAN STATEMENT

 001026         400                                SUBROUTINE SUB3 (PAR1, PAR2, PAR3)
 001027                                            "
 001028                                            PARAMETER (NJA=128, NJB=128, JL=1, JU=128, KL
=1, KU=128)
 001029                                            COMMON /ARRAYS/ A(NJA,NJB), B(NJA,NJB), C(NJA,
NJB), D(NJA,NJB),
 001030                                           : E(NJA,NJB), F(NJA,NJB,3), X(NJA,NJB), Y(NJA,
NJB), FX(NJA,NJB,3)
 001031                                            "
 001032          400(  0.01)                       DO 10 J = JL+2,JU-2
 001033        49600(  0.08)         1---------->  !  DO 20 K = KL,KU
 001034      6348800( 10.12)         ! V-------->  !  !  F(J,K,1) = SQRT( (F(J,K,1) - PAR2*F(J-2,
K,1) - PAR1*F(J-1,K,1))*PAR3 )
 001035      6348800( 10.12)         ! !           !  !  F(J,K,2) = SQRT( (F(J,K,2) - PAR2*F(J-2,
K,2) - PAR1*F(J-1,K,2))*PAR3 )
 001036      6348800( 10.12)         ! !           !  !  F(J,K,3) = SQRT( (F(J,K,3) - PAR2*F(J-2,
K,3) - PAR1*F(J-1,K,3))*PAR3 )
 001037      6348800(  0.74)         ! !           !  !  X(J,K) = (D(J,K) - PAR1*Y(J-1,K))*PAR3
 001038      6348800(  0.70)         ! !           !  !  Y(J,K) = E(J,K)*PAR3
 001039      6348800(  0.25)         ! V---------- 20  !  CONTINUE
 001040        49600(  0.01)         1------------ 10  CONTINUE
```

図１３　ANALYZER/SX による実行回数解析情報

```
*------------------------------------------------------------------------------*
   PROGRAM UNIT DETAIL LIST
      PROGRAM UNIT NAME = SUB3
      FILE NAME         = test.f
*------------------------------------------------------------------------------*

 LINE          NESTING          FREQUENCY  INCLUSIVE          EXCLUSIVE          P/V   AVERAGE  BA
NK
                                          CPU TIME(   %  )   CPU TIME(   %  )         LOOP.LEN CO
NF.

               SUB3                  400     5.870( 51.7 )      0.002(  0.0 )
 001032--001040  -DO 10              400     4.530( 40.0 )      0.012(  0.1 )              124
 001033--001039  --DO 20           49600     4.518( 39.8 )      4.518( 39.8 )  V           128  *
**
 001045--001053  -DO 30              400     0.017(  0.1 )      0.017(  0.1 )  V           128
 001057--001064  -DO 40              400     0.014(  0.1 )      0.014(  0.1 )  V           128
 001066--001076  -DO 50              400     0.008(  0.0 )      0.008(  0.0 )  V           128
```

図１４　ANALYZER-P/SX による DO ループ単位解析情報

６．日本語入力の為の個人環境設定

下記記述を＄ｈｏｍｅ／．ｃｓｈｒｃに追加して下さい。その後、ログインするかもしくは．ｃｓｈｒｃを再起動すれば、日本語入力の為の個人環境が設定されます。

```
set term = kterm
setenv LANG japan
stty -istrip
stty cs8
stty flush undef
alias rs 'eval  resize -c '
rs
/usr/necbin/can
```

CTRL ＋ o を押下すれば、日本語モードに切り替わり、日本語入力が可能となります。

７．おわりに

以上、SX-3/14R の FORTRAN77/SX を使っていただく上でのチューニング手法ならびにチューニング手順を主にご紹介してきました。皆さんが SX-3/14R の性能を引き出すための作業を行われる上で、本稿が少しでもお役にたてば幸いです。

[執筆者紹介]

*1 NEC基本ソフトウェア開発本部第一言語開発部

*2 NEC基本ソフトウェア開発本部第一言語開発部

*3 NECソフトウェア関西第一応用システム事業部官庁システム部

*4 NECソフトウェア関西第一応用システム事業部官庁システム部