SR8000チューニング支援機能を利用したプログラムチューニング

（株）日立製作所

1　はじめに

これまでに、本「スーパーコンピューティングニュース」の以下の記事において、SR8000の「診断ログメッセージ出力機能」、及び、「性能モニター機能」についてご紹介させて頂きました。

「SR8000の有効利用法」　Vol.1, No.3 1999
「SR8000性能モニター機能の利用法」　Vol.1, No.4 1999
「SR8000性能モニターとログメッセージを用いたチューニング」　Vol.2, No.1 2000
「SR8000性能モニター機能の仕様改善のご報告」　Vol.2, No.2 2000

SR8000でプログラムの実行時間を短縮し、実行性能を向上させるためには、「診断ログメッセージ出力機能」、「性能モニター機能」を活用してプログラムチューニングを行うことが有効です。

ここでは、仕様が改善された性能モニターと診断ログメッセージ（以下、ログメッセージと呼びます）を利用して、サンプルプログラムのチューニング過程を紹介しながら、SR8000の有効的な利用方法を説明します。

2　SR8000で効率良く動作するロードモジュールの作成方法

＜プログラム全体の実行状況調査>

(1) ログメッセージと性能モニター情報（関数/サブルーチン単位に情報取得）の出力を指定してコンパイル・リンクし、ロードモジュールを作成する。
(2) ソースファイル毎に(ソースファイル名).logファイルが出力されていることを確認する。
(3) ロードモジュールを実行する。
(4) 性能モニター情報出力コマンドにより関数/サブルーチン単位の情報を表示し、負荷の高い関数/サブルーチンを特定する。

＜関数/ サブルーチン単位の実行状況調査＞

(5) ログメッセージで、負荷の高い関数/サブルーチン内のDOループに要素並列化/擬似ベクトル化が適用されているか確認する。

＜要素並列化単位の実行状況調査＞

(6) 負荷の高い関数/サブルーチンに対して、性能モニター情報（要素並列化単位に情報取得）の出力を指定して再コンパイル・リンクし、ロードモジュールを再作成する。
(7) ロードモジュールを再実行する。
(8) 性能モニター情報出力コマンドにより要素並列化単位の情報を表示し、要素並列化単位毎の負荷分散状況などを調査する。
(9) ログメッセージで、負荷の高い要素並列化箇所の要素並列化/擬似ベクトル化状況を確認する。

＜高速化への分析/ 対策＞

(10) 要素並列化/擬似ベクトル化が適用されていなければ、必要に応じてサブオプションの付加、ディレクティブの挿入、プログラムソースの変更を行う。

＜チューニング効果確認＞

(11) ロードモジュールを再作成し、再実行する。

(12) ログメッセージ、性能モニター情報及び実行性能を確認する。

(13) その他、性能阻害個所について調査する。

＜プログラム全体のチューニング効果確認>

(14) 必要なチューニングが全て適用されていることを確認した後、プログラム全体のチューニング効果を確認する。

上記、(5)～(13)を繰り返し、負荷の高い箇所の要素並列化/擬似ベクトル化適用率を高めることで、SR8000で効率良く動作するロードモジュールの作成が可能になります（図2-1）。

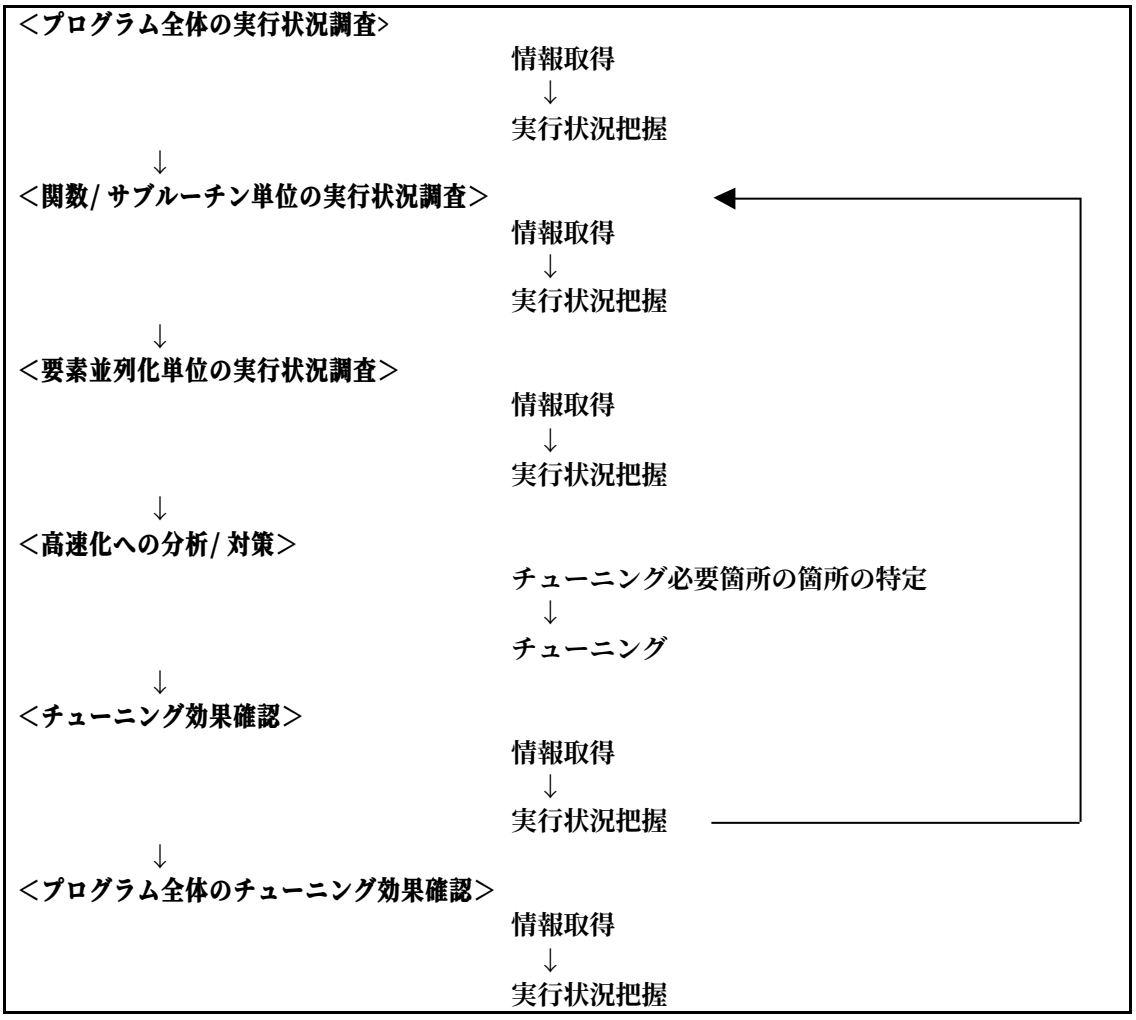

図 2-1　チューニングの流れ

3　チューニングの例

以下、サンプルプログラムを使って、チューニングの手順をご紹介します。

図 3-1に示すプログラムsample.fは、5つのサブルーチンSUB1、SUB2、SUB3、SUB4、SUB5コールから構成されています。それぞれのサブルーチンは、ソースファイルsub1.f、sub2.f、sub3.f、sub4.f、sub5.f（図 3-2～図 3-6）に記述されています。

```
      PROGRAM MAIN
      PARAMETER(NN=10000000)
      PARAMETER(NN2=3)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN),L(NN)
      DIMENSION D(NN2)
C
      WRITE(6,*) 'Test start'
      DO 5 I=1,60
         CALL SUB1(NN,A,B,L,I)
         CALL SUB2(NN,A,B,C,L)
         CALL SUB3(NN,A,B,C,L)
         CALL SUB4(NN,A,B,C,D,NN2)
         CALL SUB5(D,NN2,S)
         WRITE(6,*) I,'Sum=',S
    5 CONTINUE
      WRITE(6,*) 'Test end'
      STOP
      END
```

図 3-1　sample.f

```
      SUBROUTINE SUB1(NN,A,B,L,NC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),L(NN)
      DO 10 I=1,NN
         L(I)=I
         A(I)=DBLE(NC)*DBLE(I)*1.0D-4
         B(I)=EXP(A(I)*1.0D-5)
   10 CONTINUE
      RETURN
      END
```

図 3-2　sub1.f

```
      SUBROUTINE SUB2(NN,A,B,C,L)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN),L(NN)
      DO 20 J=1,NN/500-1
         C(J)=0.0D0
         DO 30 I=1,L(J)
            C(J)=C(J)+A(I)+B(I)
   30    CONTINUE
   20 CONTINUE
c
      DO 40 J=NN/500,NN
         C(J)=A(J)+B(J)
   40 CONTINUE
      RETURN
      END
```

図 3-3　sub2.f

```
      SUBROUTINE SUB3(NN,A,B,C,L)
      DIMENSION A(NN),B(NN),C(NN),L(NN)
      DO 50 I=1,NN
         B(L(I))=C(I)+A(I)
   50 CONTINUE
      RETURN
      END
```

図 3-4　sub3.f

```
      SUBROUTINE SUB4(NN,A,B,C,D,NN2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN)
      DIMENSION D(NN2)
      DO 60 J=1,NN2
         D(J)=0.0D0
         DO 70 I=1,NN
            D(J)=D(J)+J*(C(I)+A(I)+B(I))
   70    CONTINUE
   60 CONTINUE
      RETURN
       END
```

図 3-5　sub4.f

```
      SUBROUTINE SUB5(D,NN2,S)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION D(NN2)
      S=0.0D0
      DO 80 I=1,NN2
            S=S+D(I)
   80    CONTINUE
      RETURN
      END
```

図 3-6　sub5.f

4　プログラム全体の把握

<プログラム全体の実行状況調査>

プログラム全体の実行時間の内訳を調べます。

処理の重いサブルーチンを特定するため、関数/サブルーチン単位の実行時間を採取する性能モニターのオプション-pmfuncを指定し、更に、DOループが要素並列化されているか、擬似ベクトル化されているかを判断するために、ログメッセージを出力するオプション-loglistを指定してコンパイルします（図4-1）。

```
Prompt> f77 -Oss -procnum=8 -loglist -c sample.f -pmfunc
Prompt> f77 -Oss -procnum=8 -loglist -c sub1.f -pmfunc
Prompt> f77 -Oss -procnum=8 -loglist -c sub2.f -pmfunc
Prompt> f77 -Oss -procnum=8 -loglist -c sub3.f -pmfunc
Prompt> f77 -Oss -procnum=8 -loglist -c sub4.f -pmfunc
Prompt> f77 -Oss -procnum=8 -loglist -c sub5.f -pmfunc
```

図 4-1　コンパイル

コンパイル終了後、ログメッセージファイル（sample.log、sub1.log、…、sub5.log）がソースファイルと同じディレクトリ（コンパイルを実行したディレクトリ）に作成されます。

オプション-0ssにより要素並列化を指定しているため、オプション-parallelを、性能モニターの使用を指定しているため、オプション-pmfuncを、それぞれリンク時に指定します（図4-2）。

```
Prompt> f77 sample.o sub1.o sub2.o sub3.o sub4.o sub5.o –pmfunc -parallel -o sample
```

図 4-2　リンク

作成したロードモジュールを実行します。

実行終了後、実行性能情報ファイル（バイナリデーター）が出力されます。

例：pm_sample_Jun08_1600_node000_174889

（pm_ロードモジュール名_日付_時間_nodeノード番号_プロセス番号）

性能モニター情報出力コマンド(pmpr)により、性能モニター情報をテキスト形式で表示します。

```
Prompt> pmpr pm_sample_Jun08_1600_node000_174889
pm_sample_Jun08_1600_node000_174889:
############################################################
## 実行総計                                                ##
############################################################
  日付                  : 2000年06月08日(木) 16時00分33秒
  ノード番号            : 0
  プロセスＩＤ          : 174889
  実行モジュール名      : sample
  CPU時間               : 39.210296[秒]
  MFLOPS                : 1316.379
  MIPS                  : 2496.630
  入力回数              : 2
  出力回数              : 62
  入力量                : 108[バイト]
  出力量                : 2660[バイト]
  ---------------------------------
             FLOP      Inst     LD/ST
  ---------------------------------
  IP0        5852M    14345M>   6545M>
  IP1        6002M    11728M    4510M
  IP2        6752M    12994M    4932M
  IP3        5102M<    9235M<   3329M<
  IP4        5852M    10500M    3751M
  IP5        6602M    11766M    4172M
  IP6        7352M    13032M    4594M
  IP7        8100M>   14293M    5015M
  ---------------------------------
  合計      51616M    97894M   36847M


############################################################
## 関数／手続き                                            ##
############################################################
========================================
== 関数／手続き順位                    ==
========================================
        CPU時間[%]          回数  関数(ファイル+行番号)
-----------------------------------------------------
[  1]    13.813[35.23]        60  SUB2(sub2.f+5)
[  2]    11.160[28.46]        60  SUB1(sub1.f+5)
[  3]     7.420[18.92]        60  SUB4(sub4.f+6)
[  4]     6.807[17.36]        60  SUB3(sub3.f+4)
```

```
[  5]  0.010895[ 0.03]        1  MAIN(sample.f+8)
[  6]  0.000313[ 0.00]       60  SUB5(sub5.f+4)
----------------------
合計      39.210[100.00]
```

図 4-3　性能モニター情報出力

図 4-3より、実行時間がプログラム全体に占める割合の高いサブルーチンSUB1、SUB2、SUB3、SUB4をチューニング対象とします。MAINとサブルーチンSUB5は、全体の実行時間に対する割合が1%以下と低いため、全体の性能にほとんど影響を与えていません。

以降、チューニング対象であるサブルーチンSUB1、SUB2、SUB3、SUB4について実行比率が高い順に、

(1)ログメッセージファイルの確認

(2)性能モニターによる性能情報ファイルの確認

を行い、チューニング箇所の特定、チューニング効果の確認を行っていきます。

5　サブルーチン毎のチューニング

5.1　サブルーチンSUB2のチューニング

サブルーチンSUB2は、全体の実行時間に対する割合が35.23[%]と最も高いので、まずこのサブルーチンのチューニングを行います。

＜関数/サブルーチン単位の実行状況調査＞

始めに、ログメッセージファイルsub2.logの内容を調べ、DOループが要素並列化されているか、擬似ベクトル化されているかを確認します（図5-1）。

```
      SUBROUTINE SUB2(NN,A,B,C,L)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN),L(NN)


** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB2
** 並列ループ
**    I : TLOCAL変数
      DO 20 J=1,NN/500-1
         C(J)=0.0D0

**
**    [DO 30]
**    最内側アキュムレータ変数展開(8倍)を行った。
**    擬似ベクトル化を適用した。
**
         DO 30 I=1,L(J)
            C(J)=C(J)+A(I)+B(I)
   30    CONTINUE
   20 CONTINUE
c

** 並列処理を継続
** 並列ループ
** --- ループ入口でバリア出力 ---
** ループ出口で並列処理 終了
**
```

```
**     [DO 40]
**     最内側ループ展開(8倍)を行った。
**     擬似ベクトル化を適用した。
**
       DO 40 J=NN/500,NN
          C(J)=A(J)+B(J)
   40  CONTINUE
       RETURN
       END
```

図 5-1　ログメッセージsub2.log

サブルーチンSUB2の2つのループ（DO20とDO40）は、共に要素並列化されていることが分かります。また、擬似ベクトル化も適用されています。ログメッセージファイルには、性能上の問題点を示すメッセージはありません。

＜要素並列化単位の実行状況調査＞

次にサブルーチンSUB2には要素並列化箇所があるため、要素並列化単位の性能情報を採取する性能モニターのオプション-pmparを追加指定し、ロードモジュールを再作成、再実行します。

```
～  途中省略  ～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                   ==
========================================
        CPU時間[%]        MFLOPS      MIPS     回数  関数[順位](ファイル+行番号)
------------------------------------------------------------------------------
[  1]    13.814[35.36]  1781.965  3064.768       60  SUB2[1](sub2.f+5)
----------------------
合計      13.814[35.36]

========================================
== 要素並列部分詳細                   ==
========================================
[  1] 関数[順位](ファイル+行番号) : SUB2[1](sub2.f+5)
          CPU時間     FLOP      Inst     LD/ST    MFLOPS      MIPS      回数
   ---------------------------------------------------------------------
   IP0      13.814>    452M<     863M<     338M<   32.739<   62.458<       60>
   IP1      13.812    1202M     2128M      760M    87.044   154.101        60>
   IP2      13.799    1952M     3394M     1182M   141.477   245.963        60>
   IP3      13.791    2702M     4660M     1604M   195.946   337.884        60>
   IP4      13.777    3452M     5925M     2026M   250.578   430.082        60>
   IP5      13.748<   4202M     7191M     2447M   305.671   523.067        60>
   IP6      13.770    4952M     8457M     2869M   359.636   614.122        60>
   IP7      13.763    5700M>    9718M>    3290M>  414.158>  706.131>       60>
   ---------------------------------------------------------------------
   合計    110.273   24616M    42336M    14515M  1781.965  3064.768       480
   ---------------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間  :  99.79[%] =  110.273/(13.813742*8)
        FLOP     :  53.98[%] = 24615599160/(5699848740*8)
```

図 5-2　性能モニター情報出力(SUB2)

図 5-2よりFLOPの負荷分散比率をみると

FLOP　　: 53.98[%] = 24616M/(5700M*8)

であり、いずれも約50%とサブルーチンSUB2の要素並列化箇所のIP間の負荷バランスが悪いことが分かります。

＜高速化への分析/対策＞

そこで、サブルーチンSUB2の2重ループ（DO20とDO30）について調査します。DO30のループ長 L(J)は外側のDO20のインデックスJに依存しています。また、配列Lはサブルーチン SUB1（図 3-2）で L(I)=Iと定義されています。

要素並列化は、外側のループDO20に対して行われており、DO20のループ長（=NN/500-1）は19999（MAINルーチンでNN=10000000と定義されている）なので、各IPの担当するDO20のインデックスとDO30のループ長は表 5-1のようになります。

表 5-1　各IPの担当するDO20のインデックスとDO30のループ長の関係

計算をするIP番号	DO20のインデックスJ	DO30のループ長L(J)
0	1	L(1)=1
0	2	L(2)=2
0	3	L(3)=3
0	:	:
0	2500	L(2500)=2500
1	2501	L(2501)=2501
1	2502	L(2502)=2502
:	:	:
7	19998	L(19998)=19998
7	19999(=NN/500-1)	L(19999)=19999

表 5-1より、IP0は主にDO30のループ長の短い部分を計算し、IP7はループ長の長い部分を担当しているので、IP間の負荷バランスが悪いことが分かります。

この場合、表 5-2に示すように要素並列化すれば、IP間の負荷バランスをほぼ均等にすることができます。

表 5-2　IP間負荷バランスのよい分割

計算をするIP番号	DO20のインデックスJ	DO30のループ長L(J)
0	1	L(1)=1
1	2	L(2)=2
2	3	L(3)=3
:	:	:
3	2500	L(2500)=2500
4	2501	L(2501)=2501
5	2502	L(2502)=2502
:	:	:
5	19998	L(19998)=19998
6	19999(=NN/500-1)	L(19999)=19999

このようにループを要素並列化する分割方法をサイクリック分割といい、該当ループにディレクテ

ィブ*poption cyclic を挿入することにより実現できます（変更後のソースは、図5-3のログメッセージファイルを参照して下さい）。

＜チューニング効果確認＞

ディレクティブ挿入後sub2.fを再コンパイルし、ログメッセージファイルを再作成します。

```
      SUBROUTINE SUB2(NN,A,B,C,L)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN),L(NN)

*poption cyclic

** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB2
** 並列ループ（サイクリック分割）
**    I : TLOCAL変数
      DO 20 J=1,NN/500-1
         C(J)=0.0D0

**
**    [DO 30]
**    最内側アキュムレータ変数展開(8倍)を行った。
**    擬似ベクトル化を適用した。
**
         DO 30 I=1,L(J)
            C(J)=C(J)+A(I)+B(I)
   30    CONTINUE
   20 CONTINUE
c

** 並列処理を継続
** 並列ループ
** --- ループ入口でバリア出力 ---
** ループ出口で並列処理 終了
**
**    [DO 40]
**    最内側ループ展開(8倍)を行った。
**    擬似ベクトル化を適用した。
**
      DO 40 J=NN/500,NN
         C(J)=A(J)+B(J)
   40 CONTINUE
      RETURN
      END
```

図 5-3　ログメッセージ（ディレクティブ挿入後）sub2.log

DO20は、サイクリック分割されていることが分かります。

一般的に要素並列化は、DOループを順に等分割しますが、IP間の負荷バランスが悪い場合には、ディレクティブ*poption cyclicが有効な場合があります。

再度、要素並列化単位の性能情報を採取します。

```
～  途中省略  ～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                     ==
========================================
        CPU時間[%]        MFLOPS       MIPS      回数  関数[順位](ファイル+行番号)
------------------------------------------------------------------------------
[  1]     7.366[22.57]  3341.698  5747.325        60  SUB2[3](sub2.f+6)
----------------------
合計      7.366[22.57]

========================================
== 要素並列部分詳細                     ==
========================================
[  1] 関数[順位](ファイル+行番号) : SUB2[3](sub2.f+6)
          CPU時間      FLOP      Inst     LD/ST     MFLOPS       MIPS      回数
   ---------------------------------------------------------------------------
   IP0       7.366>   3076M     5290M     1814M   417.610<  718.187<       60>
   IP1       7.361    3077M     5291M     1814M   417.937   718.781        60>
   IP2       7.357    3077M     5292M     1814M   418.227   719.310        60>
   IP3       7.364    3077M     5293M     1815M   417.873   718.734        60>
   IP4       7.363    3077M     5293M     1815M   417.981   718.950        60>
   IP5       7.337<   3078M     5294M     1815M   419.488>  721.574>       60>
   IP6       7.365    3078M>    5295M>    1816M>  417.907   718.887        60>
   IP7       7.359    3076M<    5288M<    1813M<  417.993   718.609        60>
   ---------------------------------------------------------------------------
   合計     58.871   24616M    42336M    14515M  3341.698  5747.325       480
   ---------------------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間  :  99.90[%] =   58.871/(7.366195*8)
        FLOP     :  99.97[%] = 24615599160/(3078000060*8)
```

図 5-4　性能モニター情報出力(SUB2)（ディレクティブ挿入後）

図 5-4よりFLOPの負荷分散比率をみると

```
        FLOP     : 99.97[%] = 24616M/(3079M*8)
```

であり、いずれも約100%とサブルーチンSUB2の要素並列化箇所のIP間の負荷バランスが均等になったことが分かります。

最後に、性能向上効果を確認します。

```
～  途中省略  ～

############################################################
## 関数／手続き                                           ##
############################################################
========================================
== 関数／手続き順位                     ==
========================================
        CPU時間[%]          回数  関数(ファイル+行番号)
--------------------------------------------------
[  1]    11.018[33.78]        60  SUB1(sub1.f+5)
[  2]     7.419[22.75]        60  SUB4(sub4.f+6)
```

```
[  3]      7.356[22.55]        60  SUB2(sub2.f+6)
[  4]      6.807[20.87]        60  SUB3(sub3.f+4)
[  5]  0.014418[ 0.04]         1  MAIN(sample.f+8)
[  6]  0.000315[ 0.00]        60  SUB5(sub5.f+4)
----------------------
合計      32.614[100.00]
```

図 5-5　性能モニター情報出力（ディレクティブ挿入後）

表 5-3　サブルーチンSUB2のチューニング効果

	チューニング前の実行時間 [sec]	チューニング後の実行時間 [sec]
サブルーチンSUB2	13.813	7.356

表 5-3よりディレクティブ*poption cyclicの挿入で、負荷バランスの良い要素並列化が可能になり、実行時間が約50%に短縮したことが分かります。

5.2　サブルーチンSUB1のチューニング

＜関数/サブルーチン単位の実行状況調査＞

始めに、ログメッセージファイルsub1.logの内容を調べ、DOループが要素並列化されているか、擬似ベクトル化されているかを確認します。

```
      SUBROUTINE SUB1(NN,A,B,L,NC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),L(NN)


** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB1
** 並列ループ
** ループ出口で並列処理 終了
**
**    [DO 10]
XX    対象となる配列参照が無いので擬似ベクトル化を適用しなかった。
**
      DO 10 I=1,NN
         L(I)=I
         A(I)=DBLE(NC)*DBLE(I)*1.0D-4
         B(I)=EXP(A(I)*1.0D-5)
   10 CONTINUE
      RETURN
      END
```

図 5-6　ログメッセージsub1.log

サブルーチンSUB1のループ（DO10）は、要素並列化されていることが分かります。また、このループでは配列参照をしていない（配列Aはループ中で定義されている）ので擬似ベクトル化は適用されませんがこれは問題ありません。

＜高速化への分析/対策＞

ここでは、倍精度の数学関数EXPに着目します。べき乗、EXP、LOG、SIN、COSなどの数学関数は擬似ベクトル化数学関数の対象ですが、ログメッセージファイルを見るとDO10に対しては適用されてい

ないことが分かります（擬似ベクトル化数学関数が適用された場合には、「ベクトルライブラリ化を行った。」というメッセージが出ます）。これは、EXPの引数が式（A(I)*1.0D-5）になっているためです。このような場合は、ディレクティブ**voption pvfunc(2)**を指定します（コンパイルオプション-pvfunc=2 を指定してsub1.fをコンパイルしても同様な動作をします）。

＜チューニング効果確認＞

ディレクティブ挿入後sub1.fを再コンパイルし、ログメッセージファイルを再作成します（図5-7）。

```
      SUBROUTINE SUB1(NN,A,B,L,NC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),L(NN)

*voption pvfunc(2)

** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB1
** 並列ループ
** ループ出口で並列処理 終了
**
**    [DO 10]
**    ベクトルライブラリ化を行った(_hf_pvdexp)。
**    最内側ループ展開(2倍)を行った。
XX    対象となる配列参照が無いので擬似ベクトル化を適用しなかった。
**
      DO 10 I=1,NN
         L(I)=I
         A(I)=DBLE(NC)*DBLE(I)*1.0D-4
         B(I)=EXP(A(I)*1.0D-5)
   10 CONTINUE
      RETURN
      END
```

図 5-7　ログメッセージ（ディレクティブ挿入後）sub1.log

擬似ベクトル化数学関数（_hf_pvdexp）が適用されたことが分かります。

この変更による効果を確認します。ディレクティブを挿入したsub1.fで、ロードモジュールを再作成、再実行します。

```
～　途中省略　～

############################################################
## 関数／手続き                                           ##
############################################################
========================================
== 関数／手続き順位                    ==
========================================
        CPU時間[%]          回数  関数(ファイル+行番号)
----------------------------------------------------------
[  1]      7.460[29.57]        60  SUB2(sub2.f+6)
[  2]      7.442[29.49]        60  SUB4(sub4.f+6)
[  3]      6.806[26.98]        60  SUB3(sub3.f+4)
[  4]      3.515[13.93]        60  SUB1(sub1.f+6)
[  5]  0.007728[ 0.03]         1  MAIN(sample.f+8)
```

```
[   6]   0.000351[ 0.00]         60   SUB5(sub5.f+4)
----------------------
合計       25.232[100.00]
```

図 5-8　性能モニター情報出力(SUB1)（ディレクティブ挿入後）

表 5-4　サブルーチンSUB1のチューニング効果

	チューニング前の実行時間 [sec]	チューニング後の実行時間 [sec]
サブルーチンSUB1	11.160	3.515

表 5-4よりディレクティブ*voption pvfunc(2)の挿入で、擬似ベクトル関数が適用され、実行時間が約30%に短縮したことが分かります。

＜要素並列化単位の実行状況調査＞

次にサブルーチンSUB1には要素並列化箇所があるため、要素並列化単位の性能情報を採取する性能モニターのオプション-pmparを追加指定し、ロードモジュールを再作成、再実行します。

```
～　途中省略　～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                    ==
========================================
        CPU時間[%]         MFLOPS       MIPS      回数   関数[順位](ファイル+行番号)
------------------------------------------------------------------------------
[   1]     3.489[13.90]   5159.540   6248.566        60   SUB1[4](sub1.f+6)
----------------------
合計       3.489[13.90]

========================================
== 要素並列部分詳細                    ==
========================================
[   1] 関数[順位](ファイル+行番号) : SUB1[4](sub1.f+6)
          CPU時間      FLOP      Inst     LD/ST     MFLOPS       MIPS      回数
   ---------------------------------------------------------------------------
   IP0       3.486    2250M>    2725M>     669M>   645.530    781.782        60>
   IP1       3.489>   2250M>    2725M>     669M>   644.942<   781.071<       60>
   IP2       3.475    2250M>    2725M>     669M>   647.534    784.210        60>
   IP3       3.486    2250M>    2725M>     669M>   645.538    781.791        60>
   IP4       3.483    2250M>    2725M>     669M>   646.110    782.484        60>
   IP5       3.487    2250M>    2725M>     669M>   645.287    781.487        60>
   IP6       3.474<   2250M>    2725M>     669M>   647.768>   784.493>       60>
   IP7       3.487    2250M>    2725M>     669M>   645.380    781.601        60>
   ---------------------------------------------------------------------------
   合計     27.866    18002M    21801M     5352M   5159.540   6248.566       480
   ---------------------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間  :  99.83[%] =   27.866/(3.488974*8)
        FLOP     : 100.00[%] = 18001502400/(2250187800*8)
```

図 5-9　性能モニター情報出力(SUB1)（ディレクティブ挿入後）

図 5-9よりFLOPの負荷分散比率をみると

```
        FLOP      : 100.00[%] = 18002M/(2250M*8)
```

であり、いずれも100%とサブルーチンSUB1の要素並列化箇所のIP間の負荷バランスが均等になっていることが分かります。

＜高速化への分析/対策＞

IP間の負荷バランスが良く、特に必要な対策はありません。

5.3　サブルーチンSUB4のチューニング

＜関数/サブルーチン単位の実行状況調査＞

始めに、ログメッセージファイルsub4.logの内容を調べ、DOループが要素並列化されているか、擬似ベクトル化されているかを確認します（図5-10）。

```
      SUBROUTINE SUB4(NN,A,B,C,D,NN2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN)
      DIMENSION D(NN2)


** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB4
** 並列ループ
**    I : TLOCAL変数
**
**    [DO 60]
**    外側ループ展開(8倍)を行った。
**
      DO 60 J=1,NN2
         D(J)=0.0D0

** 並列処理を継続
** ループ出口で並列処理 終了
**
**    [DO 70]
**    最内側アキュムレータ変数展開(2倍)を行った。
**    擬似ベクトル化を適用した。
**
         DO 70 I=1,NN
            D(J)=D(J)+J*(C(I)+A(I)+B(I))
   70    CONTINUE
   60 CONTINUE
      RETURN
      END
```

図 5-10　ログメッセージsub4.log

サブルーチンSUB4のループ（DO70）は、要素並列化されていることが分かります。また、擬似ベクトル化も適用されています。ログメッセージファイルには、性能上の問題点を示すメッセージはありません。

＜要素並列化単位の実行状況調査＞

次にサブルーチンSUB4には要素並列化箇所があるため、要素並列化単位の性能情報を採取する性能

モニターのオプション-pmparを追加指定し、ロードモジュールを再作成/再実行します。

```
～  途中省略  ～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                    ==
========================================
        CPU時間[%]        MFLOPS       MIPS      回数  関数[順位](ファイル+行番号)
------------------------------------------------------------------------------
[  1]     7.419[28.15]   970.486  2031.965        60  SUB4[1](sub4.f+6)
----------------------
合計       7.419[28.15]

========================================
== 要素並列部分詳細                    ==
========================================
[  1] 関数[順位](ファイル+行番号) : SUB4[1](sub4.f+6)
          CPU時間      FLOP      Inst     LD/ST     MFLOPS       MIPS      回数
   ---------------------------------------------------------------------------
   IP0       7.418    2400M>    5025M>    2025M>  323.522>  677.375>       60>
   IP1       7.419>   2400M>    5025M>    2025M>  323.495   677.320        60>
   IP2       7.419    2400M>    5025M>    2025M>  323.512   677.354        60>
   IP3       0.000       0M<     0.0M<     0.0M<    0.000<   56.233        60>
   IP4       0.000       0M<     0.0M<     0.0M<    0.000<   52.747<       60>
   IP5       0.000       0M<     0.0M<     0.0M<    0.000<   57.024        60>
   IP6       0.000       0M<     0.0M<     0.0M<    0.000<   53.146        60>
   IP7       0.000<      0M<     0.0M<     0.0M<    0.000<   57.665        60>
   ---------------------------------------------------------------------------
   合計     22.257    7200M    15075M     6075M   970.486  2031.965       480
   ---------------------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間  :  37.50[%] =   22.257/(7.418960*8)
        FLOP     :  37.50[%] = 7200000360/(2400000120*8)
```

図 5-11　性能モニター情報出力(SUB4)

図 5-11よりFLOPの負荷分散比率をみると

FLOP　　: 37.5[%] = 7200M/(2400M*8)

であり、いずれも約37.5%とサブルーチンSUB4の要素並列化箇所のIP間の負荷バランスが悪いことが分かります。特に浮動小数点演算は、IP0、IP1、IP2のみが実行しています。

＜高速化への分析/対策＞

ログメッセージファイルsub4.log（図 5-10）を見ると、要素並列化は外側のループDO60に対して行われていますが、DO60のループ長NN2はプログラムのMAIN部分でNN2=3と定義されています。このため、DO60は3つのIPで実行されてしまいます。

一般的には外側のループで要素並列化を適用した方が効率は良いのですが、このように、極端に外側のループ長が短い場合は、内側ループを要素並列化してIP間の負荷バランスを均等化する方が効率が良くなります。

そこで、外側のループDO60で要素並列化が適用されないようにディレクティブ**poption**

noparallelを挿入します。

＜チューニング効果確認＞

ディレクティブ挿入後sub4.fを再コンパイルし、ログメッセージファイルを再作成します。

```
      SUBROUTINE SUB4(NN,A,B,C,D,NN2)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(NN),B(NN),C(NN)
      DIMENSION D(NN2)

*poption noparallel

** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB4
XX 逐次ループ
**
**    最内側ループ解消(8倍)を行った。
**
      DO 60 J=1,NN2
         D(J)=0.0D0

** 並列処理を継続
** 並列ループ
**    D : リダクション配列(SUM)
** --- ループ入口でバリア出力 ---
** --- ループ出口でバリア出力 ---
** ループ出口で並列処理 終了
**
**    [DO 70]
**    最内側アキュムレータ変数展開(8倍)を行った。
**    擬似ベクトル化を適用した。
**
         DO 70 I=1,NN
            D(J)=D(J)+J*(C(I)+A(I)+B(I))
   70    CONTINUE
   60 CONTINUE
      RETURN
      END
```

図 5-12　ログメッセージ（ディレクティブ挿入後）sub4.log

外側のDO60では要素並列化は抑止され、内側のDO70で要素並列化されていることが分かります。

再度、要素並列化単位の性能情報を採取します。

```
～　途中省略　～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                   ==
========================================
        CPU時間[%]        MFLOPS       MIPS      回数  関数[順位](ファイル+行番号)
------------------------------------------------------------------------------
[  1]     1.642[ 8.48]  4384.130  5411.781        60  SUB4[4](sub4.f+7)
----------------------
```

```
合計        1.642[ 8.48]

========================================
== 要素並列部分詳細                    ==
========================================
[  1] 関数[順位](ファイル+行番号) : SUB4[4](sub4.f+7)
          CPU時間      FLOP      Inst     LD/ST     MFLOPS       MIPS      回数
   ---------------------------------------------------------------------------
   IP0       1.641      900M>    1111M>     380M>  548.306    676.832        60>
   IP1       1.642      900M<    1111M<     380M<  548.250    676.761        60>
   IP2       1.641<     900M<    1111M<     380M<  548.446>   677.003>       60>
   IP3       1.642      900M<    1111M<     380M<  548.117    676.596        60>
   IP4       1.641      900M<    1111M<     380M<  548.408    676.955        60>
   IP5       1.642      900M<    1111M<     380M<  548.245    676.755        60>
   IP6       1.642>     900M<    1111M<     380M<  548.016<   676.472<       60>
   IP7       1.642      900M<    1111M<     380M<  548.263    676.777        60>
   ---------------------------------------------------------------------------
   合計     13.133     7200M     8888M     3038M  4384.130   5411.781       480
   ---------------------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間  :  99.96[%] =   13.133/(1.642292*8)
        FLOP     : 100.00[%] = 7200023040/(900004140*8)
```

図 5-13　性能モニター情報出力(SUB4)（ディレクティブ挿入後）

図 5-13よりFLOPの負荷分散比率をみると

FLOP　　: 100.00[%] = 7200M/(900M*8)

であり、いずれも約100%とサブルーチンSUB2の要素並列化箇所のIP間の負荷バランスが均等になったことが分かります。

最後に、性能向上効果を確認します。

```
～  途中省略  ～

############################################################
## 関数／手続き                                           ##
############################################################
========================================
== 関数／手続き順位                    ==
========================================
        CPU時間[%]          回数  関数(ファイル+行番号)
-----------------------------------------------------
[  1]      7.382[38.13]        60  SUB2(sub2.f+6)
[  2]      6.807[35.16]        60  SUB3(sub3.f+4)
[  3]      3.517[18.17]        60  SUB1(sub1.f+6)
[  4]      1.642[ 8.48]        60  SUB4(sub4.f+7)
[  5]  0.010107[ 0.05]         1  MAIN(sample.f+8)
[  6]  0.000321[ 0.00]        60  SUB5(sub5.f+4)
----------------------
合計      19.359[100.00]
```

図 5-14　性能モニター情報出力(SUB4)（ディレクティブ挿入後）

表 5-5　サブルーチンSUB4のチューニング効果

	チューニング前の実行時間 [sec]	チューニング後の実行時間 [sec]
サブルーチンSUB4	7.420	1.642

表 5-5よりディレクティブ*poption noparallelの挿入で、負荷バランスの良い要素並列化が可能になり、実行時間が約25%に短縮したことが分かります。

5.4　サブルーチンSUB3のチューニング

＜関数/サブルーチン単位の実行状況調査＞

始めに、ログメッセージファイルsub3.logの内容を調べ、DOループが要素並列化されているか、擬似ベクトル化されているかを確認します。

```
      SUBROUTINE SUB3(NN,A,B,C,L)
      DIMENSION A(NN),B(NN),C(NN),L(NN)


XX 逐次ループ
**    B : ループ内に不明依存がある
**
**    [DO 50]
**    最内側ループ展開(4倍)を行った。
**    擬似ベクトル化を適用した。
**
      DO 50 I=1,NN
         B(L(I))=C(I)+A(I)
   50 CONTINUE
      RETURN
      END
```

図 5-15　ログメッセージsub3.log

サブルーチンSUB3のループ（DO50）が、要素並列化されていないことが分かります。

＜高速化への分析/対策＞

原因は、メッセージ「B: ループ内に不明依存がある」に示されているように、配列Bのループ内の依存性の有無をコンパイラが判断できないためです。

配列Bのインデックスに使用される配列LはサブルーチンSUB1（図 3-2）でL(I)=Iと定義されているので、実は配列Bのループ内の依存性はありません。このような場合、ディレクティブ**poption indep(b)**を挿入することによってコンパイラーに、依存性の無いこと（並列化が可能であること）を明示します。

一方、擬似ベクトル化に関しては適用されているので問題ありません。

＜チューニング効果確認＞

ディレクティブ挿入後sub3.fを再コンパイルし、ログメッセージファイルを再作成します。同時に要素並列化単位の性能情報を採取する性能モニターのオプション-pmparを追加指定し、ロードモジュールを再作成、再実行します。

```
      SUBROUTINE SUB3(NN,A,B,C,L)
      DIMENSION A(NN),B(NN),C(NN),L(NN)

*poption indep(b)

** ループ入口で並列処理 開始
** 並列手続き名 : _parallel_func_1_SUB3
** 並列ループ
** ループ出口で並列処理 終了
**
**    [DO 50]
**    最内側ループ展開(4倍)を行った。
**    擬似ベクトル化を適用した。
**
      DO 50 I=1,NN
         B(L(I))=C(I)+A(I)
   50 CONTINUE
      RETURN
      END
```

図 5-16　ログメッセージ（ディレクティブ挿入後）sub3.log

DO50が要素並列化されていることが分かります。

＜要素並列化単位の実行状況調査＞

次に、要素並列化単位の性能情報を採取します。

```
～　途中省略　～

############################################################
## 要素並列部分                                           ##
############################################################
========================================
== 要素並列部分順位                    ==
========================================
        CPU時間[%]        MFLOPS      MIPS     回数  関数[順位](ファイル+行番号)
--------------------------------------------------------------------------------
[  1]  0.855274[ 5.84]   701.530  4538.117       60  SUB3[4](sub3.f+5)
----------------------
合計    0.855274[ 5.84]

========================================
== 要素並列部分詳細                    ==
========================================
[  1] 関数[順位](ファイル+行番号) : SUB3[4](sub3.f+5)
          CPU時間     FLOP      Inst     LD/ST    MFLOPS      MIPS      回数
   ------------------------------------------------------------------------
   IP0       0.855   75000k>     485M>     307M>   87.747   567.625        60>
   IP1       0.855>  75000k>     485M>     307M>   87.691<  567.265<       60>
   IP2       0.855   75000k>     485M>     307M>   87.754   567.673        60>
   IP3       0.855   75000k>     485M>     307M>   87.705   567.353        60>
   IP4       0.855<  75000k>     485M>     307M>   87.761>  567.718>       60>
   IP5       0.855   75000k>     485M>     307M>   87.701   567.327        60>
   IP6       0.855   75000k>     485M>     307M>   87.759   567.704        60>
   IP7       0.855   75000k>     485M>     307M>   87.710   567.388        60>
   ------------------------------------------------------------------------
   合計      6.839  600000k     3881M     2456M   701.530  4538.117       480
```

```
   ----------------------------------------------------------------
   ＩＰ負荷分散比率 : (合計)/(最大値 * ＩＰ数)
        CPU時間   :  99.96[%] =     6.839/(0.855274*8)
        FLOP      : 100.00[%] = 600000000/(75000000*8)
```

図 5-17　性能モニター情報出力(SUB3)（ディレクティブ挿入後）

図 5-17よりFLOPの負荷分散比率をみると

```
        FLOP      : 100.00[%] =  600M/( 75M*8)
```

であり、いずれも約100%とサブルーチンSUB4の要素並列化箇所のIP間の負荷バランスが均等になっていることが分かります。

＜チューニング効果確認＞

最後に、性能向上効果を確認します。

```
～  途中省略  ～

############################################################
## 関数／手続き                                           ##
############################################################
========================================
== 関数／手続き順位                    ==
========================================
        CPU時間[%]          回数  関数(ファイル+行番号)
------------------------------------------------------------
[  1]      7.362[50.28]        60  SUB2(sub2.f+6)
[  2]      4.768[32.56]        60  SUB1(sub1.f+6)
[  3]      1.641[11.21]        60  SUB4(sub4.f+7)
[  4]  0.857531[ 5.86]         60  SUB3(sub3.f+5)
[  5]  0.013243[ 0.09]          1  MAIN(sample.f+8)
[  6]  0.000337[ 0.00]         60  SUB5(sub5.f+4)
----------------------
合計      14.642[100.00]
```

図 5-18　性能モニター情報出力(SUB3)（ディレクティブ挿入後）

表 5-6　サブルーチンSUB3のチューニング効果

	チューニング前の実行時間 [sec]	チューニング後の実行時間 [sec]
サブルーチンSUB3	6.807	0.858

表 5-6よりディレクティブ*poption indep(b)の挿入で、要素並列化が可能になり、実行時間が約15%に短縮したことが分かります。

6　チューニング効果のまとめ

プログラム全体及び各サブルーチン毎のチューニング結果をまとめます。

＜プログラム全体のチューニング効果確認＞

表6-1に示すようにチューニングによりプログラム全体の実行時間を、約40%に短縮することができました。

表 6-1　チューニング効果のまとめ

	チューニング前の実行時間［sec］	チューニング後の実行時間［sec］
メインルーチンMAIN	0.010895	0.013243
サブルーチンSUB1	11.160	4.768
サブルーチンSUB2	13.813	7.362
サブルーチンSUB3	6.807	0.857531
サブルーチンSUB4	7.420	1.641
サブルーチンSUB5	0.000313	0.000337
合計	39.210	14.642

7　おわりに

SR8000では、プログラムチューニングを支援する機能として、「診断ログメッセージ出力機能」、「性能モニター機能」を用意しています。今回は、この2つの機能を利用したサンプルプログラムのチューニング例をご紹介させて頂きました。

これらのツールは今後も、更に使い易く、より見やすい形で情報をご提供できるように仕様改善を行っていく予定です。今後も新しい機能の追加や仕様改善など有り次第、追ってご紹介致します。

以上