［大規模科学計算システム］

FORTRAN90/SX と C++/SX の新機能

横谷 雄司
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

SX のコンパイラ FORTRAN90/SX および C++/SX では、SX-9 のハードウェア性能をより引き出すため、最適化機能の強化、メモリアクセス高速化のためのベクトルデータ・バッファリング機能の追加を行っています。また、プログラム実行解析情報出力機能および簡易性能解析機能でより詳細な性能情報を表示する強化も行いました。本稿ではこれらの新機能についてご紹介します。

1. はじめに

スーパーコンピュータ SX-9 は、SX-7、SX-7C から大幅に強化された強力なハードウェア機構により、科学技術計算分野における大規模超高速演算のニーズに応える高い演算性能を実現しています。この SX-9 の強化されたハードウェアを使いこなし、高い演算性能を引き出すためにはコンパイラの能力が重要になります。従来の SX-7、SX-7C でもご利用いただいている Fortran コンパイラ FORTRAN90/SX および C、C++コンパイラ C++/SX も、SX-9 向けに最適化、自動ベクトル化、自動並列化機能が強化され、SX-9 の高い演算性能を容易に引き出すことができます。また、SX-9 で新しく追加されたハードウェア機構である ADB(Assignable Data Buffer)を利用するための機能も追加されており、これを用いることによりさらに高い演算性能を引き出すことも可能です。

以下では、新しいコンパイラで強化された主要な機能について説明します。

2. 最適化強化

最新の FORTRAN90/SX および C++/SX では、SX-9 のハードウェア向けの最適化強化を行うとともに、ループ最適化/インライン展開/自動並列化機能の解析/変換処理を大幅に見直し、従来と比較して、より多くのループの最適化/並列化、手続き呼び出しのインライン展開を可能とするように改善しました。とくに FORTRAN90/SX では、モジュール、構造型、ポインタ、割付け配列などの Fortran95 機能を使用したプログラムに対する解析能力が改善されています。また、SX-9 の高いベクトル演算性能を引き出すために 2 つの新しいループ最適化機能も追加しました。

以下ではこれらのループ最適化機能と SX-9 向けの命令並べ換え最適化について説明します。

2.1 IF 構文のメモリ参照最適化

SX-9 は 1CPU 内に 8 本のベクトル演算パイプラインを持ち、最大 102.4GFlops という非常に高い演算性能を実現しています。この SX-9 の演算能力を最大限引き出すためには、ベクトル演算パイプラインを遊ばせないようにループ内の演算比率を高める最適化が重要になります。

FORTRAN90/SX と C++/SX は、演算比率を高めるための新しい最適化として、ベクトル化可能なループ内の IF 構文の全ての THEN、ELSE ブロック内に同一配列要素への代入が現れる場合、その配列に対するベクトルストア回数を削減する機能をサポートしました。

この最適化は FORTRAN90/SX ではコンパイラオプション **-C hopt** 指定時、または新しく追加された詳細オプション **-Wf"-pvctl cond_mem_opt"** 指定時に有効となります。C++/SX ではコンパイラオプション **-Chopt**、または **-Caopt** 指定時に有効となります。

以下の Fortran プログラムの例で、本最適化について説明します。

例 1

```
DO I=1, N
   IF(X(I).NE.0.0) THEN
```

```
      Y(I)=X(I)*2.0
    ELSE
      Y(I)=0.0
    END IF
END DO
```

このループでは、IF 構文の THEN ブロックと ELSE ブロックの両方で配列 Y(I)への代入が行われています。このループをそのままベクトル化したときのベクトル命令列のイメージは以下のようになります。

ベクトル命令列のイメージ:

```
mask(1:N) ← X(1:N).NE.0.0                          ベクトルマスク生成
vr1(1:N)  ← X(1:N)*2.0 WHEN(mask(1:N))             ベクトルレジスタへの移送
Y(1:N)    ← vr1(1:N)   WHEN(mask(1:N))             ベクトルストア
vr2(1:N)  ← 0.0        WHEN(.NOT.mask(1:N))        ベクトルレジスタへの移送
Y(1:N)    ← vr2(1:N)   WHEN(.NOT.mask(1:N))        ベクトルストア
```

mask :　　ベクトルマスクレジスタ
vr1, *vr2* :　ベクトルレジスタ

生成されたベクトル命令列には、THEN、ELSE ブロック中の Y(I)への代入に対応した 2 個のベクトルストアが含まれています。

これに対し、本最適化を適用すると元のループは以下のように変形されます。

最適化による変形イメージ:

```
DO I=1, N
  IF(X(I).NE.0.0) THEN
    wky=X(I)*2.0          ! Y(I) を 作業変数 wky に置換
  ELSE
    wky=0.0               ! Y(I) を 作業変数 wky に置換
  END IF
   Y(I)=wky               ! wky の値を Y(I) へ代入
END DO
```

この変形により、以下のようなベクトル命令列が生成されます。

最適化されたベクトル命令列のイメージ:

```
mask(1:N) ← X(1:N).NE.0.0                          ベクトルマスク生成
vr(1:N)   ← X(1:N)*2.0 WHEN(mask(1:N))             ベクトルレジスタへの移送
vr(1:N)   ← 0.0        WHEN(.NOT.mask(1:N))        ベクトルレジスタへの移送
Y(1:N)    ← vr(1:N)    WHEN(.NOT.mask(1:N))        ベクトルストア
```

mask : ベクトルマスクレジスタ
vr　: ベクトルレジスタ

最適化前と比べるとベクトルストアが 1 個に削減されています。これにより、実行性能が改善されます。

2.2 ショートループ条件ベクトル化

総和や内積、最大値/最小値といったマクロ演算を含むループに対し、実行時にループ長をチェックし、最大ベクトルレジスタ長以下の場合に専用の高速なベクトルコードを実行するショートループ条件ベクトル化機能をサポートしました。

総和や内積、最大値/最小値などは科学技術計算でよく使用される演算です。これらの演算はループ

の1つ前の繰り返しで求めた値を用いて次の値を計算するため、加算や乗算などのような通常のベクトル命令では計算できません。しかし、SX では総和や最大値・最小値を求める専用の特殊なベクトル命令(マクロ演算命令)を持っているため、これらの演算もベクトル化して高速に実行できます。ただし、これらのマクロ演算命令は通常のベクトル演算命令と比べると低速なので、まず通常のベクトル演算命令を用いて、最大ベクトルレジスタ長(1 つのベクトル命令で処理できる最大要素数。SX では256)以下の個数の部分解を求め、最後にその部分解に対してマクロ演算命令を1回だけ用いて最終的な結果を求めるベクトルコードを生成します。たとえば例 2 に示す総和の場合、まずベクトル加算で最大ベクトルレジスタ長(MaxVL)個の部分和を求め、最後にその部分和についてベクトル総和命令を使い最終的な総和結果を求めます。

例 2　総和

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

ベクトルコードのイメージ:

```
vr(1:MaxVL)=0.0
DO I=1,N, MaxVL                    ! MaxVL 個毎に分割
  L = MIN(MaxVL,N-I)
  vr(1:L) = vr(1:L) + X(I:I+L)     ! ベクトル加算で vr の各要素に部分和を求める
END DO
S = S + VSUM(vr(1:MaxVL))          ! ベクトル総和命令で部分和の総和を求める
```

この例において、ループ長 N が MaxVL 以下 (ショートループ) の場合は、MaxVL 個毎に分割して部分和を求める処理を行わなくても、直接ベクトル総和命令を用いて結果が得られます。

ループ長が MaxVL 以下であることを、コンパイル時にコンパイラが認識できる場合は、部分和を求めるループを省略して総和を直接ベクトル総和命令により求めるショートループコードが生成されます。しかし、ループ長がコンパイル時に不明な場合は、ループ長が MaxVL 以上であっても正しい結果が得られるようにするため、部分和を求めるループを省略することはできません。

ショートループ条件ベクトル化は、ループ長が不明なループに対して、総和、内積、累積、最大値/最小値を直接マクロ演算命令で求めるショートループ用ベクトルコードと通常のベクトルコードの両方を用意し、実行時にループ長をチェックし、MaxVL 以下の場合はショートループコードを、ループ長が MaxVL を越える場合は通常のベクトルコードを実行するベクトル化を行う最適化です。

ショートループ条件ベクトル化は、詳細オプション**-pvctl altcode=short** を指定してコンパイルするか、ベクトル化指示オプション **ALTCODE=(SHORT)**をループに指定したときに適用されます。

以下に、Fortran プログラムでベクトル化指示オプション ALTCODE=(SHORT)を用いたショートループ条件ベクトル化の例を示します。

例 3 ショートループ条件ベクトル化

```
!CDIR ALTCODE=(SHORT)
      DO I = 1, N
         S = S + X(I)
      END DO
```

↓　***条件ベクトル化***

```
      IF(N.LE.MaxVL) THEN          ! ループ長 N が最大ベクトルレジスタ長以下か?
        S = S + VSUM(X(1:N))       ! ベクトル総和命令1命令で S の値を求める
      ELSE
```

```
      DO I = 1, N                    !N > 最大ベクトルレジスタ長の場合
        S = S + X(I)                 ! 通常のベクトル総和コードを実行
      END DO
    END IF
```

マクロ演算を含むループにおいて、ほとんどの場合はループ長が MaxVL 以下であるが、MaxVL を越えるケースが 0 ではないとき、このショートループ条件ベクトル化を適用することにより実行性能が向上します。

ループ長が MaxVL を越える比率が高い場合にショートループ条件ベクトル化を適用すると、ループ長判定のためのオーバーヘッドにより、性能改善が見られない、または逆に性能が低下してしまうことがありますので注意してください。また、ループ長が MaxVL を越えないことが保証できる場合は、

ALTCODE=(SHORT)ではなく SHORTLOOP 指示オプションを指定してコンパイラにそのループが常にショートループであることを教えたほうが、より効率のよいベクトルコードが生成されます。

2.3 SX-9 対応の命令並べ換え最適化

SX-9 は、乗算と加算をそれぞれ 2 個ずつ、論理演算 1 個、除算・平方根演算 1 個の合計 6 個のベクトルパイプラインを備えており、2 個の乗算と 2 個のベクトル加算、ベクトル論理演算、ベクトル除算(または平方根)の同時実行が可能です。SX-9 の性能を引き出すにはこれらのベクトルパイプラインを可能な限り並列動作させることが重要となります。

FORTRAN90/SX と C++/SX は、SX-9 のハードウェアの動作をシミュレートし、ハードウェア資源を最大限活用できるように命令列を最適な順番に並べ換えます。その際には、分岐命令をまたいだ広範囲の命令列を対象に命令を並べ換え、より多くの同時実行可能な命令を探し出すことができます。

たとえば、以下のような Fortran の配列代入文

```
X(:)= 2.0*ABS(A(:))+3.0*ABS(B(:))+4.0*C(:)
```

に対して生成されるベクトル命令列のイメージは以下のようになります。

ベクトル命令列のイメージ

```
vand   vt1,0x7fffffff,A
vfmul  vt2,2.0,vt1
vand   vt3,0x7fffffff,B
vfmul  vt4,3.0,vt3
vfadd  vt5,vt2,vt4
vfmul  vt6,4.0,C
vfadd  X,vt5,vt6
```

※ vand　：ベクトル AND (ABS に対応)
　 vfmul：ベクトル乗算
　 vadd　：ベクトル加算

この例では、配列代入文に対し絶対値 ABS を求めるための 2 個のベクトル AND、3 個のベクトル乗算、2 個のベクトル加算を含むベクトル命令列が生成されています。

コンパイラは、SX-9 のベクトルユニットの動作をシミュレートし可能な限り多くのベクトルパイプラインが同時に動作できるようにベクトルレジスタの割り当てと命令の並べ換えを行い、図 1 に示すように 1 つの AND 演算と 2 個の乗算、2 個の加算の合計 5 個のベクトル演算が同時に動作できるような命令列を生成します。

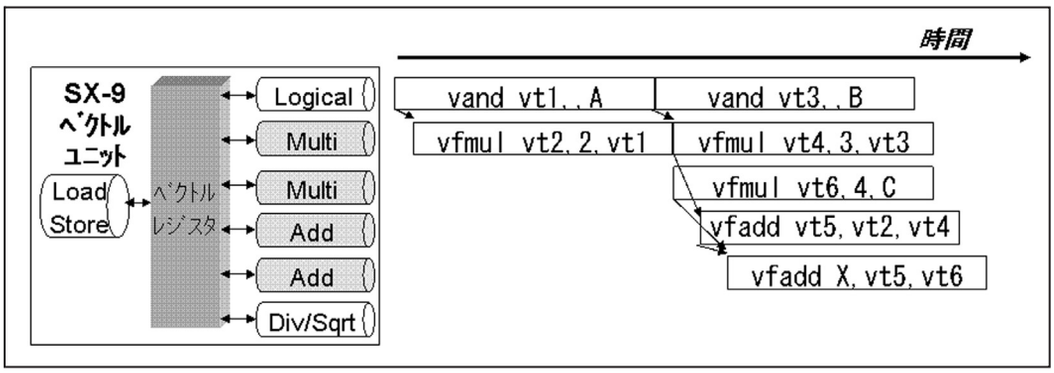

図 1　SX-9 でのベクトル命令列の動作イメージ

3. ベクトルデータ・バッファリング機能

SX-9 では、メモリアクセス高速化のために ADB (Assignable Data Buffer)と呼ばれるハードウェア機構が主記憶とベクトルレジスタの間に追加されました。ADB は主記憶より高速アクセスが可能な記憶領域です。FORTRAN90/SX と C++/SX は、この ADB をプログラムから利用するためのベクトルデータ・バッファリング機能をサポートしました。

3.1 ADB(Assignable Data Buffer)

ADB は通常のキャッシュとは異なり、選択的にデータをバッファリングできます。たとえば、他の配列や変数のアクセスによりデータが ADB から追い出されないように、頻繁にアクセスされる配列のみを ADB にバッファリングしておくことで、その配列を長期間にわたって高速にアクセスできるようになります。

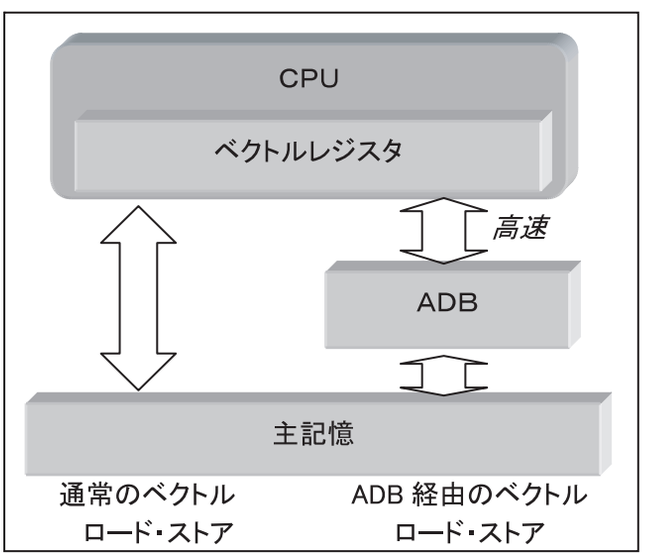

図 2　ADB(Assignable Data Buffer)

ある配列を ADB を使用してアクセスするように指定すると、その配列に対するベクトルロード、ベクトルストアは ADB 経由で行われます。そのとき、それらの最初のベクトルロード、または、ベクトルストアにおいて、データが ADB にバッファリングされます。そして、つぎにそれらのデータを利用するときには、ADB にバッファリングされたデータが使用されます。ADB からは、メモリからロードするより高速にデータをアクセスできるため、2 回目以降のベクトルロードを高速化できます。

3.2　ON_ADB コンパイラ指示オプション

ADB 経由でアクセスする配列の指定には、ON_ADB コンパイラ指示オプションを用います。
FORTRAN90/SX の ON_ADB コンパイラ指示オプションの形式は以下の通りです。

```
!CDIR ON_ADB[(配列変数指定)]
```

C++/SX での形式は以下の通りです。

```
#pragma cdir on_adb[(識別子)]
```

ON_ADB 指示オプションは、ループまたは Fortran の配列式の直前に指定します。配列変数指定や識別子で指定された配列のベクトルロード、ベクトルストアにおいて、定義・参照された配列要素のデータが ADB にバッファリングされます。配列変数指定や識別子が省略されたときは、ループまたは配列式内に現れるすべての配列のベクトルロード、ベクトルストアにおいて、データが ADB にバッファリングされます。

以下に ON_ADB 指示オプションの Fortran プログラムでの使用例を示します。

例 4　ON_ADB の指定例

```
      SUBROUTINE SUB
      COMMON A(4096),B(4096),C(4096)
!CDIR ON_ADB(A)
!CDIR ON_ADB(C)
      DO I=1,4096
            = A(I) + ...          ①
         C(I) = ...               ②
            = A(I) * ...          ③
      END DO
      ...
!CDIR ON_ADB(A)
!CDIR ON_ADB(C)
      DO I=1,4096
         B(I) = A(I) + C(I)       ④
      END DO
      END SUBROUTINE
```

①の配列 A のベクトルロード時に、配列要素 A(1)から A(4096)のデータが ADB にバッファリングされます。
②の配列 C のベクトルストア時に、配列要素 C(1)から C(4096)のデータが ADB にバッファリングされます。
③の配列 A の2番目参照では、①で ADB にバッファリングされた A のデータがベクトルロードされます。
④の 2 番目のループの配列 A、C の参照では、①②で ADB にバッファリングされた配列 A、C のデータがベクトルロードされます。
配列 B は ON_ADB 指示オプションで指定されていないため、ADB を経由せずメモリに直接ベクトルストアされます。したがって、配列 B のデータは ADB にバッファリングされません。

なお、多重ループの外側ループに ON_ADB 指示オプションを指定したときには、そのループに含まれる内側ループには効果は及びませんので注意してください。たとえば、例 5 に示す 2 重ループで配列 B を ADB にバッファリングしたい場合は、内側ループ DO I=1,500 に対して ON_ADB 指示オプションを指定する必要があります。

例 5　多重ループ

```
        DO J=1,100
!CDIR ON_ADB(B)
```

```
    DO I=1,500
      B(I,J) = A(I,J) + C(I,J)
    END DO
  END DO
```

4. プログラム実行解析情報出力機能／簡易性能解析機能強化

FORTRAN90/SXとC++/SXは、従来からプログラムのさまざまな性能情報を簡単に知ることができるプログラム実行解析情報出力機能(proginf)および簡易性能解析機能(ftrace)を提供しています。新しいコンパイラでは、メモリ性能指標のひとつであるバンク競合時間を CPU ポート競合時間とメモリネットワーク競合時間に分けて表示し、より詳しい情報を得られるよう強化されています。

4.1 バンク競合

SX-9 の主記憶は最大 32768 個のバンクで構成され、おのおののバンクに対して並列にロード、ストアを行うことができます。しかし、1つのバンクに対しては同時には1つのロード、ストア要求しか発行できないため、その要求が1つのバンクに重なると前の要求によるロード、ストアが完了するまで後の要求は待たされます。この性能低下をバンク競合と呼びます。バンク競合は競合の発生する場所により、さらに 2 種類に分類されます。SX における CPU と主記憶の接続の概念図は図 3 のようになります。

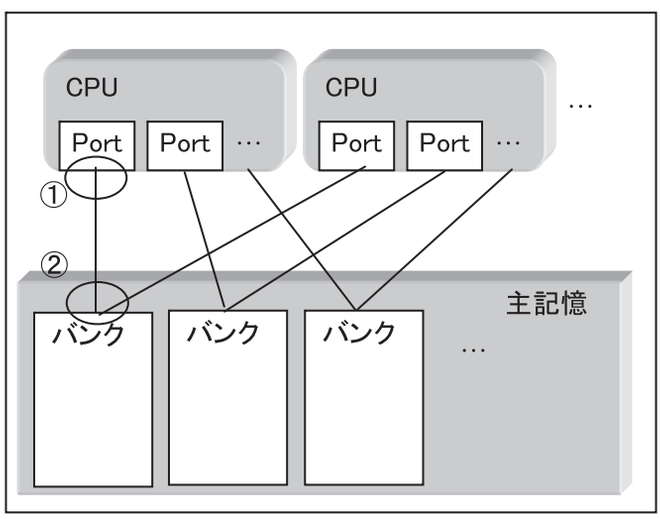

図 3 SX-9 の CPU と主記憶の接続

SX では CPU 内にある複数のポートと呼ばれる部分を通して主記憶と接続されています。ロード、ストアで主記憶の特定のバンクに対して要求が集中したとき、そのバンクに対応する特定のポートのみが使用され競合が発生します(①)。これを CPU ポート競合と呼びます。一方、主記憶側の接続部分②で発生する競合をメモリネットワーク競合と呼びます。

メモリネットワーク競合は、ひとつの CPU からのロード、ストアだけでなく、ノード内のほかの CPU から同一のバンクにロード、ストアの要求が集中したときにも発生します。たとえば、次のようなケースでも発生することがあります。

- 自動並列化、OpenMP 並列化されたプログラムの複数のタスクから同時に同一のバンクにアクセスしたケース。同一の配列要素、スカラ変数を複数のタスクで参照しているときなどに発生する。
- 同じノード上で実行されている別のプログラムが、たまたま同一のバンクに繰り返しアクセスするケース

一般的には、自動並列化、OpenMP 並列化されたプログラムの複数のタスクから同時に同一のバンク

にアクセスした場合にメモリネットワーク競合が発生しやすくなります。

4.2 プログラム実行解析情報

環境変数 F_PROGINF または C_PROGINF に DETAIL を指定した場合のプログラム実行解析情報の出力例を図 4 に示します。

網掛け部分の「CPU ポート競合（秒）」に CPU ポート競合時間が、「メモリネットワーク競合（秒）」にメモリネットワーク競合時間が、それぞれ秒単位で表示されます。

```
            ******  プログラム情報  ******
経過時間（秒）                  :          30.039067
ユーザ時間(秒)                  :          28.297595
システム時間（秒）              :           0.017765
ベクトル命令実行時間（秒）      :          28.263599
全命令実行数                    :       11892009287.
ベクトル命令実行数              :        6099457558.
ベクトル命令実行要素数          :     1528191109757.
浮動小数点データ実行要素数      :      880815869041.
MOPS 値                         :       54208.977057
MFLOPS 値                       :       31126.881228
平均ベクトル長                  :         250.545412
ベクトル演算率（%）             :          99.622385
メモリ使用量（MB）              :         512.031250
MIPS 値                         :         420.248061
命令キャッシュミス（秒）        :           0.002944
オペランドキャッシュミス（秒）  :           0.011593
バンクコンフリクト時間
  CPU ポート競合（秒）          :           0.455945
  メモリネットワーク競合（秒）  :           2.416176

開始時刻（日付）        :  2008年  9月 19日 金曜日 17:57:42 JST
終了時刻（日付）        :  2008年  9月 19日 金曜日 17:58:12 JST
```

図 4　プログラム情報の表示例

4.3 簡易性能解析機能

以下に簡易性能解析機能(ftrace)の出力例を図 5 に示します。

網掛けされた欄の「CPU PORT」に CPU ポート競合時間、「NETWORK」にメモリネットワーク競合時間が、それぞれ秒単位で表示されます。

```
*-------------------------*
  FLOW TRACE ANALYSIS LIST
*-------------------------*

Execution Date : Fri Sep 19 17:58:12 2008
Total CPU Time : 0:00'28"266 (28.266 sec.)

PROG.NAME  FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS V.OP  AVER.    VECTOR I-CACHE O-CACHE  BANK CONFLICT
                      TIME[sec](  %  )    [msec]                  RATIO V.LEN      TIME    MISS    MISS CPU PORT NETWORK

sub5           1275    23.255( 82.3)    18.239  61136.0  35800.2 99.62 250.2    23.248  0.0007  0.0036   0.034    0.911
sub3            500     4.784( 16.9)     9.567  20427.7   8298.5 99.69 256.0     4.781  0.0002  0.0010   0.061    1.840
sub7             24     0.147(  0.5)     6.124  72093.7  50785.4 99.93 250.3     0.146  0.0000  0.0005   0.001    0.006
main              1     0.039(  0.1)    39.140  47202.8   4145.6 99.52 256.0     0.029  0.0008  0.0040   0.000    0.001
sub3              9     0.031(  0.1)     3.432  49334.3  21004.3 99.79 256.0     0.031  0.0000  0.0000   0.002    0.013
sub6             24     0.007(  0.0)     0.301  67647.3  37418.1 99.57 256.0     0.007  0.0000  0.0000   0.000    0.002
sub4              1     0.003(  0.0)     2.501  43833.8  21628.6 99.80 250.3     0.002  0.0000  0.0000   0.000    0.000
sub10             1     0.001(  0.0)     0.892    150.6      1.8  0.96   7.6     0.000  0.0001  0.0002   0.000    0.000
sub8              1     0.001(  0.0)     0.563    150.5      0.8 39.48 244.1     0.000  0.0002  0.0002   0.000    0.000
sub1              1     0.000(  0.0)     0.245  18648.4      0.0 98.46 256.0     0.000  0.0000  0.0000   0.000    0.000
sub9              1     0.000(  0.0)     0.004    175.4     70.7 58.62  36.0     0.000  0.0000  0.0000   0.000    0.000
-------------------------------------------------------------------------------------------------------------------------
total          1838    28.267(100.0)    15.379  54268.1  31160.9 99.62 250.5    28.245  0.0020  0.0094   0.098    2.773
```

図 5　簡易性能解析情報の表示例

5. おわりに

本稿では、FORTRAN90/SX と C++/SX の新機能について紹介いたしました。ここで紹介した各機能が皆様のプログラムの開発やチューニングの一助となれば幸いです。

弊社は SX-9 のハードウェア性能をさらに引き出すため、今後も継続してコンパイラの最適化および言語機能の強化・改善を行っていく予定です。

参考文献

[1] 横谷雄司、工藤淑裕他,“SX-9 の言語処理系”,NEC 技報,Vol.61,No.4,pp.49-53,2008