プログラムの実行性能を調べてみましょう

~ VPP5000/64 の解析ツール紹介 ~

渡部 善隆 *

本稿は，2001 年 1 月に導入されたスーパーコンピュータ VPP5000/64 (ホスト名: kyu-vpp) で利用できる実行性能解析ツール「アナライザ」の紹介記事です．

出力されるデータは実行性能改善のためだけでなく，使われていない関数や条件文の調査など，デバッグやアルゴリズムの効率化にも利用することができます．利用方法は簡単です．この記事を参考に，ぜひ一度お手持ちのプログラムを解析してみてはいかがでしょうか．

1 実行性能解析ツールの概要

1.1 解析を行なう利点

以下に，手持ちのプログラムの実行性能を解析する利点を列挙します．

- プログラム中の副プログラム (サブルーチン・関数)，ループがどのくらい全体の中で活躍しているのかを知ることができます．多くのプログラムでは，限られた手続きにコストが集中することが知られています．これらの情報をもとに，次の性能改善のステップ (ベクトル化・並列化の促進など) に進むことができます．
- ループがベクトル化されているかどうかを調べることで，自分のプログラムがベクトル並列計算機に向いているかどうかを知ることができます．
- 並列プログラム場合，データ転送効率や並列化効率を調べることができます．
- 一般に，ベクトル化率が高いからといって，必ずしも計算機の性能を十分に引き出しているとはいえません．アナライザの機能のひとつである pa を用いると，自分のプログラムの実行性能が理論最大性能と比較してどのくらいの割合かを知ることができます．
- 実行結果だけでなく，コストの分析という違う切口からプログラムを見ることで，アルゴリズムの効率化やデバッグに役立つことがあります．

*九州大学情報基盤センター研究部　E-mail: watanabe@cc.kyushu-u.ac.jp

1.2 VPP5000/64 で利用できる実行性能解析ツール

アナライザの実行性能解析機能は，4 つのコンポーネント[1]で構成されています．使用可能なプログラム言語との対応表は以下の通りです．

アナライザの実行性能解析機能の構成

コンポーネント	機能の概要	Fortran	C	C++	VPP Fortran	HPF	MPI
pa	ハードウェアの稼働状況測定	○	○	○	○	○	○
サンプラ	負荷分布，ベクトル化・並列化情報	○	○	○	○	○	–
カウンタ	サンプラより詳細な実行状態の解析	○	○	–	○	–	–
sa	HPF プログラムの静的解析	–	–	–	–	○	–

サンプラ，カウンタ，pa は実行可能プログラムを実際に実行させて，コストの高い手続きやループなどの様々な性能情報を取得します．sa は実行は行なわず，ソースプログラムの内容から性能情報を採取します．

sa はデータ並列化 Fortran である HPF(High Performance Fortran) でのみ利用できます．本稿では sa の使用方法については触れません．詳しくはセンターホームページのオンラインマニュアルを参照してください．

2 pa(performance analyzer) の利用方法

pa はプログラムの実行時におけるハードウェアの稼働状況を測定します．これらの情報によって，プログラムの演算量，演算効率，データ転送量などを把握することができます．また，手続きの呼び出し関係を表示することもできます．

pa は，Fortran, VPP Fortran, HPF, C, C++, MPI で利用できます．

2.1 起動コマンド

pa の起動コマンドは `pa(/usr/lang/bin/pa)` です．pa による測定のために特定の翻訳時オプションを指定した実行可能ファイルを用意する必要はありません．通常使用する実行可能ファイルが利用できます．

pa コマンドの主なオプションは次の通りです．詳細は `man` コマンド (`man pa`) で確認することができます．

`-Gon`	コールグラフ機能を実行
`-Lproc`	手続き単位の情報を測定
`-Lrange`	利用者が指定した範囲の情報を測定
`-Don`	データ転送装置の測定を行なう (並列処理)
`-Pdetail`	各プロセッサ毎の情報を出力 (並列処理)

[1]「構成要素」「成分」を意味する英語 “component” から来ています．

2.2 利用例

2.2.1 対話型処理

pa コマンドに続けて，pa のオプションと実行可能ファイル名を 1 個以上の空白で区切って入力します．

```
kyu-vpp% pa a.out  ↵            <--- 実行と解析
                 :
            (通常の実行結果)
                 :
            (pa の解析結果)
                 :
```

解析結果は標準出力に書き出されます．

以下は，-Lproc オプションを指定した例です．解析結果はファイル "out" に保存しています．

```
kyu-vpp% pa -Lproc a.out > out  ↵        <--- 実行と解析．ファイルに保存
```

2.2.2 バッチ処理

解析結果は標準出力にファイルとして保存されます．

```
#                                  <--- csh で記述
cd EXAMPLE                         <--- ディレクトリの移動
pa a.out                           <--- 実行情報の採取
```

バッチ処理についての詳細は

http://www.cc.kyushu-u.ac.jp/scp/system/general/VPP5000/

を参照してください．

2.3 解析例

主な項目をあげます．

Real(sec)	実行経過時間 (秒)
SU(sec)	CPU 動作時間 (秒)
VU(sec)	ベクトル命令動作時間 (秒)
SU	スカラー演算数 (整数・浮動小数点数毎)
VU	ベクトル演算数 (整数・浮動小数点数毎)
MOPS	演算速度
MFLOPS	浮動小数点演算速度
Vect.Ratio	ベクトル化率
Average V.L.	平均ベクトル長

2.3.1 プログラムの解析例 (省略時オプション)

```
--- Performance Analyzer (Serial)--------------Date:2001.07.17 Time:09.35.52---
 Summary        Program : a.out
--------------------------------------------------------------------------------
 Time :         Real(sec)       SU(sec)          VU(sec)          Count
              9.09350e-01     5.11301e-01     4.95516e-01      1.00000000e+00
      --------------------------------------------------------------------------
 Arith./Logical Operations:     SU                 VU              Total
          Integer    -   1.25324930e+07     1.04600000e+04     1.25429530e+07
          Floating   -   7.72240000e+04     1.40828785e+09     1.40836507e+09
      --------------------------------------------------------------------------
 Performance :      MOPS          MFLOPS       Vect.Ratio       Average V.L.
                2.77900e+03   2.75447e+03       0.9911             274
--------------------------------------------------------------------------------
 Pipeline Utilization :  Memory Access   Add/Mult/etc.   Div/Sqrt      Mask
         Active         -   0.7326          0.5335        0.0071      0.0000
         Efficiency     -   0.5248          0.5407        0.0389      0.1278
--------------------------------------------------------------------------------
 CPU Statistics :               SU                 VU             Average
         Mega-Inst./sec -  2.30939e+01       1.38255e+01      3.69193e+01
         Mega-Op./sec   -  4.10616e+01       1.39175e+01      5.49791e+01
         Op./Inst.      -  1.77803e+00       1.00665e+00      1.48917e+00
      --------------------------------------------------------------------------
 Cache Information :       L1-Cache Hit      L2-Cache Hit      Cache Miss
         Instruction    -    0.99964           0.00031           0.00006
         Operand        -    0.89154           0.03120           0.07726
--------------------------------------------------------------------------------
```

★とりあえず，どこを見ればいいか？

プログラムの性能の目安として重要な数値のひとつとして，“Performance” の “MFLOPS” をあげることができます．

これは Mega Floating Operation Per Second の略で，1 秒間に何回の浮動小数点演算が実行されたかを示します．上の例では

$$\begin{aligned} \texttt{2.75447e+03}\ \mathrm{MFLOPS} &= 2754.47 \quad \mathrm{MFLOPS} \\ &= 2.754 \quad \mathrm{GFLOPS} \end{aligned}$$

となります．GFLOPS の “G” は Mega の上の単位の Giga を表します．この値が VPP5000/64 の理論最大性能 9.6GFLOPS (9,600MFLOPS) に近いほど，計算機の性能を引き出しているといえます．

ただし，計算式は浮動小数点演算総数をすべての PE の <u>総 CPU 動作時間</u> で除して得られた値になります．上の例では

$$\texttt{1.40836507e+09} \,/\, \texttt{5.11301e-01} \,/\, 1000 \,/\, 1000 \approx 2754.47$$

となります．特に並列プログラムの場合には，PE 間の通信時間も考慮して除する時間は経過時間 (表の Real) を採用することがあります．しかし，センターの計算機は他のジョブと CPU を共有する多重度制を採用しているため，必ずしも経過時間が一定ではないことに注意してください．

2.3.2 プログラムの解析例 (-Lproc オプション)

-Lproc オプションを指定すると，サブルーチン，関数などの手続き毎の性能も測定します．指定しない場合にはプログラム全体の性能を測定します．

```
--- Performance Analyzer (Serial)--------------Date:2001.07.17 Time:09.38.16---
 Summary        Program : a.out
--------------------------------------------------------------------------------
 Time :        Real(sec)       SU(sec)          VU(sec)          Count
            8.81694e-01     5.10945e-01     4.94051e-01      1.00000000e+00
     ---------------------------------------------------------------------------
 Arith./Logical Operations:    SU                VU                Total
          Integer    -   1.25333280e+07     1.04600000e+04     1.25437880e+07
          Floating   -   7.72250000e+04     1.40828785e+09     1.40836508e+09
     ---------------------------------------------------------------------------
 Performance :      MOPS          MFLOPS       Vect.Ratio       Average V.L.
               2.78094e+03   2.75639e+03       0.9911             274
--------------------------------------------------------------------------------
 Pipeline Utilization :  Memory Access   Add/Mult/etc.   Div/Sqrt      Mask
         Active         -   0.7303          0.5339         0.0070      0.0000
         Efficiency     -   0.5269          0.5408         0.0382      0.1278
--------------------------------------------------------------------------------
 CPU Statistics :              SU                VU              Average
         Mega-Inst./sec -  2.31155e+01       1.38351e+01      3.69506e+01
         Mega-Op./sec   -  4.10971e+01       1.39272e+01      5.50242e+01
         Op./Inst.      -  1.77790e+00       1.00665e+00      1.48913e+00
     ---------------------------------------------------------------------------
 Cache Information :       L1-Cache Hit      L2-Cache Hit      Cache Miss
         Instruction    -    0.99921           0.00073           0.00006
         Operand        -    0.89087           0.03186           0.07728
--------------------------------------------------------------------------------


--- Performance Analyzer (Serial)--------------Date:2001.07.17 Time:09.38.16---
 Summary        Procedure : a.out->mul_
--------------------------------------------------------------------------------
 Time :        Real(sec)       SU(sec)          VU(sec)          Count
            1.38574e-01     1.31844e-01     1.27345e-01      4.00000000e+00
     ---------------------------------------------------------------------------
 Arith./Logical Operations:    SU                VU                Total
          Integer    -   2.92058700e+06     0.00000000e+00     2.92058700e+06
          Floating   -   0.00000000e+00     1.04319900e+08     1.04319900e+08
     ---------------------------------------------------------------------------
 Performance :      MOPS          MFLOPS       Vect.Ratio       Average V.L.
               8.13392e+02   7.91240e+02       0.9728             109
--------------------------------------------------------------------------------
 Pipeline Utilization :  Memory Access   Add/Mult/etc.   Div/Sqrt      Mask
         Active         -   0.3064          0.4903         0.0000      0.0000
         Efficiency     -   0.2716          0.1682         0.0000      0.0000
--------------------------------------------------------------------------------
 CPU Statistics :              SU                VU              Average
         Mega-Inst./sec -  1.71173e+01       9.15382e+00      2.62711e+01
         Mega-Op./sec   -  3.19518e+01       9.15382e+00      4.11057e+01
         Op./Inst.      -  1.86664e+00       1.00000e+00      1.56467e+00
     ---------------------------------------------------------------------------
 Cache Information :       L1-Cache Hit      L2-Cache Hit      Cache Miss
         Instruction    -    0.99996           0.00004           0.00000
         Operand        -    0.99915           0.00085           0.00000
--------------------------------------------------------------------------------
```

2.3.3 プログラムの解析例 (-Gon オプション)

-Gon オプションを指定すると「コールグラフ」と呼ばれる，手続きの呼び出し (call) 過程が表示されます．

```
--------------------------------------------------------------------------------
 Count     Elaps         CPU         Nest  Call-graph  PE=0  Program:a.out
---------+-------------+-------------+---+--------------------------------------
        1         0.000  1.09960e-03    0 MAIN__ : : stokes.f
                  0.000 (5.10277e-01)       |
        1         0.001  1.56204e-03    1   +-- makeg_ : 77 : stokes.f
                  0.001 (1.65476e-03)       |     |
        1         0.001  6.03556e-05    2   |     +-- clearm_ : 246 : stokes.f
                  0.001 (6.03556e-05)       |     |
        1         0.001  8.80374e-07    2   |     +-- elmate_ : 246 : stokes.f
                  0.001 (8.80374e-07)       |     |
        1         0.001  1.19051e-06    2   |     +-- elmatd_ : 246 : stokes.f
                  0.001 (1.19051e-06)       |     |
       64         0.001  3.02895e-05    2   |     +-- elnode_ : 251 : stokes.f
                  0.004 (3.02895e-05)       |
        1         0.004  1.67648e-04    1   +-- makegi_ : 77 : stokes.f
                  0.004 (8.26929e-02)       |     |
        1         0.004  4.29716e-05    2   |     +-- clearm_ : 329 : stokes.f
                  0.004 (4.29716e-05)       |     |
        1         0.004  8.08677e-06    2   |     +-- clearm_ : 329 : stokes.f
                  0.004 (8.08677e-06)       |     |
        1         0.004  4.29683e-05    2   |     +-- clearm_ : 329 : stokes.f
                  0.004 (4.29683e-05)       |     |
        1         0.004  4.29683e-05    2   |     +-- clearm_ : 329 : stokes.f
                  0.004 (4.29683e-05)       |     |
        1         0.004  7.77373e-02    2   |     +-- minvw_ : 329 : stokes.f
                  0.004 (7.77373e-02)       |     |
        1         0.308  4.65089e-03    2   |     +-- mulmmw_ : 371 : stokes.f
                  0.308 (4.65089e-03)       |
                          :
```

2.3.4 並列プログラムの解析例 (-Don オプション)

並列プログラムの解析時に -Don オプションを指定すると，各 PE のデータ転送に関する性能を測定することができます．

```
--------------------------------------------------------------------------------
 DTU Statistics :   Data(MB)      Time(sec)    Throughput(MB/sec)  Efficiency
    Send    -     8.94678e+03   5.71385e+00     1.56580e+03        0.9786
    Receive -     8.94355e+03   5.62730e+00     1.58931e+03        0.9933
      --------------------------------------------------------------------------
 Send Information :
    Request Time(sec) - 5.71392e+00     Service Time(sec) -  5.71385
            --------------------------------------------------------------------
    Service Time      -   Data Transfer     Send Setup     Network Blocked
       1.0000              0.9786           0.0000           0.6599
--------------------------------------------------------------------------------
```

3　サンプラの利用方法

サンプラは，Fortran, VPP Fortran, HPF, C, C++ で翻訳・編集結合して作成した実行可能ファイルに対し，実行中に一定の周期で割り込みをかけ，プログラム単位，手続き単位，ループや配列式ごとの実行コストのデータを収集します．手続きごとのベクトルヒット率の測定，動的にリンクされたプログラムの解析も可能です．

利用者は実行解析用に特別に再翻訳などを行う必要はなく，実行可能ファイルをそのままサンプラに渡すことでプログラムの解析情報を得ることができます．

ただし，実行中に割り込みをかけながら情報を収集するため，実行時間が増えることにご注意ください．

3.1　サンプラを使用する前に

サンプラを起動する前に実行可能ファイルを用意します．特別な翻訳時オプションの指定は必要ありません．各言語の実行可能ファイルの作成方法はセンターホームページを参照してください．

3.2　環境変数の設定

サンプラを使用して情報収集，解析を行うために，環境変数名 `FJSAMP` に変数を設定する必要があります．

設定できる変数は以下の通りです．

`file:`*filename*	*filename* にサンプラの解析用の情報を出力する．バイナリファイル．
`type:rtime`	経過時間をもとにした情報を出力する．省略した場合は CPU 時間をもとにした情報を出力する．
`interval:`*time*	タイマーの割り込み間隔 (10 ~ 2147480) をミリ秒単位で指定．省略値は 10．
`pe:on`	各プロッセッシングエレメント (PE) ごとの情報を出力することを指定．並列プログラムの場合のみ有効．

環境変数の設定はバッチリクエストの記述が `sh`(先頭に `#` がない) か `csh`(先頭に `#` がある) かに応じて設定が異なります．使用例を参照してください．

○注意事項

- サンプラ情報ファイルは実行時間がかかるほど大きくなります．
- 長時間実行するプログラムは `interval` の値を大きくすることで情報量を少なくすることができます．

- 実行時間が極端に短い並列プログラム場合，逐次プログラムの情報が収集されてしまうことがあります．
- 周期的な動作をするプログラムの場合には，偏った測定結果が得られる場合かあります．

3.3　解析コマンド

サンプラによる解析は fjsamp(/usr/lang/bin/fjsamp) コマンドによって行います．fjsamp に続けて実行可能ファイル名を指定します．実行時オプションも指定できます．

Fortran の経過時間による実行打ち切りを指定する -Wl,-t オプションは指定できません．

3.4　対話型処理の例

1PE のジョブに限って対話的にサンプラを利用できます．

• Fortran の情報収集例

ソースプログラム “test.f90” を翻訳・編集結合し，実行可能ファイル “a.out” を作成し，サンプラによって実行解析を行ないます．解析情報ファイル名は “sampler.data” とします．

```
kyu-vpp% frt test.f90 ↵                           <--- 実行可能ファイルの作成
kyu-vpp% setenv FJSAMP file:sampler.data ↵       <--- 環境変数の設定
kyu-vpp% ./a.out ↵                                <--- サンプラ情報ファイルの作成
kyu-vpp% fjsamp a.out > out ↵                     <--- サンプラによる解析
```

解析結果は標準出力に出力されます．ここではリダイレクション機能を用いてファイル “out” に結果を書き出しています．

• C の情報収集

C の場合も同様に，作成した実行可能ファイルを一度実行することでサンプラ情報を作成し fjsamp による解析を行います．

```
kyu-vpp% cc -Kvp test.c ↵                         <--- 実行可能ファイルの作成
kyu-vpp% setenv FJSAMP file:sampler.data ↵       <--- 環境変数の設定
kyu-vpp% ./a.out ↵                                <--- サンプラ情報ファイルの作成
kyu-vpp% fjsamp a.out > out ↵                     <--- サンプラによる解析
```

対話的に情報を収集できる実行可能ファイルは記憶容量が 1GB 以下の非並列ジョブに限られます．それ以外の情報収集はバッチ処理になります．

3.5　バッチリクエストの記述例

以下は VPP5000/64 に投入するバッチリクエストの記述例です．実行可能ファイル “a.out” は作成済みだとします．

●情報収集 + 解析

```
#                                    <---csh で記述
cd EXAMPLE                           <--- ディレクトリの移動
setenv FJSAMP file:sampler.data      <--- 環境変数の設定
./a.out                              <--- サンプラ情報ファイルの作成
fjsamp a.out                         <--- サンプラによる解析
```

最初の a.out の実行により，“sampler.data” に情報が収集されます．ファイル名は何でも構いません．次の fjsamp で，収集された情報をもとに解析を行います．通常，実行時間は情報を収集するための a.out の処理に集中します．

スクリプトを sh で記述する (先頭に # がない) 場合は環境変数の設定が少し異なります．

●情報収集と解析を分ける

情報収集と解析とは分けてバッチリクエストに記述し，順番に処理しても構いません．ただし情報ファイル名 (ここでは “sampler.data”) は同じにします.

```
#
cd EXAMPLE
setenv FJSAMP file:sampler.data         <--- 環境変数の設定
./a.out                                 <--- サンプラ情報ファイルの作成
```

情報ファイル sampler.data を作成した後に，fjsamp により解析します．

```
#                                              <---csh で記述
setenv FJSAMP file:sampler.data                <--- 環境変数の設定
fjsamp a.out                                   <--- サンプラによる解析
```

変数のうち “file” は情報収集，解析とも必ず指定します．“type”，“pe” は情報収集の段階では指定不要です．

●並列プログラム (情報収集 + 解析)

並列プログラムの場合，環境変数に pe:on を指定することで各プロセッサの情報を取得できます．

```
#                                       <---csh で記述
setenv FJSAMP file:sampler.data,pe:on   <--- 環境変数の設定，各 PE の情報を出力
./a.out                                 <--- サンプラ情報ファイルの作成
fjsamp a.out                            <--- サンプラによる解析
```

なお，実行可能ファイル (上の例では “a.out”) は 並列化オプション (たとえば VPP Fortran の場合は frt コマンドのオプション-Wx) によって翻訳されている必要があります．

3.6 サンプラの解析例

3.6.1 逐次版 Fortran プログラムの解析例

手続きやループ / 配列式の全体に占める割合と平均ベクトル長が主な情報となります．平均ベクトル長が長いほどベクトル計算機の性能を十分に引き出していると考えて結構です．

```
Status                      : Serial                  <--- 逐次実行
Number of Processors        : 1                       <--- PE 数

Type                        : cpu                     <--- CPU 時間で情報を採取
Interval (msec)             : 10                      <--- 割り込み間隔

Synthesis Information                                 <--- 総合情報
    Count|   Percent(Accum)|    VL| V_Hit(%)| Name           Count  : 割り込み回数
       14|       27.5( 27.5)|   109|     92.9|  MUL_        Percent  : 割合 (%)
        9|       17.6( 45.1)|   330|     88.9|  HOBSVW_          VL  : 平均ベクトル長
        9|       17.6( 62.7)|   188|     88.9|  GHBSVW_        Name  : 副プログラム名
        8|       15.7( 78.4)|   493|    100.0|  MINVW_
        6|       11.8( 90.2)|   450|    100.0|  MULMMW_
        4|        7.8( 98.0)|     -|    100.0|  CHOLFW_
        1|        2.0(100.0)|   450|    100.0|  MAKEGI_

       51|                 |   287|         |  TOTAL


Program Unit Information(MUL_)-----------------------------------
 Procedure List:                                            <--- 手続き情報
    Count|          Percent|    VL| V_Hit(%)| Name
       14|     100.0( 27.5)|   109|     92.9|  mul_

       14|          ( 27.5)|   109|         |  PROCEDURE_TOTAL

 Loop & Array Expression List:                              <--- ループ / 配列式情報
    Count|          Percent| V_Mark| kind  |    VL| Line
       13|       92.9( 25.5)|   V   | DO    |   109|  00001998-00002000
        1|        7.1(  2.0)|   S   | DO    |     -|  00001996-00002002

Program Unit Information(HOBSVW_)--------------------------------
 Procedure List:
    Count|          Percent|    VL| V_Hit(%)| Name
        9|     100.0( 17.6)|   330|     88.9|  hobsvw_

        9|          ( 17.6)|   330|         |  PROCEDURE_TOTAL

 Loop & Array Expression List:
    Count|          Percent| V_Mark| kind  |    VL| Line
        4|       44.4(  7.8)|   V   | DO    |   317|  00001499-00001500
        3|       33.3(  5.9)|   V   | DO    |   346|  00001514-00001518
        1|       11.1(  2.0)|   V   | DO    |     -|  00001557-00001559
        1|       11.1(  2.0)|   S   | DO    |     -|  00001592-00001606
                          :
```

3.6.2 VPP Fortran プログラムの解析例

重要な項目をまとめました．表中の x はその項目の値を変数としたものです．詳細はオンラインマニュアルを参照してください．

`Parallel speedup` 並列効果
: $1 \leq x \leq$ プロセッサ数．値が大きいほど並列効果が高い．

`Parallelization ratio` 並列化率
: $0 \leq x \leq 1$．値が大きいほど並列化されている．HPF プログラムの場合常に 1．

`Parallel to Serial ratio` 並列加速率
: $1 \leq x \leq$ プロセッサ数．逐次実行した場合と比較した性能向上の比率．冗長実行が少ないプログラムではプロセッサ数に近い値となる．

`Load balance` 負荷バランス
: $0 \leq x \leq 1$．値が小さいほどプロセッサが効率よく動作している．

`Asynchronous transfer` 非同期転送待ち発生率
: $0 \leq x \leq 1$．値が小さいほど演算とデータ転送が効率良く行なわれている．

`ALL` 割込み回数
: プログラム全体の割り込み回数．

`AW` 待ち状態割込み回数
: 回数が少ないほどよい．

`AMW` 転送待ち状態割込み回数
: 回数が少ないほどよい．

`VL` 平均ベクトル長
: 2048 に近いほど実行効率が高い．

```
Status                       : Parallel
Number of Processors         : 16

Type                         : cpu
Interval (msec)              : 10

 Performance Information :      Parallel Information :
      Parallel                  Parallelization  Parallel to serial  Load balance  Asynchronous    Name

          speedup                         ratio        speed ratio                 transfer ratio

      15.91681416                    0.99985918        15.95387890    0.00011085      0.00011085  DP_UALUX_
      16.00000000                    1.00000000        16.00000000    0.00000000      0.00000000  MAIN__
      14.00000000                    1.00000000        12.47058824    0.00000000      0.00000000  DP_ULUXX_
       1.00000000                    0.00000000         1.00000000    0.00000000      0.00000000  MAIN

      15.88830716                    0.99976572        15.94744832    0.00010946      0.00010946  TOTAL

Synthesis Information (Count)
       PM        PMW       PMMW|         RM        RMW       RMMW|       ALL         AW        AMW|    VL| Name
      565          0          0|       8995          1          1|      9021          1          1|   847| DP_UALUX_
        6          0          0|         96          0          0|        96          0          0|   399| MAIN__
        1          0          0|         14          0          0|        18          0          0|  1336| DP_ULUXX_
        1          0          0|          1          0          0|         1          0          0|   150| MAIN

      573          0          0|       9106          1          1|      9136          1          1|   843| TOTAL

Synthesis Information (Percent)
       PM        PMW       PMMW|         RM        RMW       RMMW|       ALL         AW        AMW| Name
     98.6        0.0        0.0|       98.8        0.0        0.0|      98.7        0.0        0.0| DP_UALUX_
      1.0        0.0        0.0|        1.1        0.0        0.0|       1.1        0.0        0.0| MAIN__
      0.2        0.0        0.0|        0.2        0.0        0.0|       0.2        0.0        0.0| DP_ULUXX_
      0.2        0.0        0.0|        0.0        0.0        0.0|       0.0        0.0        0.0| MAIN


Program Unit Information(DP_UALUX_)------------------------------
 Performance Information :      Parallel Information :
      Parallel                  Parallelization  Parallel to serial  Load balance  Asynchronous    Name
          speedup                         ratio        speed ratio                 transfer ratio

      15.91681416                    0.99985918        15.95387890    0.00011085      0.00011085  dp_ualux_

      15.91681416                    0.99985918        15.95387890    0.00011085      0.00011085  PROCEDURE_TOTAL

 Procedure List (Count):
       PM        PMW       PMMW|         RM        RMW       RMMW|       ALL         AW        AMW|    VL| Name
      565          0          0|       8995          1          1|      9021          1          1|   847| dp_ualux_

 Procedure List (Percent):
       PM        PMW       PMMW|         RM        RMW       RMMW|       ALL         AW        AMW| Name
    100.0        0.0        0.0|      100.0        0.0        0.0|     100.0        0.0        0.0| dp_ualux_
   ( 98.6)    (  0.0)    (  0.0)    ( 98.8)    (  0.0)    (  0.0)    ( 98.7)    (  0.0)    (  0.0)

 Loop & Array Expression List:
       PM         RM        ALL|         Percent| V_Mark| kind  |    VL| Line
      157       2371       2371|      27.8( 27.4)|   V   | DO    |  1005|  00001116-00001133
      148       2344       2344|      26.2( 25.8)|   V   | DO    |  1004|  00000894-00000911
      110       1783       1783|      19.5( 19.2)|   V   | DO    |   637|  00000969-00000986
      106       1800       1800|      18.8( 18.5)|   V   | DO    |   639|  00001198-00001215
        6        101        101|       1.1(  1.0)|   V   | DO    |  1006|  00000920-00000937
        5        105        105|       0.9(  0.9)|   V   | DO    |  1006|  00001146-00001163
        4         57         57|       0.7(  0.7)|   V   | DO    |   668|  00001228-00001245
        4         67         67|       0.7(  0.7)|   V   | DO    |   491|  00000608-00000613
        4         73         73|       0.7(  0.7)|   V   | DO    |   595|  00000999-00001016
        3         32         32|       0.5(  0.5)|   S   | DO    |     -|  00001114-00001134
        3         19         19|       0.5(  0.5)|   V   | DO    |  2048|  00000292-00000296
        3         24         24|       0.5(  0.5)|   S   | DO    |     -|  00000527-00000554
        3         15         15|       0.5(  0.5)|   S   | DO    |     -|  00000559-00000615
        2         20         20|       0.4(  0.3)|   S   | DO    |     -|  00000892-00000912
        0         16         16|       0.0(  0.0)|   S   | DO    |     -|  00000848-00000856
```

3.6.3　C プログラムの解析例

```
Status                       : Serial                  <--- 逐次実行
Number of Processors         : 1                       <--- PE 数

Type                         : cpu                     <--- CPU 時間で情報を採取
Interval (msec)              : 10                      <--- 割り込み間隔

Synthesis Information                                  <--- 総合情報
    Count|   Percent(Accum)|    VL| V_Hit(%)| Name        Count : 割り込み回数
    31558|      99.8( 99.8)|     -|      0.0|  laplace
       48|       0.2(100.0)|     -|      0.0|  strline    Percent : 割合 (%)
        1|       0.0(100.0)|     -|      0.0|  main            VL : 平均ベクトル長
    31607|                 |     -|         |  TOTAL         Name : 副プログラム名


Function Information(laplace)------------------------------------
 Function List:                                            <--- 手続き情報
    Count|          Percent|    VL| V_Hit(%)| Name
    31558|     100.0( 99.8)|     -|      0.0|  laplace

 Loop List:                                                <--- ループ情報

    Count|          Percent| V_Mark| kind  |    VL| Line
    31473|      99.7( 99.6)|   S   | for   |     -|  00000035-00000043
       85|       0.3(  0.3)|   S   | for   |     -|  00000033-00000044

Function Information(strline)------------------------------------
 Function List:
    Count|          Percent|    VL| V_Hit(%)| Name
       48|     100.0(  0.2)|     -|      0.0|  strline

 Loop List:
    Count|          Percent| V_Mark| kind  |    VL| Line
       48|     100.0(  0.2)|       | for   |     -|  00000056-00000056

Function Information(main)---------------------------------------
 Function List:
    Count|          Percent|    VL| V_Hit(%)| Name
        1|     100.0(  0.0)|     -|      0.0|  main

 Loop List:
    Count|          Percent| V_Mark| kind  |    VL| Line
        1|     100.0(  0.0)|       | while |     -|  00000069-00000072
```

3.7　注意事項

- DO ループを 1 つの端末文で共有している場合，最も内側のループに情報が集中することがあります．
- 1 行に複数の文を書くと，情報が正しく収集されない場合があります．
- include 文に実行文がある場合，ループ・配列に関する情報が正しくないことがあります．

4　カウンタの利用方法

カウンタは，逐次版の Fortran, C および HPF プログラムに対し，手続き，ループ，文の実行回数，ループの回転数， IF 文の正しかった割合などの詳細な実行状況をプログラムリストと共に表示するツールです．ここでは，逐次版の Fortran, C の利用方法を説明します．

カウンタを使用するためには翻訳時オプション `-Wc` の 指定が必要です．ただし，`-Wc` オプションの指定によりカウンタ用のオブジェクトコードが挿入されるため，通常より実行性能が劣化 します[2]．

4.1　実行可能ファイルの作成

カウンタによる情報収集を行なうため，翻訳時オプション `-Wv` を指定して実行可能ファイルを作成します．

• Fortran の実行可能ファイル作成例

ソースプログラム “`test.f90`” を翻訳・編集結合し，実行可能ファイル “`a.out`” を作成します．

```
kyu-vpp% frt -Wv test.f90 ⏎                    <--- 実行可能ファイルの作成
```

• C の実行可能ファイル作成例

ソースプログラム “`test.c`” を翻訳・編集結合し，実行可能ファイル “`a.out`” を作成します．

```
kyu-vpp% cc -Kvp -Wv test.c ⏎                  <--- 実行可能ファイルの作成
```

Fortran の場合，`-Wc` オプションとインライン展開を指示するオプション `-Ne`, `-No` および最適化オプション `-On`, `-O0` とは排他になります．その他，`-Wc` オプションと排他となる翻訳時オプションがあります．詳細はマニュアルを参照してください．

`-Wc` オプションを指定して翻訳すると，実行可能ファイルとともに拡張子 “`.ainf`” の翻訳情報ファイル[3]が作成されます．ファイル名は上の例では “`test.ainf`” になります．

4.2　環境変数の設定

カウンタを使用して情報収集，解析を行うために，環境変数名 `FJCNT` に変数を設定する必要があります．

設定できる変数は以下の通りです．

[2]この点がサンプラとの大きな違いです．サンプラは通常の実行ファイルに対し解析ができるため，実行性能はそれほど変わりません．ただし，カウンタの方がより詳細なチューニング情報を収集しますので，状況に応じて使い分けることをお勧めします．

[3]バイナリファイルなので中身は参照できません．

`file:`*filename*	*filename* にカウンタの解析用の情報を出力する．バイナリファイル．
`dtlist:on`	各プロッセッシングエレメント (PE) ごとのデータ転送情報を出力することを指定．HPF プログラムの場合のみ有効．

環境変数の設定はバッチリクエストの記述が `sh`(先頭に `#` がない) か `csh`(先頭に `#` がある) かに応じて設定が異なります．使用例を参照してください．

○注意事項

- 通常の実行可能ファイルと比較して，`-Wc` オプションの指定により，本来ベクトル化される部分がスカラー実行されることがあります．このため，実行結果が異なる可能性があります．
- カウンタ用に作成した実行可能ファイルを用いてサンプラを起動することはできません．
- インクルードファイルに記述された実行文に関する情報は得ることができません．

4.3 解析コマンド

カウンタによる解析は `fjsamp(/usr/lang/bin/fjsamp)` コマンドによって行います．`fjsamp` に続けて翻訳情報ファイル名を指定します．

4.4 対話型処理の例

• Fortran の情報収集例

ソースプログラム “`test.f90`” を翻訳・編集結合し，実行可能ファイル “`a.out`” を作成し，カウンタによって実行解析を行ないます．解析情報ファイル名は “`counter.data`” とします．翻訳情報ファイル名はこの場合 “`test.ainf`” となります．

```
kyu-vpp% frt -Wctest.f90  ↵                    <--- 実行可能ファイルの作成
kyu-vpp% setenv FJCNT file:counter.data  ↵     <--- 環境変数の設定
kyu-vpp% ./a.out  ↵                            <--- カウンタ情報ファイルの作成
              :
          (実行結果)
              :
kyu-vpp% fjsamp test.ainf > out  ↵             <--- カウンタによる解析
```

解析結果は標準出力に出力されます．ここではリダイレクション機能を用いてファイル “`out`” に結果を書き出しています．

- **C の情報収集例**

ソースプログラム “test.c” を翻訳・編集結合し，実行可能ファイル “a.out” を作成し，カウンタによって実行解析を行ないます．ベクトル処理を行なう -Kvp オプションとカウンタの情報収集を行なう-Wc オプションを指定しています．解析情報ファイル名は “counter.data” とします．翻訳情報ファイル名はこの場合 “test.ainf” となります．

```
kyu-vpp% cc -Kvp -Wctest.f90 ↵                <--- 実行可能ファイルの作成
kyu-vpp% setenv FJCNT file:counter.data ↵      <--- 環境変数の設定
kyu-vpp% ./a.out ↵                              <--- カウンタ情報ファイルの作成
                :
          （実行結果）
                :
kyu-vpp% fjsamp test.ainf > out ↵               <--- カウンタによる解析
```

解析結果は標準出力に出力されます．ここではリダイレクション機能を用いてファイル “out” に結果を書き出しています．

4.5 バッチリクエストの記述例

Fortran プログラム “example.f90” に対し，翻訳からカウンタによる解析まで行なうバッチリクエストを記述すると，次の例のようになります． C プログラムも同様です．

```
#                                      <--- csh で記述
cd EXAMPLE                             <--- ディレクトリの移動
frt -Wc example.f90                    <--- -Wc オプションを付け翻訳
setenv FJCNT file:counter.data         <--- 環境変数の設定
./a.out                                <--- 実行情報の採取
fjsamp example.ainf                    <--- カウンタによる解析
```

frt コマンドに -Wc オプションを付加した翻訳の結果，実行可能ファイル “a.out” と翻訳情報ファイル “example.ainf” が生成されます．拡張子 “.ainf” は自動的につきます．翻訳を対話的に行なうことも可能です．

次に csh の setenv サブコマンドによる環境変数を設定します．環境変数名は “FJCNT” です． “file:” に続くファイルに実行情報が出力されます．ファイル名は任意です．ここでは “counter.data” という名前にしています．カウンタによる解析はサンプラと同じ fjsamp コマンドです．翻訳情報ファイル (例では “example.ainf”) を引数に指定します．

fjsamp コマンドの機能は kyu-vpp の man fjsamp でも検索できます．

カウンタの解析結果は標準出力に返却されます．もしバッチリクエストを sh で記述している (即ち先頭の # をつけない) 場合，環境変数の設定は

```
FJCNT=file:counter.data
export FJCNT
```

となります．

4.6 解析結果の出力例

4.6.1 Fortran プログラムの解析例

```
Status                       : Serial                  <-- 逐次プログラム
Number of Processors         : 1                       <--PE 数は 1

Synthesis Information
   Exec-cnt| Loop-leng| Name                           <-- プログラム単位の情報
          2|       464|  MINVW_                             Exec-cnt.... 実行回数
          4|       110|  MUL_                               Loop-leng... 平均ループ長
          3|       415|  MULMMW_                            Name........ 手続き（サブルーチン / 関数）名
          2|       201|  HOBSVW_
          2|       219|  GHBSVW_
          2|       184|  CHOLFW_
                    :

           |       218|  TOTAL

                    :

Program Unit Information(MUL_)+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Procedure List:                                                 <-- プログラム単位の個別情報
   Exec-cnt| Loop-leng| Name
          4|       110|  mul_

           |       110|  PROCEDURE_TOTAL

 Loop & Array Expression List:                                   <-- ループ / 配列式情報
   Loop-exe| Loop-leng| V_Mark| kind   | Line
     477900|       109|   V   | DO    |  00001998 - 00002000
       1431|       334|   S   | DO    |  00001996 - 00002002
       1431|       102|   V   | DO    |  00001992 - 00001995
          4|       358|   S   | DO    |  00001991 - 00002003

 Vectorize Statement List:                                       <-- ソースと比較した文情報

  Line         Exec-cnt  true  v o a
  00001986                           !
  00001987            4                     SUBROUTINE MUL(A,KA,B,KB,C,KC,M,N,L,VW)
  00001988                                  DOUBLE PRECISION A(KA,1),B(KB,1),C(KC,1),VW(1),SUM
  00001989                                  INTEGER KA,KB,KC,M,N,L,N1,JJ,II
  00001990            4                     N1=N+1
  00001991            4        s            DO 30 I=1,M
  00001992         1431        v             DO 10 J=1,N
  00001993       145800        v              JJ=N1-J
  00001994       145800        v              VW(JJ)=A(I,JJ)
  00001995       145800        v      10     CONTINUE
  00001996         1431        s 2           DO 20 J=1,L
  00001997       477900        v 2            SUM = 0.0D0
  00001998       477900        v 2            DO 15 II= 1,N
  00001999     52159950        v 2             SUM = VW(II)*B(II,J) + SUM
  00002000     52159950        v 2    15      CONTINUE
  00002001       477900        v 2            C(I,J) = SUM
  00002002       477900        v 2    20     CONTINUE
  00002003         1431        s      30    CONTINUE
  00002004            4                     RETURN
  00002005            4                     END

Message List:

 vectorization messages:                                          <-- ベクトル化・最適化メッセージ
 Program name(mul_)
    jpc1001i-i  "stokes.f", line 1993 - 1995: この範囲の文は DO 変数 J でベクトル化されました.
    jpc1202i-i  "stokes.f", line 1996: ループ内の実行文は 2 回展開されました.
    jpc2306i-i  "stokes.f", line 1996: 部分ベクトル化による性能向上が得られないため，この DO ループはベクトル化されません.
    jpc1001i-i  "stokes.f", line 1999 - 2000: この範囲の文は DO 変数 II でベクトル化されました.


Program Unit Information(MULMMW_)++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Procedure List:
   Exec-cnt| Loop-leng| Name
          3|       415|  mulmmw_

           |       415|  PROCEDURE_TOTAL
```

4.6.2 C プログラムの解析例

```
Status                      : Serial                                    <-- 逐次プログラム
Number of Processors        : 1                                         <--PE 数は 1

Synthesis Information                                                   <-- プログラム単位の情報
   Exec-cnt| Loop-leng| Name                                                  Exec-cnt.... 実行回数
      26880|       239|  laplace                                              Loop-leng... 平均ループ長
          1|       239|  strline                                              Name........ 手続き（サブルーチン / 関数）名
          1|     26880|  main
          1|       321|  init

           |       239|  TOTAL

File Information(sampler.c)++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Function List:
   Exec-cnt| Loop-leng| Name
      26880|       239|  laplace
          1|       239|  strline
          1|     26880|  main
          1|       321|  init

           |       239|  FILE_TOTAL

 Loop List:                                                                    <-- ループ / 配列式情報
   Loop-exe| Loop-leng| V_Mark| kind  | Line
   10725120|       239|   S    | for   |  00000035 - 00000043
        399|       239|        | for   |  00000053 - 00000058
          1|     26880|        | while |  00000069 - 00000072
      26880|       399|   S    | for   |  00000033 - 00000044
          1|       401|   V    | for   |  00000011 - 00000017
          1|       241|   V    | for   |  00000018 - 00000024
          1|       399|        | for   |  00000051 - 00000059

 Vectorize Statement List:                                                     <-- ソースと比較した文情報

  Line         Exec-cnt  v o
  00000001                    #include <stdio.h>
  00000002                    #define  IXMESH  241
  00000003                    #define  IYMESH  401
  00000004                    #define  CENTER  201
  00000005                    #define  RADIUS  101
  00000006                    double   p[IYMESH][IXMESH],stream[IYMESH][IXMESH];
  00000007
  00000008                    void init()
  00000009            1
  00000010                       int i, j ;
  00000011            1  v       for (i=0 ; i<IYMESH ; i++)
  00000012
  00000013          401  v          p[i][0] = 0.0 ;
  00000014          401  v          p[i][IXMESH-1] = (double)IXMESH-1 ;
  00000015          401  v          stream[i][0] = 0.0 ;
  00000016          401  v          stream[i][IXMESH-1] = (double)IXMESH-1 ;
  00000017               v
  00000018            1  v        for (j=0 ; j<IXMESH ; j++)
  00000019
  00000020          241  v          p[0][j] = (double)j ;
  00000021          241  v          p[IYMESH-1][j] = (double)j ;
  00000022          241  v          stream[0][j] = (double)j ;
  00000023          241  v          stream[IYMESH-1][j] = (double)j ;
  00000024               v
  00000025                        return ;
  00000026            1
  00000027
                                          :
Message List:

 vectorization messages:                                                              <-- ベクトル化・最適化メッセージ
  "sampler.c", line 13 - 17: この範囲の文はループ変数 i でベクトル化されました。
  "sampler.c", line 20 - 24: この範囲の文はループ変数 j でベクトル化されました。
  "sampler.c", line 35: 部分ベクトル化による性能向上が得られないため、このループはベクトル化されません。
  "sampler.c", line 39: p の定義引用が回帰的であるため、この配列はベクトル化されません。
  "sampler.c", line 56: for 文の式 1・式 2・式 3 が複雑であるため、このループはベクトル化されません。
  "sampler.c", line 69: ループ内の演算が複雑なため、このループはベクトル化されません。
```