性能解析ツールScalasca Hands-On セットアップ

2014年3月7日
中村朋健
理化学研究所 計算科学研究機構

京
K computer

作業環境

作業環境の最初

/home/ss/xxx/Scalasca/NPB_original/…

作業環境の最終形

/home/ss/xxx/Scalasca/NPB_original/…
/NPB_Test/…
/NPB_Summary/…
/NPB_Filt/…
/NPB_Trace/…
/NPB_Papi/…
/NPB_Pomp/…
/NPB_User/…

Nas Parallel Benchmarks

1. メールで送信
   springschoolscalasca.tar.gzファイル
2. 環境変数（パス）の設定 (.bashrcなどへ)
   export PATH=/home/ss/scalasca/bin:$PATH
3. piへ転送
   1. sftp, scpなどを使ってpiへ転送
      $ sftp xxxx@pi.ircpi.kobe-u.ac.jp
      $ put SpringSchoolScalasca.tar.gz
4. 展開
   $ ssh xxxx@pi.ircpi.kobe-u.ac.jp
   $ tar zxvf SpringSchoolScalasca.tar.gz

K computer

Nas Parallel Benchmarks

6. コピーを作成
   $ cd ~/Scalasca
   $ cp –r NPB_original NPB_test
7. NPB の config ファイルの設定
   $ cd ~/Scalasca/NPB_test/config
   $ cp make.def.template make.def
   $ cp suite.def.template suite.def
   1. suite.defファイルの編集
      mg A 4
      （この1行だけに）

Nas Parallel Benchmarks

7. NPB の config ファイルの設定
   2. make.defファイルの編集
      32行目 MPIF77 = f77 → **MPIF77 = mpifrtpx**

      39行目 FMPI_LIB = -L/usr/local/lib –lmpi
      ↓
      FMPI_LIB = -L/opt/FJSVfxlang/1.2.1/lib64

      44行目 MPI_INC = -I/usr/local/include
      ↓
      FMPI_INC = -I/opt/FJSVtclang/1.2.1/include/mpi/fujitsu

      49行目 FFLAGS = -O
      ↓
      FFLAGS = -Kfast,parallel,ocl,openmp

      55行目 FLINKFLAGS = -O
      ↓
      FLINKFLAGS = -Kfast,parallel,ocl,openmp

K computer

NPB MGの動かし方

```
$ cd ~/Scalasca/NPB_test
$ make suite
$ cd ~/Scalasca/NPB_test/bin
$ ls -l
```
(作成した実行可能ファイルの確認)

・実行

```
$ pjsub ./run.sh
```

run.sh.oxxxxx を確認

Scalasca Hands-on

K computer

本日のHands-on

1. 基本的なコマンド4つ
2. 自動Instrumentation
    1. Summary解析
    2. フィルタの使い方
    3. Trace解析
    4. PAPIを用いた解析
3. 半自動 Instrumentation
4. 手動 Instrumentation

X Window Systemの設定

Xの設定

1. Mac Mountain Lion 以降
   /Applications/Utilities/Xquartz.app がインストールされているか確認
   http://xquartz.macosforge.org/landing/
2. Mac Mountain Lion より前のバージョンなら問題ないはず
3. Windows
   cygwinでX11をインストール

Xの確認

$ **ssh –Y user@pi.ircpi.kobe-u.ac.jp**

$ **xeyes**

K computer

基本的なコマンド scalasca

$ scalasca
Scalasca 1.4.3
Toolset for scalable performance analysis of large-scale parallel applications
usage: scalasca [-v][-n] {action}

1. prepare application objects and executable for measurement:
   scalasca -instrument <compile-or-link-command> **# skin**
2. run application under control of measurement system:
   scalasca -analyze <application-launch-command> **# scan**
3. interactively explore measurement analysis report:
   scalasca -examine <experiment-archive|report> **# square**

-v: enable verbose commentary
-n: show actions without taking them
-h: show quick reference guide (only)

基本的なコマンド skin

```
$ skin
SCALASCA 1.4.3: application instrumenter
usage: skin [-v] [-comp] [-pomp] [-user] <compile-or-link-command>
  -comp={all|none|...}: routines to be instrumented by compiler [default: all]
                (... custom instrumentation specification depends on compiler)
  -pomp: process source files for POMP directives
  -user: enable EPIK user instrumentation API macros in source code
  -v:    enable verbose commentary when instrumenting
```


K computer

基本的なコマンド scan

$ scan (フロントエンドでは使わない)

```
Scalasca 1.4.3: measurement collection & analysis nexus
usage: scan {options} [launchcmd [launchargs]] target [targetargs]
    where {options} may include:
 -h    Help: show this brief usage message and exit.
 -v    Verbose: increase verbosity.
 -n    Preview: show command(s) to be launched but don't execute.
 -q    Quiescent: execution with neither summarization nor tracing.
 -s    Summary: enable runtime summarization. [Default]
 -t    Tracing: enable trace collection and analysis.
 -a    Analyze: skip measurement to (re-)analyze an existing trace.
 -e epik      : Experiment archive to generate and/or analyze.
                (overrides default experiment archive)
 -f filtfile  : File specifying measurement filter.
 -l lockfile  : File that blocks start of measurement.
 -m metrics   : Metric specification for measurement.
```


K computer

基本的なコマンド square

```
$ square
SCALASCA 1.4.3: analysis report explorer
usage: square [-v] [-s] [-f filtfile] [-F] <experiment archive | cube file>
  -F          : Force remapping of already existing reports
  -f filtfile : Use specified filter file when doing scoring
  -s          : Skip display and output textual score report
  -v          : Enable verbose mode
```


K computer

自動 Instrumentation Summary解析

どんなとき？

- まず大雑把にアプリケーションの挙動を確認したい

自動 Instrumentation Summary解析(準備)

```
$ cd ~/Scalasca
$ cp –r NPB_test NPB_Summary
$ cd NPB_Summary
$ make clean (不要なものを削除)
$ cd bin
$ rm –f mg*    (不要なものを削除)
$ cd ../MG
```

（エディタでMakefileファイルを編集）

13行目 **${FLINK}**

↓

skin ${FLINK}

16行目 **${FCOMPILE}**

↓

skin ${FCOMPILE}

K computer

自動 Instrumentation Summary解析(instrumentation)

```
$ cd ~/Scalasca/NPB_Summary
$ make suite
$ cd bin
$ ls –l  (作成したファイルを確認)
```

自動 Instrumentation Summary解析 (measurement)

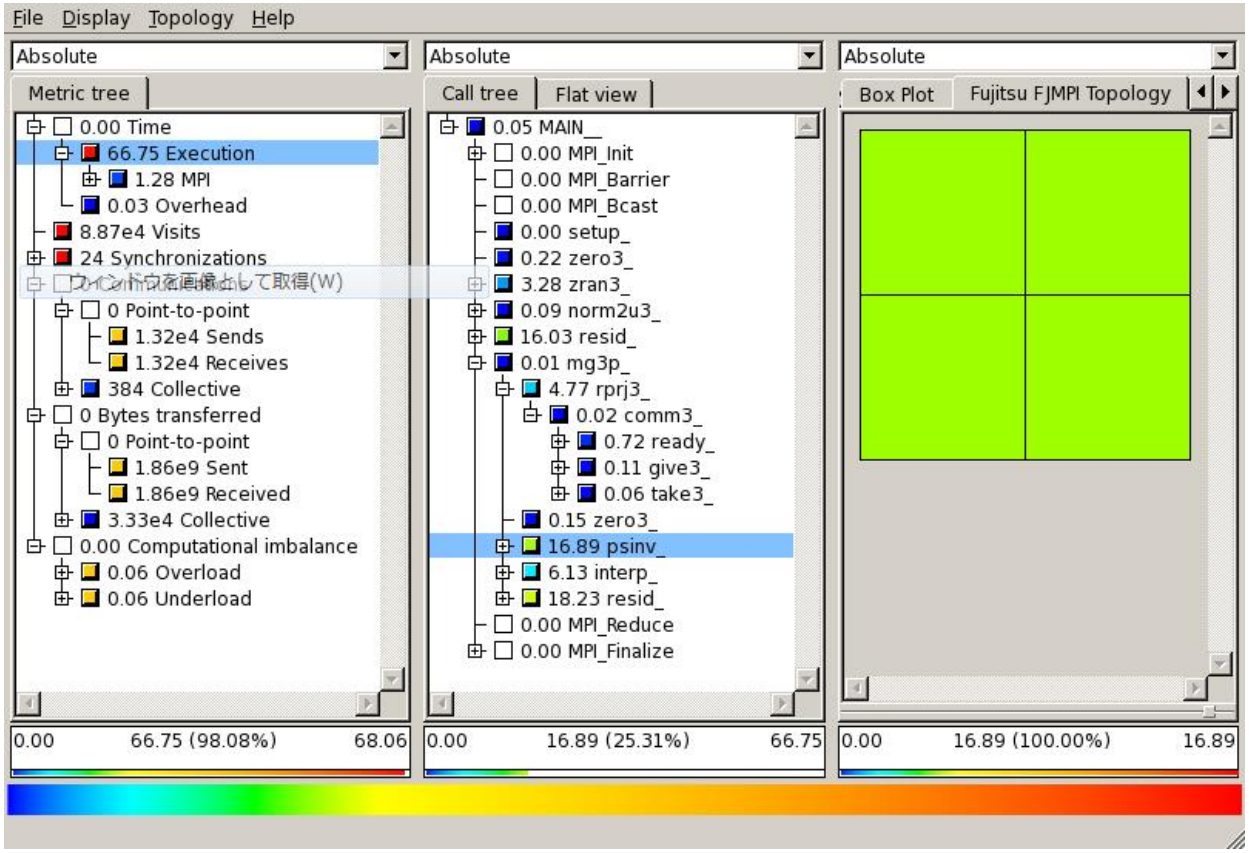
(run.shファイルを編集)
以下の1行をコメントにする
mpiexec -n 4 ./mg.A.4

以下の#を外す
#. /home/ss/scalasca/Env_scalasca
#export SCAN_ANALYZE_OPTS="-i -s“
#scan –e epik_summary mpiexec –n 4 ./mg.A.4

$ pjsub ./run.sh

自動 Instrumentation Summary解析（解析）

```
# cd ~/Scalasca/NPB_Summary/bin
# square ./epik_summary
```

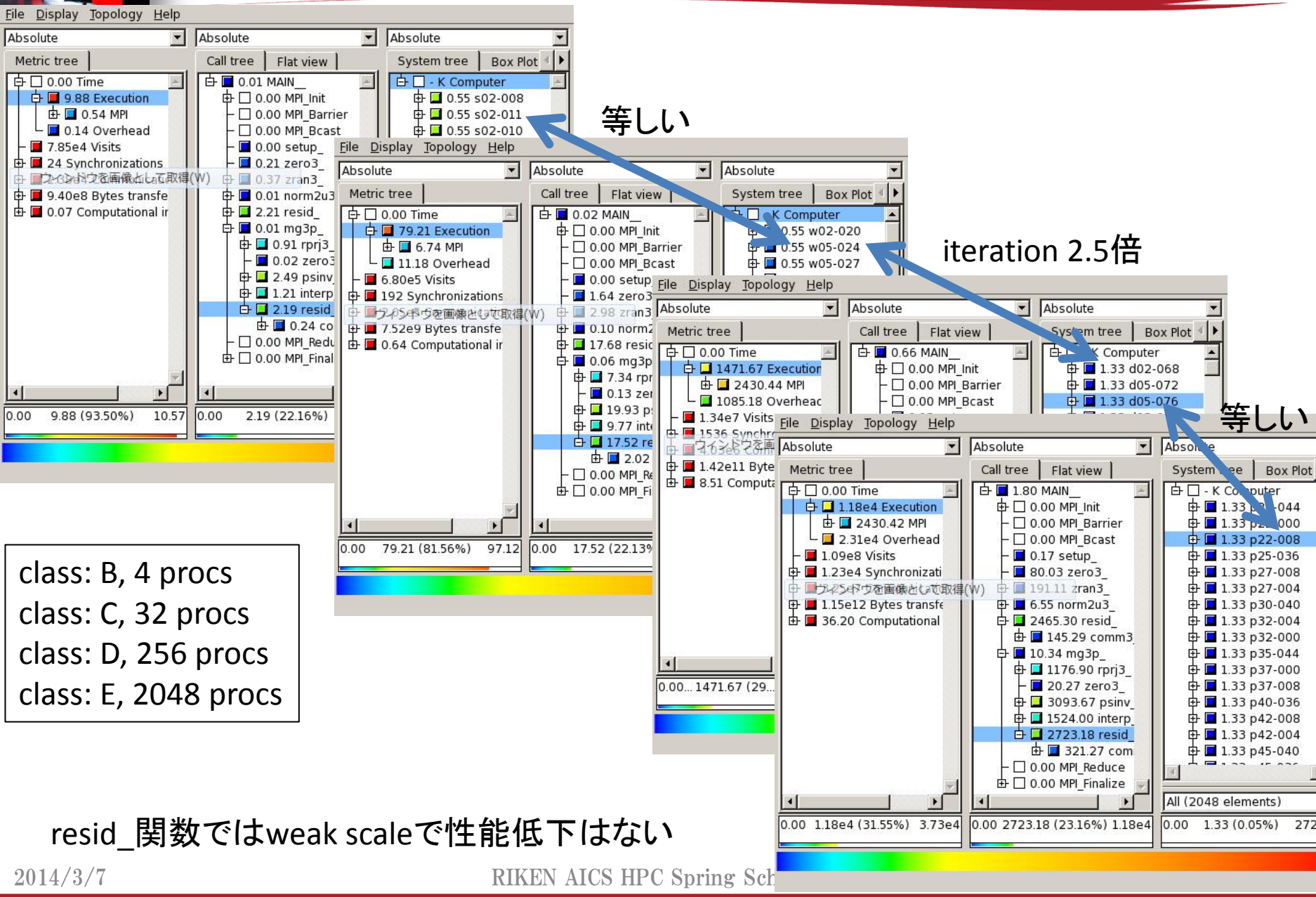


自動 Instrumentation Summaryでの解析

CUBEの操作

- ツリーの展開 (展開後は子の処理コストを含まない)
- 色によるコスト分類
- metric tree
  - 処理時間，通信回数，
  - 通信と演算の割合
  - ヘルプ（オンラインドキュメント）の表示
- call tree
  - 各関数の計算時間，割合
  - ソースコード表示
  - 検索 (ex. MPI_Wait)
- System tree
  - 3次元での表示

Weak scalingの調べ方



class: B, 4 procs
class: C, 32 procs
class: D, 256 procs
class: E, 2048 procs

resid_関数ではweak scaleで性能低下はない

自動 Instrumentation Filterの使い方

（今回のMGは例として適当ではありませんが。。）

どんなとき？

- Summary解析で不要な情報まで取得したくない
- オーバヘッドが大きくなりうまく性能を測れない
- 後で出てくるTrace解析でScalascaの制限にひっかかる
  - イベントトレースの合計は34GBまで
  - 各ランクのトレースバッファサイズは8.5GBまで

Filterの使い方

```
$ cd ~/Scalasca
$ cp –r NPB_Summary NPB_Filter
$ cd NPB_Filter/bin/epik_summary
$ square –s ./
$ cat epik.score
```

例えばresid_関数,ready_関数，take3_関数についてはトレース情報は必要ない場合以下のようなscalasca.filtファイルを作成

```
$ cat scalasca.filt
resid_
ready_
take3_
```

（run.shを編集）

- 以下の行をコメントに

  scan mpiexec -n 4 ./mg.A.4

- 以下の#を外す

  #scan –f scalasca.filt –e epik_filter mpiexec -n 4 ./mg.A.4

自動 Instrumentation Trace解析

どんなとき？

- フィルタで要らない情報を取らないようにした後，通信待ちオペレーションの発生箇所を特定する
- 通信待ちオペレーションによるボトルネックを見つける

自動 Instrumentation Trace解析(準備, instrumentation)

```
$ cd ~/Scalasca
$ cp –r NPB_Summary NPB_Auto
$ cd NPB_Auto
$ make clean
$ cd bin
$ rm –fr epik_*
```
（実行ファイルはSummary解析と同じものを使います）

（run.shを編集）

- 以下の行をコメントに

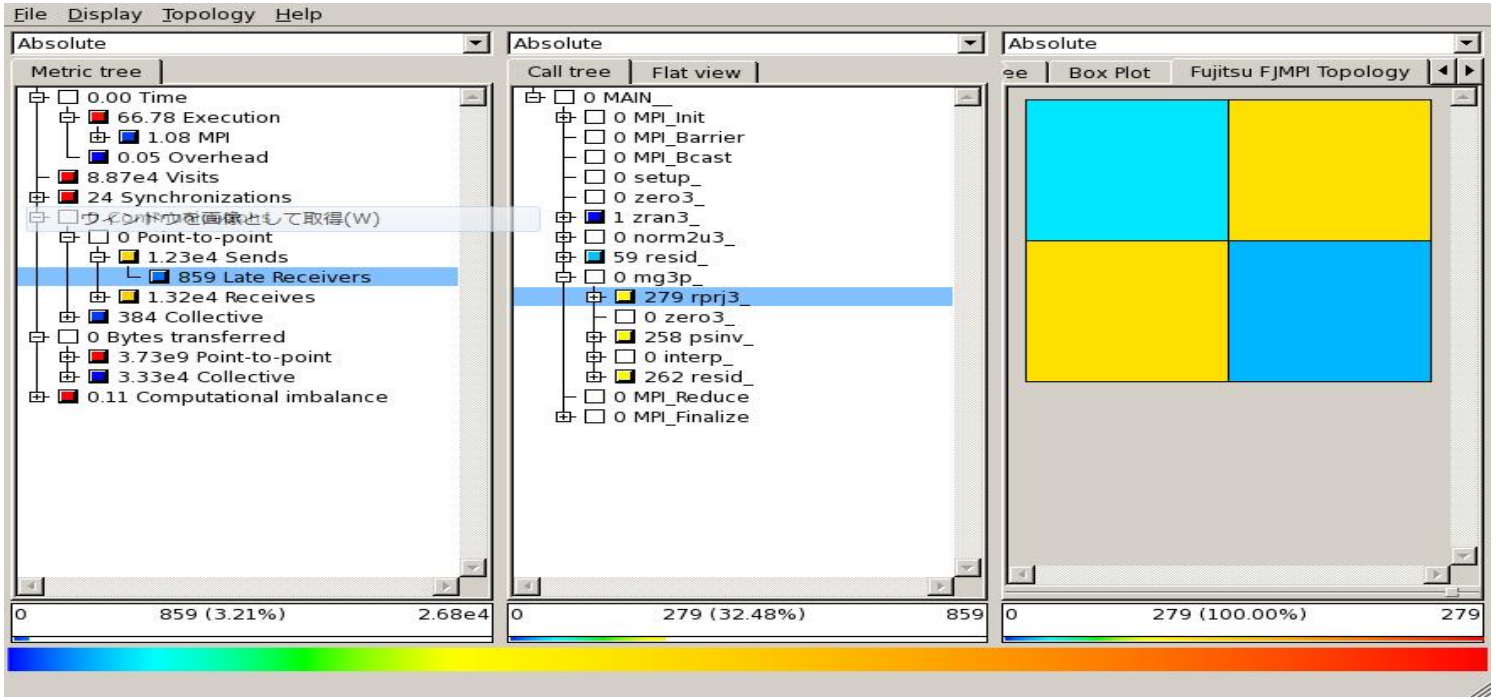
  ```
  scan –f scalasca.filt –e epik_filter mpiexec -n 4 ./mg.A.4
  ```

- 以下の#を外す

  ```
  #scan -t –e epik_auto mpiexec -n 4 ./mg.A.4
  ```

自動 Instrumentation Trace解析（measurement，表示）

```
$ cd ~/Scalasca/NPB_Auto/bin
（MGはフィルタを必要としないのでつけません）
$ pjsub ./run.sh
$ square ./epik_auto
```



K computer

自動 Instrumentation Trace解析(表示&解析)

CUBEの操作

- metric tree
  - Late Senderなどの待ち時間を確認
  - ヘルプ(オンラインドキュメント)の表示

自動 Instrumentation Trace解析(表示&解析)

(例として以下を確認)

- ホットスポットをみつける
- 各ノードの演算時間
- 通信と演算の比
- Late Sender, Early Recieverなど起こっている部分の確認

自動 Instrumentation PAPIを使った解析

どんなとき？

以下のようなハードウェアカウンタの値を見たいとき
・キャッシュミス
・TLBミス
・浮動小数点演算回数

PAPIを使った解析 (instrumentation, measurement)

```
$ cd ~/Scalasca
$ cp –r NPB_Summary NPB_Papi
$ cd NPB_Papi/bin
$ rm –fr epik*
```

(run.shファイル編集)
以下のコメントを外す
#/opt/FJSVXosPA/bin/xospastop (課金PA機能を止める)
#scan –t –m L1_MISS:…

以下をコメントに
scan -t –e epik_auto mpiexec -n 4 ./mg.A.4

```
$ pjsub ./run.sh
$ square ./epik_papi
```

PAPIを使った解析

Metric treeに
・L1_MISS
・L1_I_MISS
など-mオプションで指定したものが表示

例として以下を確認
L1_MISSが多い関数→resid_, psinv_, ready_, give3_, take3_
L1_I_MISSが多い関数→rprj3_, psinv_, interp_, resid_
FLOATING_POINTが多い関数→zran3_, resid_, rprj3_, psinv_

scanの-mオプションに指定可能なものは以下のファイルを参照
/home/scalasca/doc/papi_avail.txt

ただし，ハードウェアカウンタが8個あり，バッティングしないものなら同時に調べられる。
（SPARC64TM VIIIfx Extensions p304,305を参照）

半自動 POMP User Instrumentation

どんなとき？

- 自動 Instrumentation で失敗する場合
- プログラムの一部の詳細を分析したい場合

半自動 POMP User Instrumentation

```
$ cd ~/Scalasca
$ cp –r NPB_Summary NPB_Pomp
$ cd ~/Scalasca/NPB_Pomp/MG
（mg.f を半手動でのインストルメント）
```

rpj3サブルーチンのループ部分の詳細を分析したいときの例
(91行目あたり)
!POMP$ INST INIT ! mainプログラムに1回だけ
…
(824行目あたり)
!POMP$ INST BEGIN(rprj3_loop)
…
(857行目あたり)
!POMP$ INST END(rprj3_loop)

実行は同様
$ cd ~/Scalasca/NPB_Pomp

手動 EPIK User Instrumentation

どんなとき？

- 自動や半自動で Instrumentation に失敗する場合
- プログラムの一部の詳細を分析したい場合

手動 EPIK User Instrumentation 準備

```
$ cd ~/Scalasca
$ cp –r NPB_Summary NPB_User
$ cd NPB_User/bin
$ rm –fr (runmg_*.sh以外削除)
$ cd ..
$ make clean
$ cd MG
$ mv mg.f mg.F   (CPPを通すため)
```

（Makefileを編集）
“mg.f” を “mg.F” に変更
“skin” を “skin –user” に変更
自動instrumentationが必要なければ .o ファイル作成時に -comp=noneをつける

手動 EPIK User Instrumentation Instrumentation

mg.F ファイルの編集

54行目あたりに以下を追加

```
#include "/home/ss/Scalasca/include/fe/epik_user.inc“
```

(Makefileに ”-I” オプションでディレクトリを指定しても構いません)

```
 subroutine mg3P(u,v,r,a,c,n1,n2,n3,k)
      …
      EPIK_FUNC_REG(“func_mg3P”)
      EPIK_USER_REG(r_name1,”iter_loop”)
      EPIK_USER_REG(r_name2,”iter_loop_4call”)
      …
```


K computer

手動 EPIK User Instrumentation Instrumentation

mg.F ファイルの編集つづき

```
...
EPIK_FUNC_START()
EPIK_USER_START(r_name1)
do  k = lb+1, lt-1
   j = k-1
   EPIK_USER_START(r_name2)
   call zero3(u(ir(k)),m1(k),m2(k),m3(k))
   call interp(u(ir(j)),m1(j),m2(j),m3(j),u(ir(k)),m1(k),m2(k),m3(k),k)
   call resid(u(ir(k)),r(ir(k)),r(ir(k)),m1(k),m2(k),m3(k),a,k)
   call psinv(r(ir(k)),u(ir(k)),m1(k),m2(k),m3(k),c,k)
   EPIK_USER_END(r_name2)
enddo
EPIK_USER_END(r_name1)
...
EPIK_FUNC_END()
return
end
```

EPIK User Instrumentation

C

```
#include " /home/…/epik_user.inc "
…
void foo() {
   …
   EPIK_FUNC_START();
   …
   if (...) {
      EPIK_FUNC_END();
      return;
   } else {
      EPIK_USER_REG(r_name,"region");
      EPIK_USER_START(r_name);
      …
      EPIK_USER_END(r_name);
   }
   …
   EPIK_FUNC_END();
   return;
}
```

Fortran

```
#include “/home/…/epik_user.inc"
…
subroutine bar()
   EPIK_FUNC_REG("bar")
   EPIK_USER_REG(r_name, "region")
   …
   EPIK_FUNC_START()
   if (...) then
      EPIK_FUNC_END()
      return
   else
      EPIK_USER_START(r_name)
      …
      EPIK_USER_END(r_name)
   endif
   …
   EPIK_FUNC_END()
   return
end subroutine bar
```


K computer