PMlib 講習会

理化学研究所 計算科学研究機構
可視化技術研究グループ
2015年1月16日

本日使用する資料の入手方法

- 各自のPCへWebブラウザからアクセス・ダウンロード

- 本日使用する資料
  - スライドおよびハンズオンプログラムは下記から
  - https://github.com/mikami3heart/PMlib-tutorials

- PMlibパッケージ
  - パッケージファイル一式の tar.gz ファイル
  - http://avr-aics-riken.github.io/PMlib/

講習会の内容

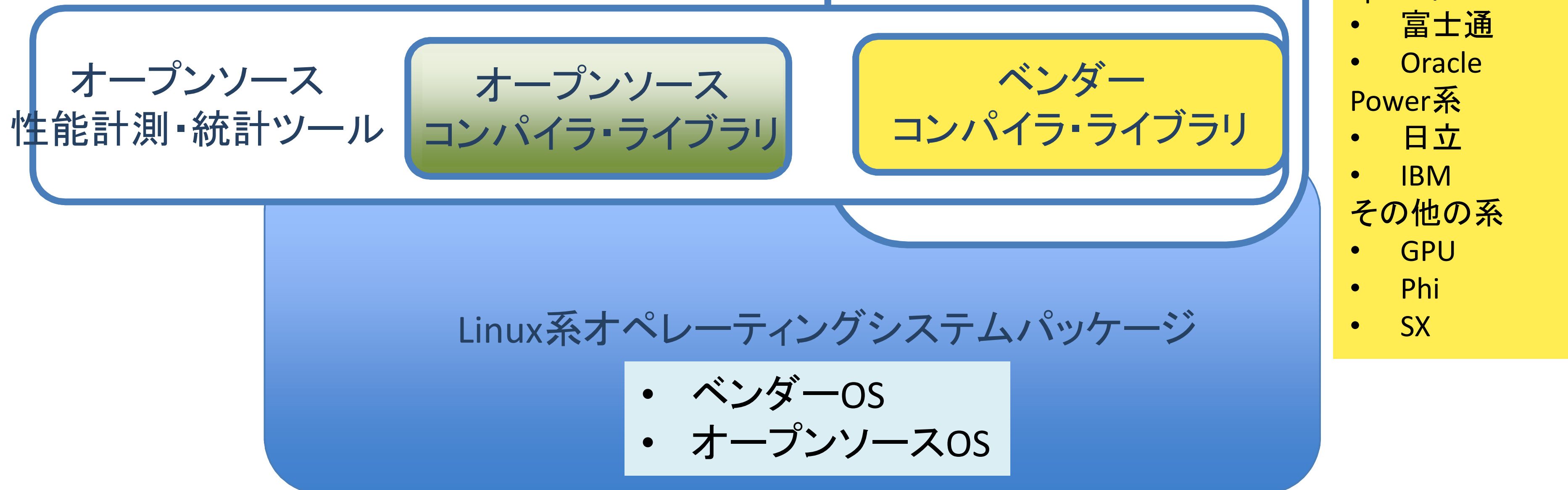
- PMlib概要
  - PMlibとは
  - PMlibの利用方法
  - PMlibの機能説明
- PMlibのインストールとテスト
  - PMlibの入手方法
  - テストシステムへのログイン
  - PMlibのインストール
  - 動作確認プログラムの実行
- ハンズオン
  - ハンズオンプログラムについて
  - プログラムの実行
  - プログラムへのPMlibのくみこみ
  - プログラムのPMlib統計情報の解釈と検討

PMlibとは

- アプリケーション計算性能モニター用のクラスライブラリ
- ユーザーライブラリとしてもシステムライブラリとしても利用可
- オープンソースソフトウエア（理研 AICSが開発・提供）
- アプリケーション中に計測区間を指定し、実行終了時に区間の統計情報を出力する
- ソースプログラム中でPMlibライブラリを呼び出して利用する
- アプリケーション性能改善用の一時的な利用だけでなく、プロダクションランに常用して性能モデリングの支援に用いられる事を期待
- APIはC++に対応

性能統計ツールの位置づけ

- オープンソース性能統計ツール一般
  - Gprof: 簡易機能、コンパイラに制約
  - Scalasca: 高機能、Score-P共通インフラ
  - TAU: 高機能
  - PAPI : HWPCへのアクセス
  - など多数、、、プラス
  - PMlib



ベンダーツール

X86系
- Intel
- PGI
- Cray

Sparc系
- 富士通
- Oracle

Power系
- 日立
- IBM

その他の系
- GPU
- Phi
- SX

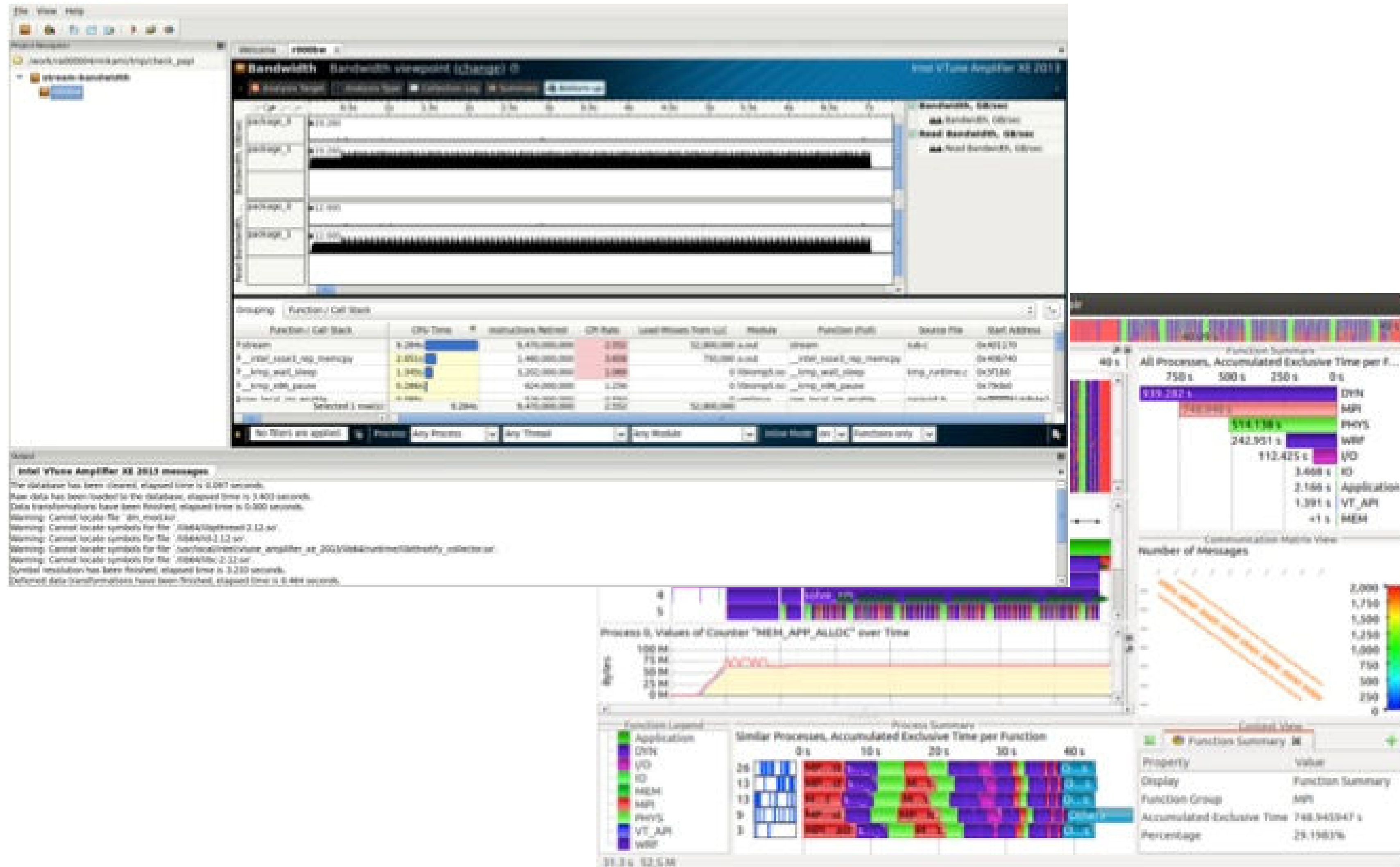
各ツールの位置づけ

- ベンダー性能計測・統計ツール
  - ○豊富な機能、高度なインタフェイス、システムに統合化された安心感、詳しいドキュメント、ベンダーによるサポート
  - △習熟に相当期間が必要、システム機種毎にツールが決まってしまう、それなりの価格
- オープンソース性能統計ツール
  - ○各ツール毎に高機能、無料
  - △ユーザーインタフェイスが個性的、インストールの手間・利用方法の習熟がそれなりに大変→周囲にツールをよく知っている人がいないとハードルは高い
- 高機能GUIツールのハードル

- PMlib
  - 機能・出力情報をテキストに絞ったコンパクトなツール

高機能GUIベースツールのハードル

- 見た目の豪華さ ∝ 利用に必要な習熟期間の長さ



PMlibの特徴

- PMlibの特徴
  - 機能を絞ったテキストベースのコンパクトなツール
  - インストール・利用が容易
    - プラットフォーム依存性が低い
  - 利用のオーバーヘッドが少ない軽量ツール
  - 性能情報の採取方法を選択可能
    - ユーザ自身の明示的申告か、HWPCによる自動取得
  - 性能計測結果は指定区間毎に出力
    - 出力タイプ1:全プロセスの平均した基本情報
    - 出力タイプ2:MPIランク（プロセス）毎の情報
    - 出力タイプ3:ハードウエアイベントグループの情報

PMlibが対応する並列プログラムモデル

- シリアルプログラム
- OpenMP (SMPスレッド) 並列プログラム
  - 測定区間内にOpenMPループを含む場合に相当
  - ただしスレッド自身からのPMlibよびだしには未対応
- MPI並列プログラム
- MPIとOpenMPの組み合わせ並列プログラム
- APIはC++に対応

PMlibの利用方法

- PMlibライブラリのインストール
- アプリケーション へPMlib呼び出しを追加
- アプリケーションの実行
- PMlib出力情報の評価

PMlib利用プログラム例

- 元のソース

```
int main (int argc, char *argv[])
{
    subkerel(); //演算を行う関数

    return 0;
}
```

PMlib組み込み後のソース

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main (int argc, char *argv[])
{
    PM.initialize();
    PM.setParallelMode("Serial" , 1, 1);
    PM.setProperties(" my_check_1", 1);
    PM.start(" my_check_1");
    subkerel();
    PM.stop (" my_check_1", 0.0, 1);
    PM.gather();
    PM.print(stdout, " London", "Mr.Bean");
    PM.printDetail(stdout);
    return 0;
}
```

ヘッダー部

初期設定

測定区間

結果を出力

PMlib関数の仕様詳細

- 以降のスライドは「本日使用する資料のページ」からダウンロードしたファイルに含まれる
- 下のコマンドで復元したindex.htmlファイルを各自のPC上のWebブラウザで表示すると見やすい

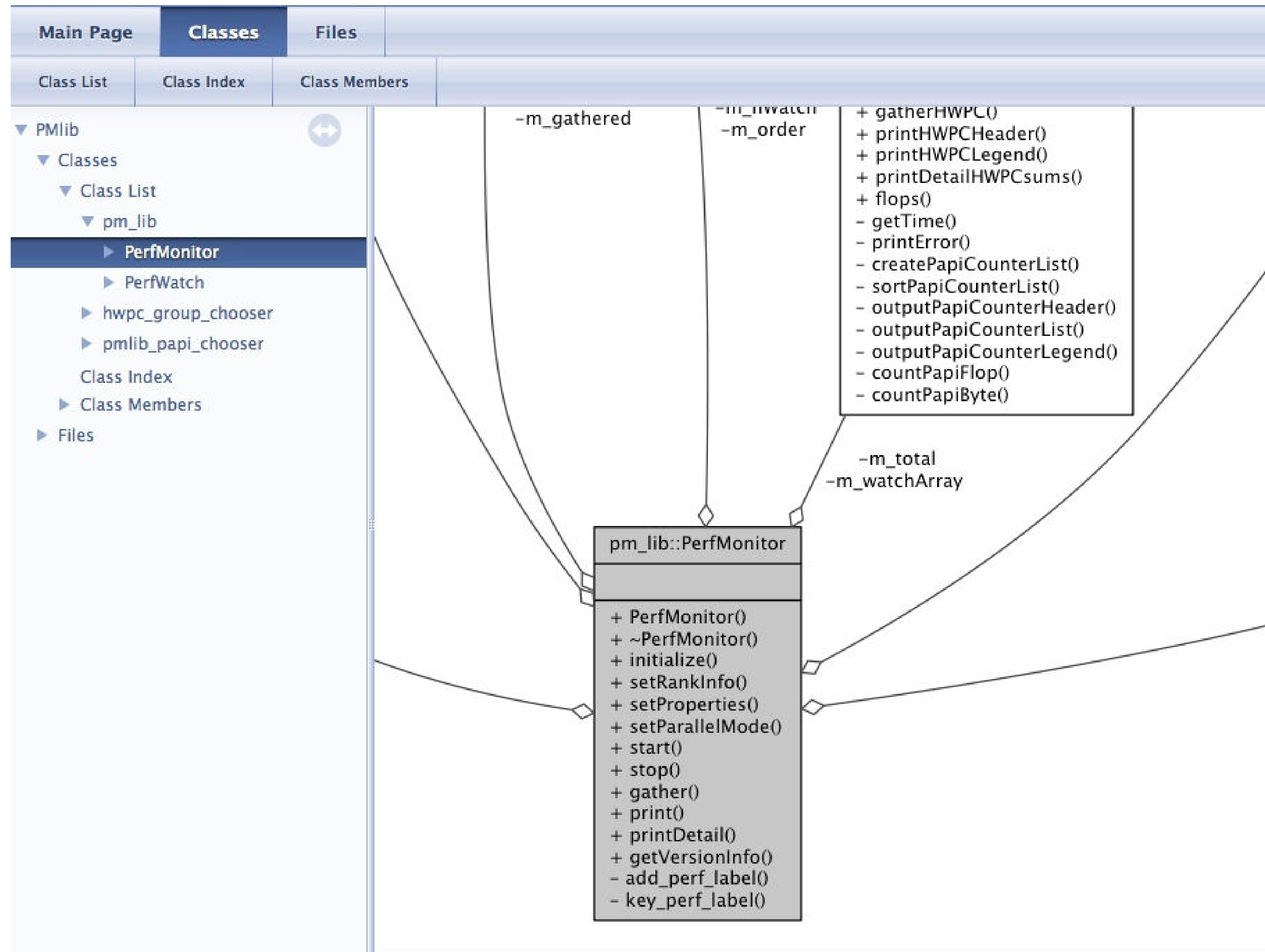
```
$ tar –zxf PMlib-doxygen.tar.gz
$ cd PMlib-doxygen-html
$ file index.html
```

PMlib関数一覧

関数名	機能	呼び出し位置・回数	引数の仕様
initialize()	PMlib全体の初期化	冒頭・一回	(1)測定区間数
setParallelMode()	並列処理のタイプ指定	冒頭・一回	(1)並列モード、(2)スレッド数、(3)プロセス数
setRankInfo()	MPIランク番号の設定	冒頭・一回	(1)ランク番号
setProperties()	測定区間のラベル化	任意・一回	(1)ラベル、(2)測定対象タイプ、(3)排他指定
start()	測定の開始	任意・任意 （startとstopでペア）	(1)ラベル
stop()	測定の停止	任意・任意 （startとstopでペア）	(1)ラベル、(2)計算量、(3)計算のタスク数
gather()	測定結果情報をマスタープロセスに集約	測定終了後・一回	なし
print()	測定区間毎の基本統計結果表示	測定終了後・一回	(1)出力ファイルポインタ、(2)ホスト名、(3)実施者名
printDetail()	並列ランク・性能情報毎の詳細表示	測定終了後・一回	(1)出力ファイルポインタ

これら関数の仕様や引数詳細説明は Doxygenで生成・表示

各関数の仕様（Webブラウザで表示）



各関数の仕様　initialize()

void pm_lib::PerfMonitor::initialize (int init_nWatch = 100)　inline

初期化.

測定区間数分の測定時計を準備. 全計算時間用測定時計をスタート.

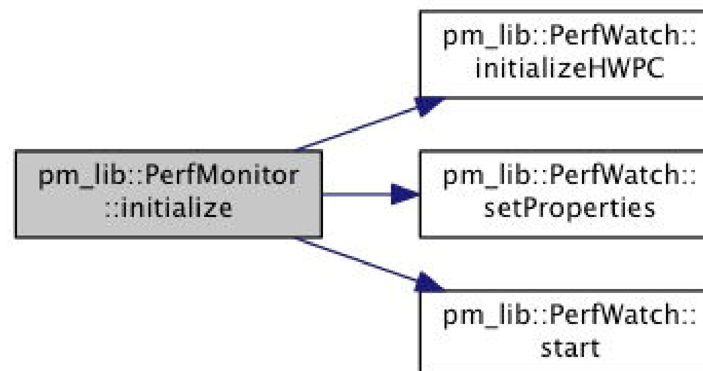
Parameters

[in]　(引数はオプション)　init_nWatch 最初に確保する測定区間数

Note

測定区間数 m_nWatch は動的に増えていく事もある 最初にinit_nWatch区間分を確保し、不足したらさらにinit_nWatch追加する

Here is the call graph for this function:



各関数の仕様　setParallelMode()

```
void pm_lib::PerfMonitor::setParallelMode ( const std::string &  p_mode,
                                            const int            n_thread,
                                            const int            n_proc
                                          )
```

inline

並列モードを設定

Parameters

[in] p_mode　並列モード "Serial","OpenMP","FlatMPI","Hybrid"

[in] n_thread スレッド数

[in] n_proc　MPIプロセス数

各関数の仕様　setProperties()

```
void pm_lib::PerfMonitor::setProperties ( const std::string &  label,
                                          Type                 type,
                                          bool                 exclusive = true
                                        )
```

inlin

測定時計にプロパティを設定.

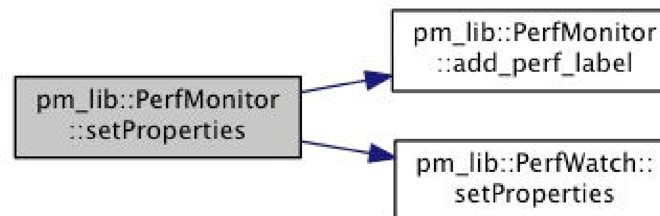
Parameters

[in]	**label**	ラベル文字列
[in]	**type**	測定対象タイプ(COMM:通信, CALC:計算, AUTO:自動決定)
[in]	**exclusive**	排他測定フラグ(ディフォルトtrue)

Note

測定区間を識別するためにlabelを用いる。 各labelに対応したキー番号 key は各ラベル毎に内部で自動生成する 最初に確保した区間数init_nWatchが不足したらさらにinit_nWatch区間追加する

Here is the call graph for this function:



各関数の仕様　start()/stop()

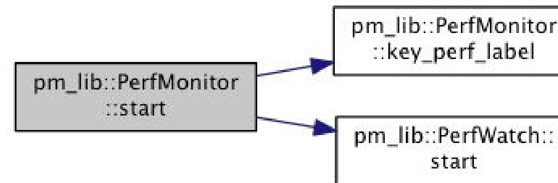
void pm_lib::PerfMonitor::start (const std::string & label) inline

測定スタート.

Parameters

[in] label ラベル rev.2.2 からkey キー番号ではなくラベルを使用

Here is the call graph for this function:



void pm_lib::PerfMonitor::stop (const std::string & label,
double flopPerTask = 0.0,
unsigned iterationCount = 1
) inline

測定ストップ.

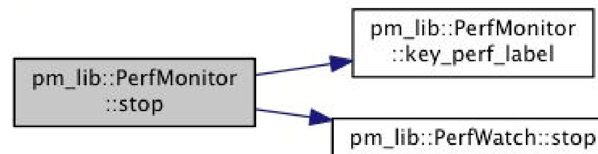
Parameters

[in] label　ラベル rev.2.2 からkey キー番号ではなくラベルを使用

[in] flopPerTask　「タスク」あたりの計算量/通信量(バイト) (ディフォルト0)

[in] iterationCount 実行「タスク」数 (ディフォルト1)

Here is the call graph for this function:



各関数の仕様　gather()

void pm_lib::PerfMonitor::gather (void)

全プロセスの全測定結果情報をマスタープロセス(0)に集約.

全計算時間用測定時計をストップ.

各関数の仕様　print()/printDetail()

```
void pm_lib::PerfMonitor::print ( FILE *             fp,
                                  const std::string  hostname,
                                  const std::string  operatorname
                                )
```

測定結果の基本統計情報を出力.

排他測定区間のみ

Parameters

[in]	fp	出力ファイルポインタ
[in]	hostname	ホスト名
[in]	operatorname	作業者名

Note

ノード0以外は, 呼び出されてもなにもしない

```
void pm_lib::PerfMonitor::printDetail ( FILE * fp )
```

詳細な測定結果を出力.

ノード毎に非排他測定区間も出力

Parameters

[in] fp 出力ファイルポインタ

Note

ノード0以外は, 呼び出されてもなにもしない

各関数の仕様　setRankInfo()

void pm_lib::PerfMonitor::setRankInfo (const int myID)

ランク番号の通知

Parameters

[in] **myID** 通常はMPIランクIDを指定する

Pmlibの動作確認がとれているシステム

- 京/FX10
  - ログインノードでのクロスコンパイル環境
  - 計算ノードでのネイティブコンパイル環境
  - 富士通コンパイラ＋MPI
- Intel Xeon E5 クラスタ
  - Intelコンパイラ＋IntelMPI
  - GNUコンパイラ+OpenMPI/gnu
  - PGIコンパイラ+OpenMPI/pgi
- 必要なソフトウエア環境
  - C, C++ compiler
  - HWPC/PAPIを組み込む場合はLinux kernel 2.6.32+

PMlibの入手方法

- PMlibパッケージの入手方法
  - 下記公開リポジトリからDownload
  - http://avr-aics-riken.github.io/PMlib/

- PMlibに関するドキュメント
  - パッケージに含まれるdoc/ディレクトリ以下にある
    - How_to_use_PMlib.pdf　:クラスライブラリの説明書
    - PMlib_getting_started.pdf :本資料

PMlib計算性能モニター機能

- 指定した測定区間毎に性能統計情報を蓄積・出力
- 各測定区間は小数のプロパティを持つ
  - ラベル：任意の文字列（統計情報出力時のラベル ）
  - 測定対象タイプ：「計算時間」、「通信時間」、「自動決定」
  - 排他測定フラグ：「排他測定」または「非排他測定」
- 性能統計の種類と算出方法を選択
  - 計算量をユーザが明示的に申告する場合
    - 測定区間の「量」を計算式で引数として与える
    - 測定対象タイプにより、「量」を浮動小数点演算量あるいはデータ移動量として評価

性能統計：明示的な自己申告

- 計算量をユーザが明示的に申告する場合
  - ソースプログラムに記述されたの計算式に忠実な実行性能を測定可能
  - 演算の種類に応じて四則演算の「重さ」を指定することも可能
    - 計算式を実行するのに必要な演算数は、計算の種類、実行するシステム毎で異なる
    - 例えばFX10のPA情報から評価すると．．．
      - +, −, x　:　1 flop
      - ÷　:　8 flops（単精度），13 flops（倍精度）
      - abs()　:　1 flops
    - 加算・乗算以外の複雑な演算も一回の計算として計上可能
  - 実行時の各セクションのタイミングと演算数を積算して記録
    - タイミング測定区間はラベル管理で，コーディング時に指定

性能統計:計算量の自動算出

- 実行するシステムのCPUハードウエア性能カウンター(HWPC)を内部に持ち、そのイベント情報を測定可能な場合に採取して出力
- HWPCのイベントリスト別表
- PMlib用にイベントの種類毎グループを定義
  - FLOPS
  - VECTOR
  - BANDWIDTH
  - CACHE
  - CYCLE
- プログラム実行時に環境変数で動的に選択する
  - マスタープロセス(MPI rank 0)の測定値を代表値として出力
  - もしOpenMPスレッド並列処理の場合はマスタープロセスが発生するスレッド群の合計値を出力

出力する情報

- １、基本プロファイル
  - 全プロセスの平均情報
  - プログラム終了時に各MPIプロセス（ランク）の情報をマスターランクに集計。統計処理して出力
- ２、詳細プロファイル(1：MPIプロセス毎）
  - MPIの各プロセス毎の情報を出力
- ３、詳細プロファイル(2：HWPCイベント統計 ）
  - 計測するHWPCイベントグループを環境変数で指定
  - プロセスがOpenMPスレッドを発生した場合各プロセスのにスレッド測定値を内部で合計する。マスタープロセス（MPI rank 0)の値を出力

基本プロファイル例

```
----------------------------------------------------
    Report of Timing Statistics PMlib version 2.1.2
    Operator  : Kenji_Ono
    Host name : vsp22
    Date      : 2014/05/26 : 03:49:43

    Parallel Mode                   :   Hybrid (4 processes x 8 threads)

    Total execution time            = 4.429648e+00 [sec]
    Total time of measured sections = 3.695136e+00 [sec]

    Statistics per MPI process [Node Average]
    Label                    |     call      |              accumulated time                     |         flop | messages[Bytes]
                             |               |   avr[sec]     avr[%]      sdv[sec]  avr/call[sec] |   avr          sdv           speed
    -------------------------+---------------+---------------------------------------------------+-------------------------------------------
    Poisson_SOR2_SMA        :         26200   1.071254e+00   28.99   6.4536e-03  4.088757e-05    2.318e+10   0.000e+00    21.64 Gflops
    File_Output             :            11   8.003855e-01   21.66   1.3644e-02  7.276232e-02    1.311e+07   0.000e+00    16.38 Mflops
    Poisson_Src_Norm        :         13300   7.372693e-01   19.95   2.6937e-03  5.543378e-05    4.112e+10   0.000e+00    55.77 Gflops
    Sync_Pressure           :         26200   5.582871e-01   15.11   7.6391e-03  2.130867e-05    1.717e+09   0.000e+00     2.86 GB/sec
    A_R_Poisson_Src         :         13300   2.086478e-01    5.65   2.1901e-02  1.568781e-05    8.512e+05   0.000e+00     3.89 MB/sec
    Pseudo_Velocity         :           100   7.437694e-02    2.01   4.6558e-04  7.437694e-04    5.230e+09   0.000e+00    70.31 Gflops
    Projection_Velocity     :           262   7.227474e-02    1.96   6.3243e-04  2.758578e-04    1.820e+09   0.000e+00    25.18 Gflops
    Poisson_BC              :         26200   5.180532e-02    1.40   8.1025e-04  1.977302e-06    0.000e+00   0.000e+00     0.00 Mflops
    Poisson_Setup_for_Itr   :         13100   2.677810e-02    0.72   2.0531e-04  2.044130e-06    0.000e+00   0.000e+00     0.00 Mflops
    Sync_Velocity           :           100   1.325995e-02    0.36   3.9118e-04  1.325995e-04    1.573e+08   0.000e+00    11.05 GB/sec
    Sync_Pseudo_Velocity    :           100   1.039678e-02    0.28   7.3619e-04  1.039678e-04    3.932e+07   0.000e+00     3.52 GB/sec
    Variation_Space         :           100   9.503841e-03    0.26   5.3508e-05  9.503841e-05    4.260e+08   0.000e+00    44.82 Gflops
    Force_Calculation       :           100   6.575465e-03    0.18   7.8837e-03  6.575465e-05    0.000e+00   0.000e+00     0.00 Mflops
    Copy_Array              :           200   6.332040e-03    0.17   8.1976e-05  3.166020e-05    0.000e+00   0.000e+00     0.00 Mflops
    Projection_Velocity_BC  :           262   6.013393e-03    0.16   8.4818e-05  2.295188e-05    0.000e+00   0.000e+00     0.00 Mflops
    Divergence_of_Pvec      :           100   5.138874e-03    0.14   5.1616e-05  5.138874e-05    2.425e+08   0.000e+00    47.19 Gflops
    A_R_Poisson_Norm        :           262   5.013704e-03    0.14   6.8061e-04  1.913628e-05    1.677e+04   0.000e+00     3.19 MB/sec
    Allocate_Arrays         :             4   5.009890e-03    0.14   2.1477e-04  1.252472e-03    0.000e+00   0.000e+00     0.00 Mflops
   …
-----------------------+---------------+---------------------------------------------------+-------------------------------------------
    Total                |                3.695136e+00                                         7.225e+10                19.55 Gflops
    Performance          |                                                                                              78.21 Gflops
```

詳細プロファイル(1)

```
  Elapsed time variation over MPI ranks

*Initialization_Section
MPI_rank        call        accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0     :           1  1.623359e-01    4.39  6.041527e-04  1.623359e-01   0.000e+00     0.00 Mflops
#1     :           1  1.629400e-01    4.41  0.000000e+00  1.629400e-01   0.000e+00     0.00 Mflops
#2     :           1  1.543641e-01    4.18  8.575916e-03  1.543641e-01   0.000e+00     0.00 Mflops
#3     :           1  1.500421e-01    4.06  1.289797e-02  1.500421e-01   0.000e+00     0.00 Mflops

Allocate_Arrays
MPI_rank        call        accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0     :           4  5.146027e-03    0.14  9.012222e-05  1.286507e-03   0.000e+00     0.00 Mflops
#1     :           4  5.236149e-03    0.14  0.000000e+00  1.309037e-03   0.000e+00     0.00 Mflops
#2     :           4  4.867315e-03    0.13  3.688335e-04  1.216829e-03   0.000e+00     0.00 Mflops
#3     :           4  4.790068e-03    0.13  4.460812e-04  1.197517e-03   0.000e+00     0.00 Mflops

*Voxel_Prep_Section
MPI_rank        call        accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0     :           1  6.849098e-02    1.85  0.000000e+00  6.849098e-02   0.000e+00     0.00 Mflops
#1     :           1  6.354094e-02    1.72  4.950047e-03  6.354094e-02   0.000e+00     0.00 Mflops
#2     :           1  6.057596e-02    1.64  7.915020e-03  6.057596e-02   0.000e+00     0.00 Mflops
#3     :           1  2.413917e-02    0.65  4.435182e-02  2.413917e-02   0.000e+00     0.00 Mflops

Restart_Process
MPI_rank        call        accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0     :           1  4.053116e-06    0.00  9.536743e-07  4.053116e-06   0.000e+00     0.00 Mflops
#1     :           1  5.006790e-06    0.00  0.000000e+00  5.006790e-06   0.000e+00     0.00 Mflops
#2     :           1  3.099442e-06    0.00  1.907349e-06  3.099442e-06   0.000e+00     0.00 Mflops
#3     :           1  4.053116e-06    0.00  9.536743e-07  4.053116e-06   0.000e+00     0.00 Mflops
```

詳細プロファイル(2)

```
PMlib detected the CPU architecture:
    The available Hardware Performance Counter (HWPC) events depend on this CPU architecture.
        GenuineIntel
        Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
    HWPC event values of the master rank, sum of threads. count unit in Giga (x 10e9)
    -                        :     LD_INS     SR_INS   :HIT_LFB    :L1_HIT    :L2_HIT    :L3_HIT    OFFCORE  [Mem GB/s]
    Initialization_Section   :      0.702      0.104      0.000      0.702      0.000      0.000      0.000      0.069
    Allocate_Arrays          :      0.125      0.023      0.000      0.125      0.000      0.000      0.000      2.371
    Voxel_Prep_Section       :      0.126      0.021      0.000      0.126      0.000      0.000      0.000      0.023
    Restart_Process          :      0.000      0.000      0.000      0.000      0.000      0.000      0.000     14.726
    Time_Step_Loop_Section   :      0.005      0.001      0.000      0.005      0.000      0.000      0.000      0.001
    Search_Vmax              :      0.027      0.003      0.001      0.026      0.000      0.000      0.002     64.719
    A_R_Vmax                 :      0.024      0.005      0.000      0.024      0.000      0.000      0.000      3.541
    Copy_Array               :      0.074      0.042      0.022      0.051      0.000      0.000      0.004     36.320
    assign_Const_to_Array    :      0.010      0.005      0.000      0.009      0.000      0.000      0.000      2.816
    Flow_Section             :      0.005      0.001      0.000      0.005      0.000      0.000      0.000      0.001
    NS__F_Step_Section       :      0.004      0.001      0.000      0.004      0.000      0.000      0.000      0.035
    NS__F_Step_Sct_1         :      0.005      0.001      0.000      0.005      0.000      0.000      0.000      0.571
    NS__F_Step_Sct_2         :      0.005      0.001      0.000      0.005      0.000      0.000      0.000      0.047
    Pseudo_Velocity          :      0.721      0.363      0.002      0.705      0.000      0.000      0.010      8.086
```