PMlib 講習会資料

理化学研究所 計算科学研究機構
可視化技術研究チーム

2015年8月26日

2015年10月8日一部更新

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

- はじめに
  - ゲスト無線LANの利用について（別資料）
  - 資料のダウンロード
- PMlib概要説明
  - 性能統計ツールの位置づけ
  - PMlibの機能と特徴
  - PMlibの関数のAPI仕様
- PMlib実習のインストールとテスト
  - PMlibパッケージのダウンロード
  - テストシステムへのログイン・ファイル転送
  - PMlibのインストール
  - exampleプログラムの実行

資料のダウンロード

- 本日使用する資料のダウンロード
  - https://github.com/mikami3heart/PMlib-tutorials

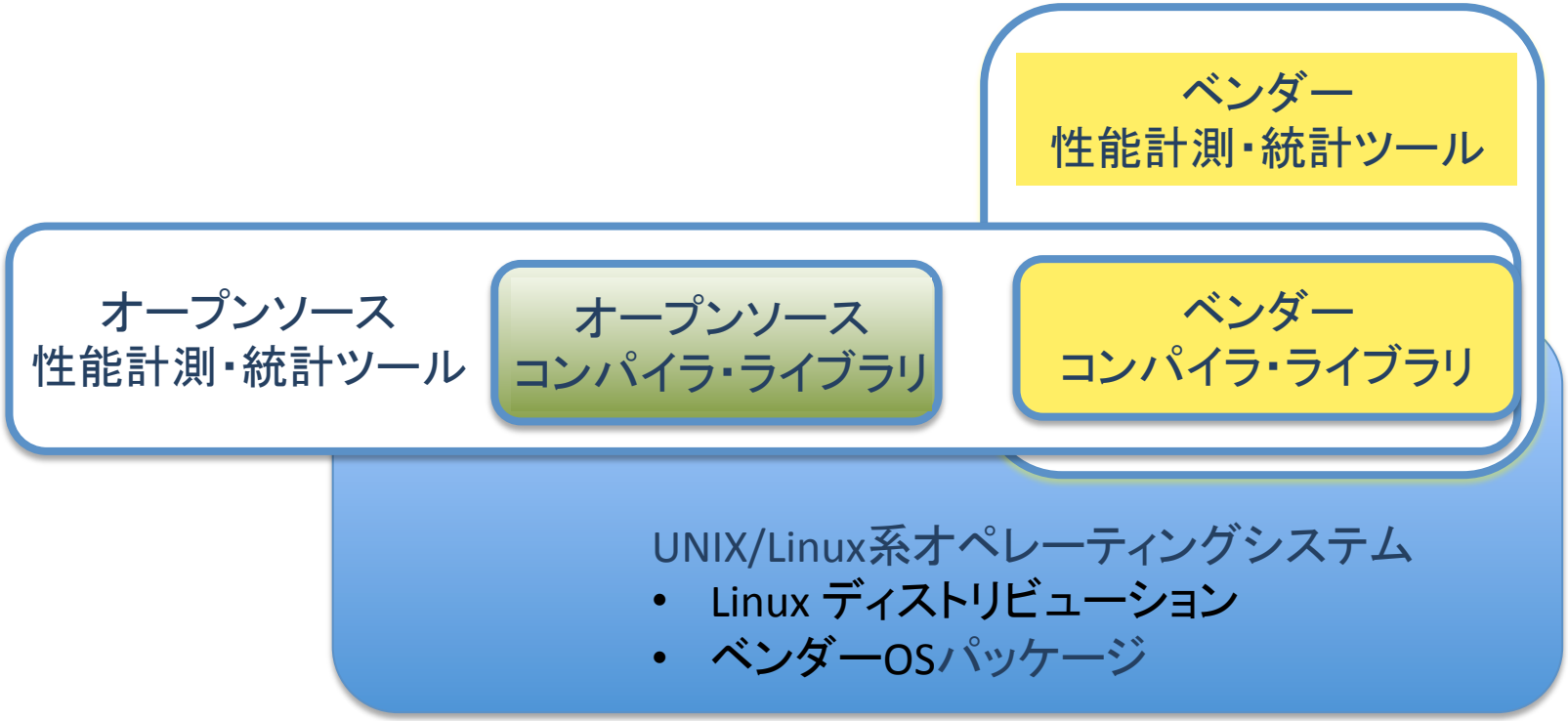
    - PMlib-doxygen.tar.gz
    - Tutorial-slide1-overview.pdf
    - Tutorial-slide2-installation.pdf
    - Tutorial-xtra-slides.pdf

- PMlibパッケージの入手方法は、後半の「インストールとテスト」にて説明

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
- オープンソースソフトウエア（理研 AICSが開発・提供）
- 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
- アプリケーションのソースプログラム中にPMlib計測区間を指定して実行し、終了時に区間の統計情報を出力
- 主な用途として、計算負荷のホットスポット同定や、プロセス間の計算負荷バランスの確認に利用を想定
- ベンチマーク的な一時利用だけでなく、アプリケーションに常時組み込み、プロダクションランでの性能モデリング支援に利用される事を期待
- C++とFortranに対応したAPI

性能統計ツールの位置づけ



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
- GPU
- Phi
- SX

- オープンソース性能統計ツール類
  - 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
  - PAPI : HWPCへのアクセス
  - など多数、そしてPMlib

各ツールの位置づけ

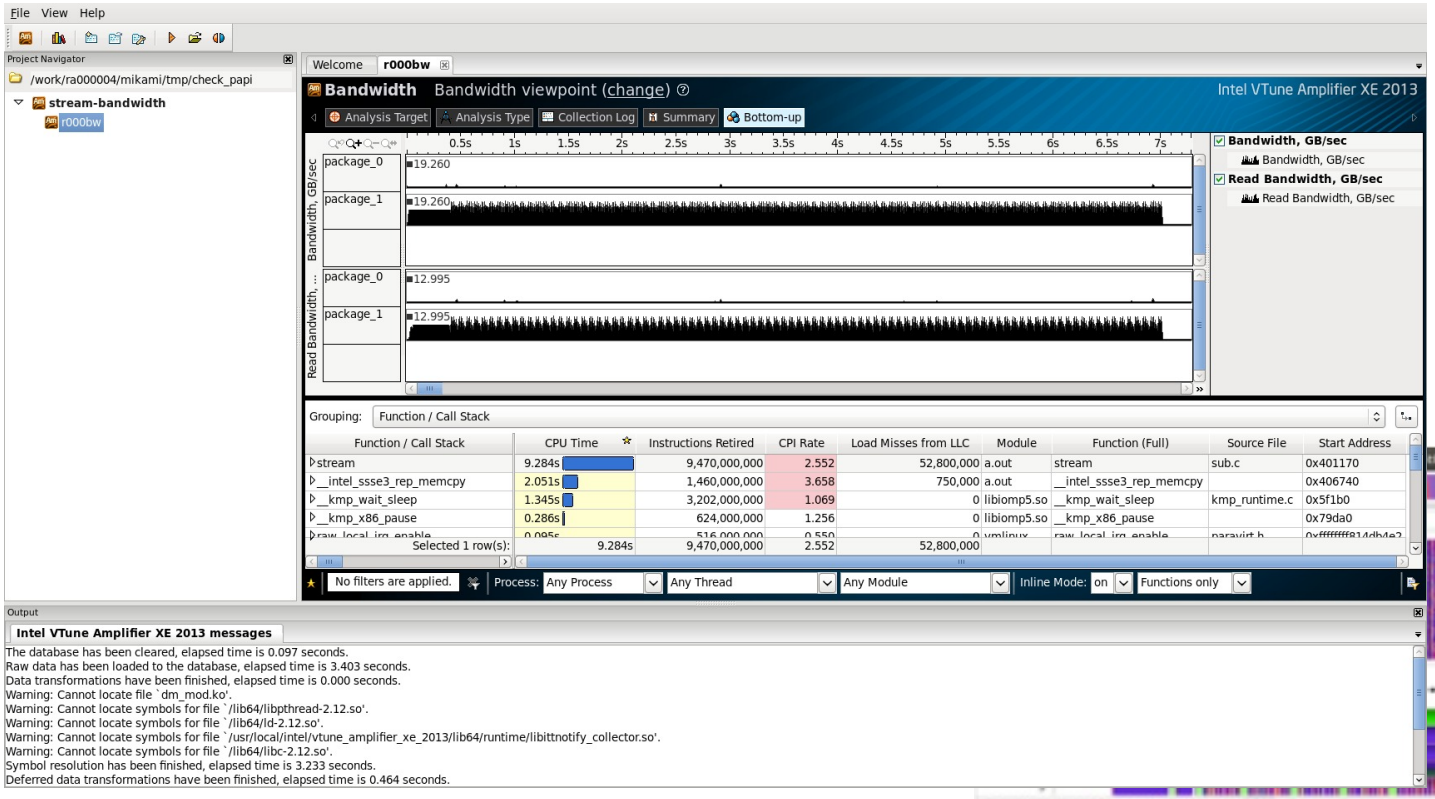
- ベンダー性能計測・統計ツール
  - 〇豊富な機能、高度なインタフェイス、システムに統合化された安心感、詳しいドキュメント、ベンダーによるサポート
  - △習熟に相当期間が必要、システム機種毎にツールが決まってしまう、それなりの価格

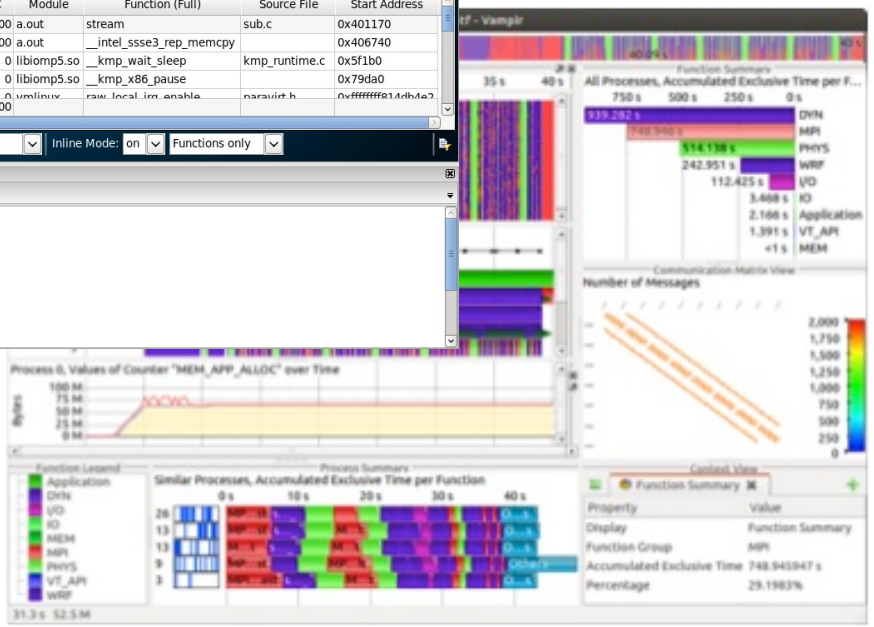
- オープンソース性能計測・統計ツール
  - 〇各ツール毎に高機能、無料
  - △ユーザーインタフェイスが個性的、インストールの手間・利用方法の習熟がそれなりに大変→周囲にツールをよく知っている人がいないとハードルは高い

- PMlib
  - オープンソース。テキストレポートを基本とした簡易なツール

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





⬆マニュアル 60MB online HTML

マニュアル100ページー＞

PMlibの特徴

- PMlibの特徴
  - テキストレポートを基本としたコンパクトなツール
    - 機能・出力情報を絞って容易に利用
  - インストールが簡単（後ほど実習）
    - プラットフォーム依存性が低い
  - 利用時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
    - ソース中のAPIへ明示的に計算量を申告
    - HWPCを用いた性能情報の自動取得（内部でPAPI利用）
  - 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呼び出しには未対応
- 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++とFortranに対応

Pmlibの動作確認がとれているシステム

- 京/FX10:
  - 計算ノード用(ログインノードでのクロスコンパイル可能)
  - 富士通コンパイラ＋富士通MPI
- Intel Xeon クラスタ: RedHat 6.1+、Suse 11.3+、Linux kernel 2.6.32+を推奨
  - Intelコンパイラ＋IntelMPI
  - GNUコンパイラ+OpenMPI/gnu
  - PGIコンパイラ+OpenMPI/pgi
- Apple Macbook: OSX 10.8以降
  - Apple Clang/LLVMコンパイラ＋OpenMPI
  - Intelコンパイラ＋OpenMPI
  - GNUコンパイラ
- 必要なソフトウエア環境
  - C++, C, Fortranコンパイラ
  - (必要に応じて)MPIライブラリ
  - (必要に応じて)PAPIライブラリ

PMlibの利用方法

- PMlibライブラリのインストール
- アプリケーションのソース中にPMlib APIの呼び出しを追加
- アプリケーション実行時に標準出力または指定ファイルへPMlib性能情報がレポートされる
- 出力情報の評価
  - テキスト情報の読み取り解釈
  - 別途パッケージGraphPlotFiltersによる簡易可視化

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.setProperties(" Kokodayo", 1);
     PM.start(" Kokodayo");
     subkerel();
     PM.stop (" Kokodayo", 0.0, 1);
     PM.gather();
     PM.print(stdout, " Mr.Bean");
     PM.printDetail(stdout);
     return 0;
}
```

ヘッダー部追加

初期設定

測定区間

レポートを出力

PMlib計算性能モニター機能

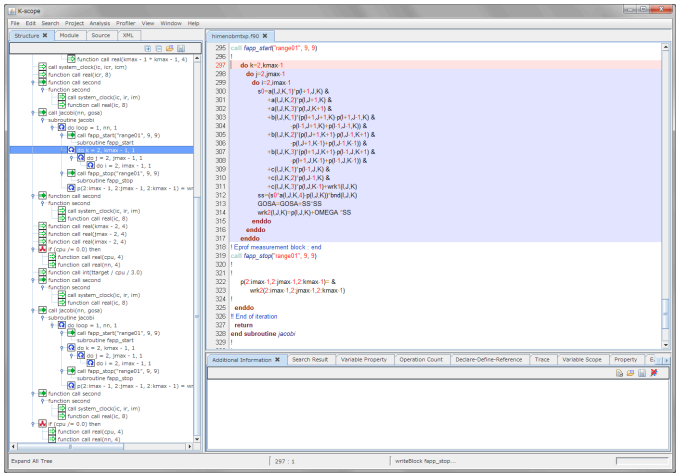
- 指定した測定区間毎に性能統計情報を蓄積・出力
- 各測定区間は少数のプロパティを持つ
  - ラベル：任意の文字列（統計情報出力時のラベル ）
  - 測定対象タイプ：「演算時間」、「通信時間」、「自動決定」
  - 排他測定フラグ：「排他測定」または「非排他測定」
- 性能統計の種類と算出方法を選択
  - 測定対象タイプによって、計算量を演算量（浮動小数点演算量）、あるいはデータ移動量として評価
- 計算量の決定方法
  - ユーザが明示的に申告する場合
    - 測定区間の計算量を計算式で引数として与える
  - ハードウエア性能カウンター（HWPC)を利用して自動採取

出力する情報

- １、基本レポート
  - 測定区間毎の平均プロファイル
  - プロセスあたりの平均プロファイル
  - ジョブあたりの総合性能
- ２、詳細プロファイル(1：MPIプロセス毎)
  - 各MPIプロセス毎のプロファイルを出力
  - プロセスがOpenMPスレッドを発生した場合、各スレッドの計算量は発生元プロセスに合計される。
- ３、詳細プロファイル(2：HWPCイベント統計)
  - HWPCイベントグループを環境変数で指定
  - 各MPIプロセス毎のHWPCイベント統計量を出力
  - プロセスがOpenMPスレッドを発生した場合、各スレッドの計算量は発生元プロセスに合計される。

測定区間の計算量：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
  - ソースプログラムで記述された計算量を式で引数化
    - 意図する計算量とその性能を忠実に測定可能
  - 測定区間毎の実行経過時間と演算数を内部で積算
  - ユーザ独自に計算の「重さ」を指定することも可能
- 計算量の明示的な評価方法（静的に数え上げ）
  - ソースプログラムからプログラマが読み取り分析
  - ツールで四則演算・メモリアクセス量の分析
    - Kscope (http://www.aics.riken.jp/ungi/soft/kscope/)など



測定区間の計算量 :HWPCによる自動算出

- 実行するシステムがCPUハードウエア性能カウンター(HWPC)を内部に持ち、そのイベント情報がPAPIライブラリで採取可能な場合は、PMlibが内部でPAPI低レベルAPIを呼び出してその統計情報を採取・積算する
- HWPCのイベントリスト別表(参考資料)
- PMlib用にイベントの種類毎にカウンターグループを定義
  - FLOPS
  - VECTOR
  - BANDWIDTH
  - CACHE
  - CYCLE
- プログラム実行時に環境変数HWPC_CHOOSERで選択する

出力する情報の選択

- 出力レポートに表示される情報はモード・引数の組み合わせで決める
- ユーザ申告モード
  - HWPC APIが利用できないシステムや環境変数HWPC_CHOOSERが指定されていないジョブでは自動的にユーザ申告モードで実行される。
  - ユーザ申告モードではsetProperties() とstop()への引数により出力内容が決定、HWPC詳細レポートは出力されない。
  - (1) ::setProperties(区間名, type, exclusive)の第2引数typeは測定量のタイプを指定する。計算(CALC)タイプか通信(COMM)タイプかの選択を行なう、ユーザ申告モードで有効な引数。
  - (2) ::stop (区間名, fPT, iC)の第2引数fPTは測定量。計算(浮動小数点演算)あるいは通信(MPI通信やメモリロードストアなどデータ移動)の量を数値や式で与える。

setProperties() type引数	stop() fP引数	基本・詳細レポート出力
CALC	指定あり	時間、fPT引数によるFlops
COMM	指定あり	時間、fPT引数によるByte/s
任意	指定なし	時間のみ

出力する情報の選択

- HWPCによる自動算出モード
  - HWPC/PAPIが利用可能なプラットフォームで利用できる
  - 環境変数HWPC_CHOOSERの値によりユーザ申告値を用いるかPAPI情報を用いるかを切り替える。
  - (FLOPS| BANDWIDTH| VECTOR| CACHE| CYCLE)

- ユーザ申告モードかHWPC自動算出モードかは、内部的に下記表の組み合わせで決定される。

環境変数 HWPC_CHOOSER	setProperties()の type引数	stop()の fP引数	基本・詳細レポート出力	HWPCレポート出力
NONE(無指定)	CALC	指定値	時間、fP引数によるFlops	なし
NONE(無指定)	COMM	指定値	時間、fP引数によるByte/s	なし
FLOPS	無視	無視	時間、HWPC自動計測Flops	FLOPSに関連するHWPC統計情報
VECTOR	無視	無視	時間、HWPC自動計測SIMD率	VECTORに関連するHWPC統計情報
BANDWIDTH	無視	無視	時間、HWPC自動計測Byte/s	BANDWIDTHに関連するHWPC統計情報
CACHE	無視	無視	時間、HWPC自動計測L1$,L2$	CACHEに関連するHWPC統計情報

基本レポート

```
# PMlib Basic Report -----------------------------------------------------

     Report of Timing Statistics PMlib version 4.1.2
     Linked PMlib supports: MPI, OpenMP, HWPC
     Operator  : Mr. Bean
     Host name :
     Date      : 2015/08/23 : 02:14:53
     Parallel Mode:   Hybrid (8 processes x 2 threads)

     Total execution time            = 2.135668e+00 [sec]
     Total time of measured sections = 2.197705e+00 [sec]

     Exclusive sections Statistics per process and per job.

     Section         |    call  |        accumulated time[sec]           | [flop counts or byte counts ]
     Label           |          |      avr   avr[%]      sdv   avr/call  |      avr       sdv   speed
     ----------------+----------+----------------------------------------+--------------------------
     First location  :        3   1.371e+00  62.37  1.80e-01  4.569e-01    0.000e+00  0.00e+00   0.00 Mflops
     Second location :        1   4.353e-01  19.81  4.01e-02  4.353e-01    4.000e+09  0.00e+00   9.19 Gflops
     Third location  :        1   3.917e-01  17.82  1.27e-01  3.917e-01    2.002e+09  0.00e+00   5.11 GB/sec
     ----------------+----------+----------------------------------------+--------------------------
     Per Process flop sections    1.806e+00                                 4.000e+09             2.21 Gflops
     Per Process byte sections    3.917e-01                                 2.002e+09             5.11 GB/sec
     ----------------+----------+----------------------------------------+--------------------------
     Job Total flop sections      1.806e+00                                 3.200e+10            17.72 Gflops
     Job Total byte sections      3.917e-01                                 1.602e+10            40.89 GB/sec
```

詳細レポート(プロセス毎)

```
# PMlib Process Report --- Elapsed time for individual MPI ranks ------

Label  First location
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   flop|msg    speed
Rank     0 :        3  1.316e+00   59.9  3.225e-01  4.387e-01  0.000e+00  0.000e+00 Flops
Rank     1 :        3  1.274e+00   58.0  3.647e-01  4.246e-01  0.000e+00  0.000e+00 Flops
Rank     2 :        3  1.224e+00   55.7  4.141e-01  4.081e-01  0.000e+00  0.000e+00 Flops
Rank     3 :        3  1.638e+00   74.6  0.000e+00  5.462e-01  0.000e+00  0.000e+00 Flops
Rank     4 :        3  1.198e+00   54.5  4.403e-01  3.994e-01  0.000e+00  0.000e+00 Flops
Rank     5 :        3  1.588e+00   72.3  5.060e-02  5.293e-01  0.000e+00  0.000e+00 Flops
Rank     6 :        3  1.214e+00   55.2  4.249e-01  4.045e-01  0.000e+00  0.000e+00 Flops
Rank     7 :        3  1.514e+00   68.9  1.250e-01  5.045e-01  0.000e+00  0.000e+00 Flops
Label  Second location
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   flop|msg    speed
Rank     0 :        1  4.103e-01   18.7  1.044e-01  4.103e-01  4.000e+09  9.750e+09 Flops
Rank     1 :        1  4.456e-01   20.3  6.900e-02  4.456e-01  4.000e+09  8.976e+09 Flops
Rank     2 :        1  4.646e-01   21.1  5.006e-02  4.646e-01  4.000e+09  8.610e+09 Flops
Rank     3 :        1  5.146e-01   23.4  0.000e+00  5.146e-01  4.000e+09  7.772e+09 Flops
Rank     4 :        1  3.903e-01   17.8  1.244e-01  3.903e-01  4.000e+09  1.025e+10 Flops
Rank     5 :        1  4.383e-01   19.9  7.631e-02  4.383e-01  4.000e+09  9.126e+09 Flops
Rank     6 :        1  4.110e-01   18.7  1.037e-01  4.110e-01  4.000e+09  9.733e+09 Flops
Rank     7 :        1  4.080e-01   18.6  1.067e-01  4.080e-01  4.000e+09  9.805e+09 Flops
Label  Third location
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   flop|msg    speed
Rank     0 :        1  4.039e-01   18.4  2.124e-01  4.039e-01  2.002e+09  4.957e+09 Bytes/sec
Rank     1 :        1  4.227e-01   19.2  1.936e-01  4.227e-01  2.002e+09  4.737e+09 Bytes/sec
Rank     2 :        1  4.964e-01   22.6  1.198e-01  4.964e-01  2.002e+09  4.033e+09 Bytes/sec
Rank     3 :        1  2.557e-01   11.6  3.605e-01  2.557e-01  2.002e+09  7.829e+09 Bytes/sec
   ...
```

詳細レポート(HWPC/PAPI)

```
 # PMlib hardware performance counter (HWPC) Report ------------------------
      HWPC_CHOOSER=FLOPS statistics were collected.
      The values of each process are the sum of threads.

Label   First location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.531e+10  5.100e+01  1.005e+10
Rank     1 :  1.534e+10  4.500e+01  1.184e+10
Rank     2 :  1.530e+10  4.700e+01  1.130e+10
Rank     3 :  1.537e+10  4.800e+01  1.243e+10
Rank     4 :  1.534e+10  4.300e+01  1.137e+10
Rank     5 :  1.535e+10  4.400e+01  1.148e+10
Rank     6 :  1.536e+10  5.200e+01  1.261e+10
Rank     7 :  1.530e+10  4.100e+01  1.012e+10
Label   Second location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  5.116e+09  1.000e+01  1.236e+10
Rank     1 :  5.111e+09  8.000e+00  1.254e+10
Rank     2 :  5.041e+09  9.000e+00  6.982e+09
Rank     3 :  5.118e+09  7.000e+00  1.254e+10
Rank     4 :  5.106e+09  8.000e+00  1.126e+10
Rank     5 :  5.058e+09  8.000e+00  8.807e+09
Rank     6 :  5.110e+09  7.000e+00  1.256e+10
Rank     7 :  5.105e+09  8.000e+00  1.261e+10
Label   Third location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  4.873e+09  1.000e+01  1.758e+10
Rank     1 :  5.079e+09  1.200e+01  1.315e+10
Rank     2 :  4.770e+09  1.200e+01  1.161e+10
...
```

HWPC legend 京コンピュータ

```
Detected CPU architecture:
    Sun
    Fujitsu SPARC64 VIIIfx
    The available HWPC events on this CPU architecture is limited.
 HWPC events legend:
    FP_OPS: floating point operations
    VEC_INS: vector instructions
    FMA_INS: Fused Multiply-and-Add instructions
    LD_INS: memory load instructions
    SR_INS: memory store instructions
    L1_TCM: level 1 cache miss
    L2_TCM: level 2 cache miss (by demand and by prefetch)
    L2_WB_DM: level 2 cache miss by demand with writeback request
    L2_WB_PF: level 2 cache miss by prefetch with writeback request
    TOT_CYC: total cycles
    MEM_SCY: Cycles Stalled Waiting for memory accesses
    STL_ICY: Cycles with no instruction issue
    TOT_INS: total instructions
    FP_INS: floating point instructions
 Derived statistics:
    [GFlops]: floating point operations per nano seconds (10^-9)
    [Mem GB/s]: memory bandwidth in load+store GB/s
    [L1$ %]: Level 1 cache hit percentage
    [LL$ %]: Last Level cache hit percentage
```

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
```

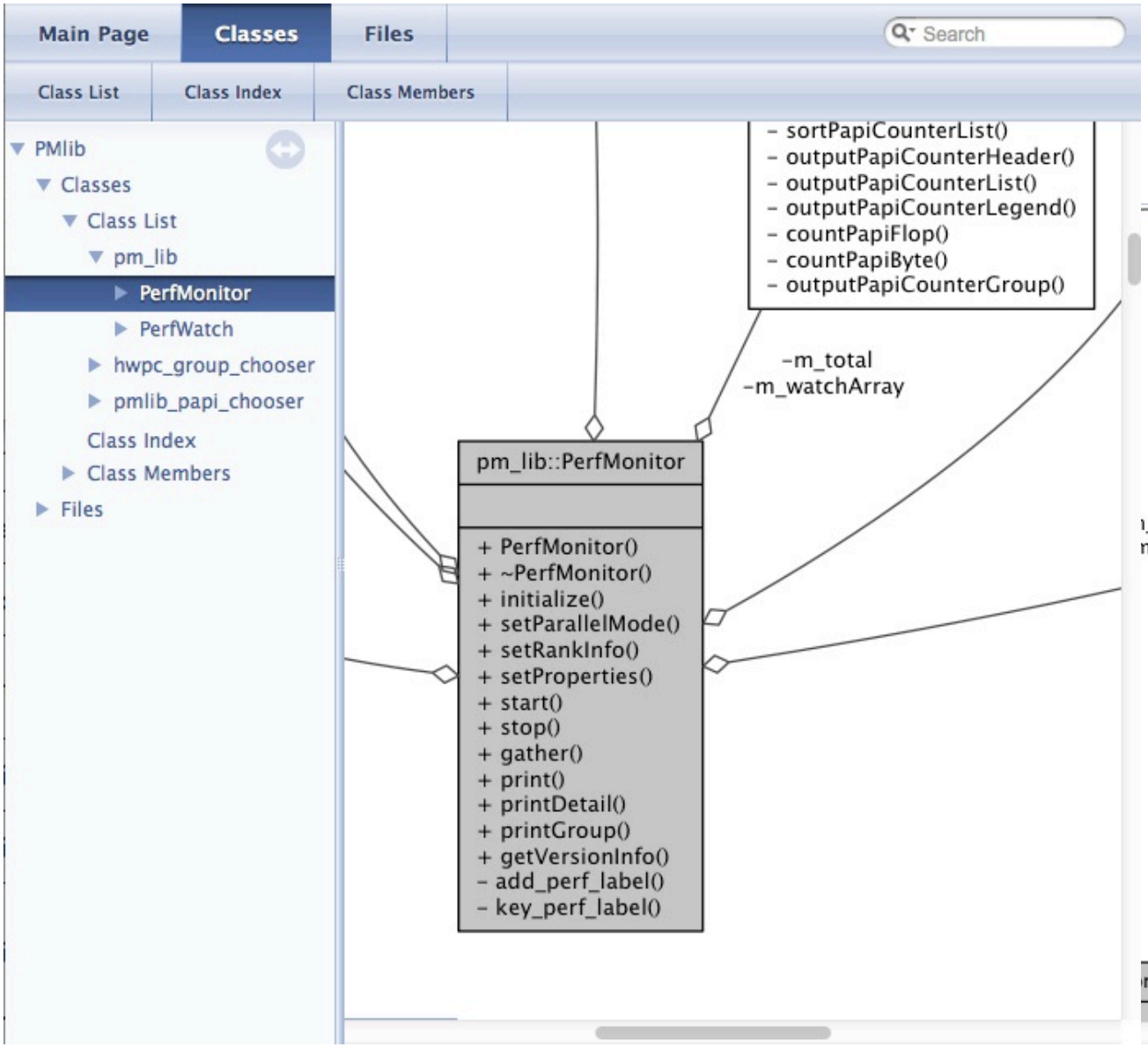
PMlib関数一覧

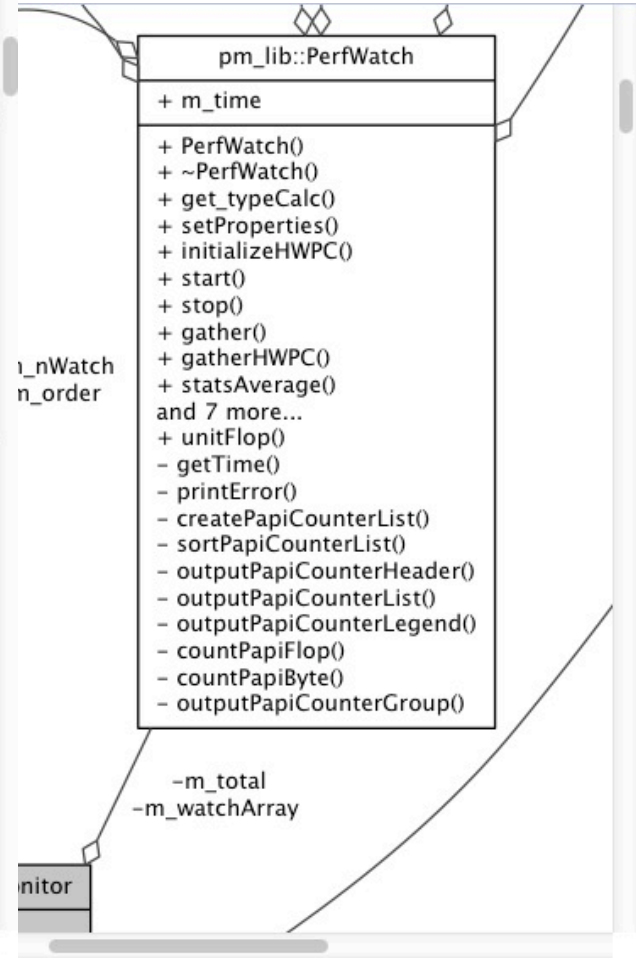
関数名（C++）	関数名（Fortran）	機能	呼び出し位置と回数	引数
initialize()	f_pm_initialize()	PMlib全体の初期化	冒頭・一回	(1)測定区間数
setProperties()	f_pm_setproperties()	測定区間のラベル化	任意・区間毎一回	(1)ラベル、(2)測定対象タイプ、(3)排他指定
start()	f_pm_start()	測定の開始	任意(startとstopでペア)	(1)ラベル
stop()	f_pm_stop()	測定の停止	任意(startとstopでペア)	(1)ラベル、(2)計算量、(3)計算のタスク数
gather()	f_pm_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()	f_pm_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)任意のコメント、(4)区間の表示順序指定
printDetail()	f_pm_printdetail()	MPIランク毎の詳細性能情報の表示	測定終了後・一回	(1)出力ファイルポインタ、(2)記号説明の表示、(3)区間の表示順序指定
printGroup ()	f_pm_printgroup ()	指定プロセスグループに属するMPIランク毎の詳細性能情報の表示	測定終了後・グループ数回	(1)出力ファイルポインタ、(2)group handle、(3)communicator、(4)rank番号配列、(5)グループ番号、(6) 記号説明の表示、(7)区間の表示順序指定
printComm ()		MPI_Comm_split()で作成したcommunicatorからプロセスグループ自動作成し、printGroup ()をよびだすヘルパー関数	測定終了後・一回	(1)出力ファイルポインタ、(2)communicator handle、(3)カラー変数、(4)key変数、(5) 記号説明の表示、(6)区間の表示順序指定

関数の仕様や引数詳細説明は doc/ディレクトリでDoxygen生成（後出）

各関数の仕様

Doc/html/index.html
Webブラウザで表示





各関数の仕様　initialize()

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

初期化. 測定区間数分の測定時計を準備. 最初にinit_nWatch区間分を確保し、不足したら動的にinit_nWatch追加する 全計算時間用測定時計をスタート.

Parameters

[in] **init_nWatch** 最初に確保する測定区間数（C++では省略可能）

Note

測定区間数 m_nWatch は不足すると動的に増えていく

各関数の仕様　setProperties()

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

測定区間にプロパティを設定.

Parameters

- [in] **label** ラベルとなる文字列
- [in] **type** 測定量のタイプ(COMM:通信, CALC:計算)
- [in] **exclusive** 排他測定フラグ。bool型(省略時true)、 Fortran仕様は整数型(0:false, 1:true)

Note

labelラベル文字列は測定区間を識別するために用いる。 各ラベル毎に対応したキー番号 key を内部で自動生成する 最初に確保した区間数init_nWatchが不足したら動的にinit_nWatch追加する

各関数の仕様　start()/stop()

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

測定区間スタート

Parameters

[in] **label** ラベル文字列。測定区間を識別するために用いる。

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

測定区間ストップ

Parameters

[in] **label** ラベル文字列。測定区間を識別するために用いる。

[in] **flopPerTask** 測定区間の計算量(演算量Flopまたは通信量Byte):省略値0

[in] **iterationCount** 計算量の乗数（反復回数）:省略値1

Note

引数とレポート出力情報の関連はPerfWatch.hのコメントに詳しく説明されている。

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()

```
void pm_lib::PerfMonitor::print ( FILE *            fp,
                                  const std::string hostname,
                                  const std::string comments,
                                  int               seqSections = 0
                                )
```

測定結果の基本統計レポートを出力。 排他測定区間毎に出力。プロセスの平均値、ジョブ全体の統計値も出力。

Parameters

[in]	**fp**	出力ファイルポインタ
[in]	**hostname**	ホスト名(省略時はrank 0 実行ホスト名)
[in]	**comments**	任意のコメント
[in]	**seqSections**	(省略可)測定区間の表示順 (0:経過時間順、1:登録順で表示)

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　printDetail()

```
void pm_lib::PerfMonitor::printDetail ( FILE *  fp,
                                        int     legend = 0,
                                        int     seqSections = 0
                                      )
```

MPIランク別詳細レポート、HWPC詳細レポートを出力。 非排他測定区間も出力

Parameters

[in]	**fp**	出力ファイルポインタ
[in]	**legend**	int型 (省略可) HWPC 記号説明の表示 (0:なし、1:表示する)
[in]	**seqSections**	(省略可)測定区間の表示順 (0:経過時間順、1:登録順で表示)

Note

本APIよりも先にPerfWatch::gather()を呼び出しておく必要が有る HWPC値は各プロセス毎に子スレッドの値を合算して表示する

各関数の仕様　printGroup()

```
void pm_lib::PerfMonitor::printGroup ( FILE *        fp,
                                       MPI_Group     p_group,
                                       MPI_Comm      p_comm,
                                       int *         pp_ranks,
                                       int           group = 0,
                                       int           legend = 0,
                                       int           seqSections = 0
                                     )
```

プロセスグループ単位でのMPIランク別詳細レポート、HWPC詳細レポート出力

Parameters

[in]	**fp**	出力ファイルポインタ
[in]	**p_group**	MPI_Group型 groupのgroup handle
[in]	**p_comm**	MPI_Comm型 groupに対応するcommunicator
[in]	**pp_ranks**	int*型 groupを構成するrank番号配列へのポインタ
[in]	**group**	int型 (省略可)プロセスグループ番号
[in]	**legend**	int型 (省略可)HWPC記号説明の表示(0:なし、1:表示する)
[in]	**seqSections**	int型 (省略可)測定区間の表示順 (0:経過時間順、1:登録順で表示)

Note

プロセスグループはp_group によって定義され、p_groupの値は MPIライブラリが内部で定める大きな整数値を基準に決定されるため、 利用者にとって識別しずらい場合がある。 別に1,2,3,..等の昇順でプロセスグループ番号 groupをつけておくと レポートが識別しやすくなる。

各関数の仕様　printComm()

```
void pm_lib::PerfMonitor::printComm ( FILE *        fp,
                                      MPI_Comm      new_comm,
                                      int           icolor,
                                      int           key,
                                      int           legend = 0,
                                      int           seqSections = 0
                                    )
```

MPI communicatorから自動グループ化したMPIランク別詳細レポート、HWPC詳細レポートを出力

Parameters

[in]	**fp**	出力ファイルポインタ
[in]	**p_comm**	MPI_Comm型 MPI_Comm_split()で対応つけられたcommunicator
[in]	**icolor**	int型 MPI_Comm_split()のカラー変数
[in]	**key**	int型 MPI_Comm_split()のkey変数
[in]	**legend**	int型 (省略可)HWPC記号説明の表示(0:なし、1:表示する)
[in]	**seqSections**	int型 (省略可)測定区間の表示順 (0:経過時間順、1:登録順で表示)

PMlib利用プログラム例(再掲)

- 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.setProperties(" Kokodayo", 1);
     PM.start(" Kokodayo");
     subkerel();
     PM.stop (" Kokodayo", 0.0, 1);
     PM.gather();
     PM.print(stdout, " Mr.Bean");
     PM.printDetail(stdout);
     return 0;
}
```

ヘッダー部追加

初期設定

測定区間

レポートを出力

前半の資料説明終了