10分でわかるPMlib 😚

理化学研究所 計算科学研究機構
可視化技術研究チーム
2017年7月11日

PMlibとは

- アプリケーションの計算性能をモニターするソフトウエアライブラリ
  - 注目する区間の性能統計情報を簡単に測定・レポートする。
  - C++言語、Fortran言語に対応 。
  - Linux系OSをもつコンピュータの上で使える。
    - 京コンピュータ・FX100:
    - Intel Xeon系サーバ・PC
    - Apple Macbook
  - オープンソース。
    - パッケージ公開リポジトリ
      - http://avr-aics-riken.github.io/PMlib/
    - 必要な前提ソフトウエア環境
      - Linux OS, C++, C, Fortranコンパイラ
      - （オプションに応じて）MPIライブラリ、PAPIライブラリ、OTFライブラリ

計算性能のモニターとは？

- 注目する区間を指定して性能統計情報を蓄積・記録すること

- 測定区間は少数の属性を持つ
  - ラベル：任意の名称　”ここ”とか”そこ”とか
  - 測定する計算量の種類：「演算量」、「通信量」など　＊1

    ＊1　浮動小数点演算量、データ移動量、その他HW固有の統計量がいくつか選べる
  - 排他性：「排他的」か「非排他的」 他の区間とカブっていないかという意味

- 測定する計算量をどう選択・算出するか？
  - PMlibがHWPC値を自動測定する方法
  - ユーサ゛か゛明示的に申告する方法

PMlibの利用方法

1. PMlibライブラリをインストールする
2. アプリケーション中の測定区間を決める
   – ソースプログラム中の注目箇所にPMlib APIを追加
3. アプリケーションをコンパイルしてPMlibとリンクする
4. アプリケーションを実行する
   – 実行時に性能統計情報がレポートされる
5. 性能統計レポートを確認評価する

PMlib基本APIの一覧

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・各区間一回	(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)・任意	(1)ラベル
stop()	f_pm_stop()	測定の停止	任意(startとstopでペア)・任意	(1)ラベル、(2)計算量、(3)任意の係数
print()	f_pm_print()	測定区間毎の基本統計結果表示	測定終了時・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	測定終了時・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

PMlibを利用するプログラムの構成例（Fortran）

- 元のソース

```
program main

call mykernel()

end
```

注目箇所

- PMlib組み込み後のソース

```
program main
call f_pm_initialize (nWatch)
call f_pm_setproperties ("Koko!" icalc, iexcl)
call f_pm_start ("Koko!")
call mykernel (msize,n,a,b,c)
call f_pm_stop ("Koko!", fops, ncall)
call f_pm_print ("", isort)
call f_pm_printdetail ("", ilegend, isort)
end
```

初期設定

測定区間

レポート出力

PMlibを利用するプログラムの構成例（C++）

- 元のソース

```
int main(int argc, char *argv[])
{
mykernel(); //注目箇所
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("Koko!", PM.CALC);
PM.start("Koko!");
mykernel();
PM.stop ("Koko!");
PM.print(stdout, "", "");
PM.printDetail(stdout);
return 0;
}
```

PMlibヘッダー

初期設定

測定区間

レポート出力

PMlibの出力情報

- １、基本レポート
  - 各測定区間のプロセスあたり平均性能統計値
    - 時間：各区間の平均時間、呼び出し回数、累積経過時間
    - 計算量：呼び出し1回あたりの量、合計量、速度
    - 区間を登録順または経過時間順にソート出力
  - ジョブ全体での総合性能
- ２、詳細プロファイル
  - 各MPIプロセス毎のプロファイルを出力
  - （オプション）各MPIプロセス毎のHWPCイベント統計量
    - 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、各スレッドの計算量は元プロセスに合算する。
- ３、（オプション）ポスト処理用性能トレースファイル

基本レポート例 (HWPC自動測定モード)

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

      Timing Statistics Report from PMlib version 5.0.3
      Linked PMlib supports: MPI, OpenMP, HWPC, OTF
      Host name : vsp01
      Date      : 2016/06/19 : 15:26:50
      Mrs. Kobe
      Parallel Mode:   Hybrid (4 processes x 4 threads)
      The environment variable HWPC_CHOOSER=FLOPS is provided.

      Total execution time            = 9.848690e-01 [sec]
      Total time of measured sections = 9.816217e-01 [sec]

      Exclusive sections statistics per process and total job.
      Inclusive sections are marked with (*)

      Section           |    call  |        accumulated time[sec]           | [hardware counter byte counts]
      Label             |          |      avr    avr[%]      sdv   avr/call  |      avr       sdv    speed
      ------------------+----------+----------------------------------------+---------------------------
      First section     :        1   1.039e-01  10.59  1.32e-03  1.039e-01    4.807e+09  1.89e+06  46.26 Gflops
      Second section(*) :        1   8.412e-01  85.70  4.72e-03  8.412e-01    5.226e+09  1.79e+06   6.21
Gflops(*)
      Subsection X      :        3   3.106e-01  31.64  9.48e-04  1.035e-01    1.614e+10  3.24e+06  51.97 Gflops
      Subsection Y      :        3   3.127e-01  31.85  4.06e-03  1.042e-01    1.568e+10  2.73e+06  50.14 Gflops
      ------------------+----------+----------------------------------------+---------------------------
      Sections per process          7.272e-01     -Exclusive CALC sections-  3.663e+10           50.37 Gflops
      ------------------+----------+----------------------------------------+---------------------------
      Sections total job            7.272e-01     -Exclusive CALC sections-  1.465e+11          201.47 Gflops
```

基本レポート例（ユーザ申告モード）

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

      Timing Statistics Report from PMlib version 5.0.3
      Linked PMlib supports: MPI, OpenMP, HWPC, OTF
      Host name : vsp01
      Date      : 2016/06/19 : 15:28:19
      Mrs. Kobe
      Parallel Mode:   Hybrid (4 processes x 4 threads)
      The environment variable HWPC_CHOOSER is not provided. No HWPC report.

      Total execution time            = 9.795189e-01 [sec]
      Total time of measured sections = 9.816882e-01 [sec]

      Exclusive sections statistics per process and total job.
      Inclusive sections are marked with (*)

      Section           |    call  |        accumulated time[sec]           | [user defined counter values ]
      Label             |          |      avr   avr[%]      sdv   avr/call  |      avr       sdv   speed
      ------------------+----------+----------------------------------------+---------------------------
      First section     :        1   1.043e-01  10.62  1.47e-03  1.043e-01    4.000e+09  0.00e+00  38.35 Gflops
      Second section(*) :        1   8.420e-01  85.77  6.86e-03  8.420e-01    1.960e+10  0.00e+00  23.28
Gflops(*)
      Subsection X      :        3   3.120e-01  31.78  3.28e-03  1.040e-01    4.800e+10  0.00e+00 153.84 GB/sec
      Subsection Y      :        3   3.118e-01  31.76  2.72e-03  1.039e-01    1.440e+10  0.00e+00  46.18 Gflops
      ------------------+----------+----------------------------------------+---------------------------
      Sections per process          4.161e-01     -Exclusive CALC sections-  1.840e+10           44.22 Gflops
      Sections per process          3.120e-01     -Exclusive COMM sections-  4.800e+10          153.84 GB/sec
      ------------------+----------+----------------------------------------+---------------------------
      Sections total job            4.161e-01     -Exclusive CALC sections-  7.360e+10          176.87 Gflops
      Sections total job            3.120e-01     -Exclusive COMM sections-  1.920e+11          615.36 GB/sec
```

以降のスライドはコンピュータシステム毎に
別れた内容になっています。
Intel環境編
京・FX100編
Mac・OSX編
適切なものを選んでお読みください

10分+でできるPMlibのインストールと利用😚

Intel環境編
（Intel サーバ w/ Intelコンパイラ＋Intel MPI）

PMlibのインストールと利用実行

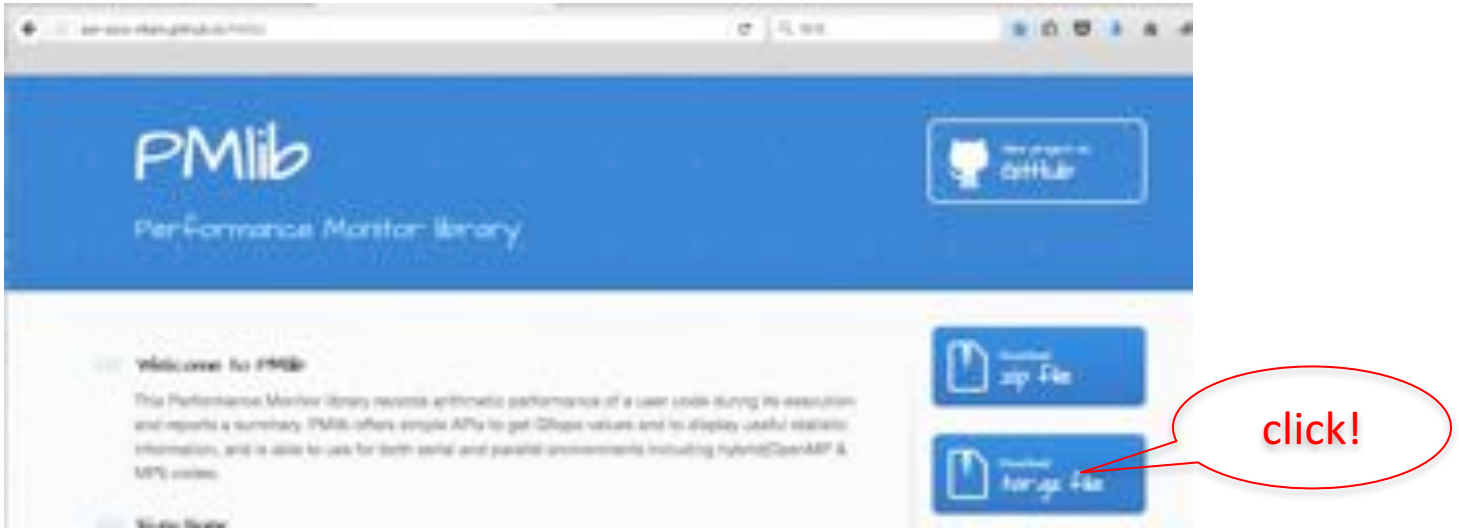
- PMlibのインストール
  - PMlibパッケージの入手
  - PMlibのインストール

- PMlibの利用実行
  - 例題プログラムの作成（C++言語で作成）
  - 例題プログラムへのPMlibの追加（ソースプログラムの編集）
  - 例題プログラムをコンパイルしてPMlibをリンクする
  - 例題プログラムを実行して、PMlibのレポートを確認する

- （注）ここではPMlibのインストールと例題プログラムの利用実行が同じ種類のIntel Xeon CPU上で行われることを想定している。
- （注）ここではIntelコンパイラ＋Intel MPIのソフトウエア環境を想定している。GNUコンパイラを用いた場合、あるいはOpenMPIを用いた場合などのインストール例についてはパッケージに含まれる doc/scripts/GNU 以下のシェルスクリプトを参照

PMlibパッケージの入手（共通）

- パッケージ公開リポジトリ
  - http://avr-aics-riken.github.io/PMlib/



- ダウンロードしたファイル名は avr-aics-riken-PMlib-*.tar.gz
  - （*の部分はバージョンにより変わる）
- ダウンロードしたファイルをインストール先のコンピュータに転送する。
  - 以降の例では ${HOME}/tmp/　下に転送したと仮定している

PMlibパッケージの展開（共通）

- インストール先のコンピュータ上で、転送したパッケージを展開する
- 展開したディレクトリにシンボリックリンクと、パスの環境変数を設定する。
- 以下の例ではログイン後ホームに pmlib ディレクトリを作って、その下に転送したパッケージのファイルを展開する。

```
$  mkdir ~/pmlib
$  cd ~/pmlib
$  tar –zxf  ~/tmp/avr-aics-riken-PMlib-*.tar.gz
$  ls –go
drwxr-xr-x  9 4096 2017-06-22 14:31 avr-aics-riken-PMlib-073c316
$  mv  avr-aics-riken-PMlib-*  package
$  export  PACKAGE_DIR=~/pmlib/package
```

- Intelサーバ/Intel環境, Intelサーバ/GNU環境、FX100システム、京コンピュータ、Macbookの各システムに対応したインストールスクリプトが以下に提供されている。

```
$ ls  –F  package/doc/scripts/
GNU/       Intel/     Kcomputer/ Mac/       fx100/
```

PMlibのインストール　Intel環境

- Intel環境用のインストールスクリプトを確認する

```
$  vi  ${PACKAGE_DIR}/doc/scripts/Intel/x.cmake-intel.sh
```

- コンパイラ・ライブラリはシステムによってインストールされているパスが異なるので、スクリプトで設定されているパスが正しいか確認し、必要であれば修正する。

```
N行目　　INTEL_DIR=/usr/local/intel/composer_xe_2015
N行目　　MPI_DIR=/usr/local/intel/impi/5.0.3.048
N行目　　PACKAGE_DIR=~/pmlib/package
N行目　　PMLIB_DIR=~/pmlib/usr_local_pmlib/intel
N行目　　PAPI_DIR=/usr/local/papi/papi-5.5.1/intel
```

- INTEL_DIR　　# INTELコンパイラディレクトリ（必須：既存）
- MPI_DIR　　# INTEL MPIディレクトリ（必須：既存）*Intel 2016版以降は不要
- PACKAGE_DIR # PMlibパッケージを展開したディレクトリ（必須：既存）
- PMLIB_DIR　　# PMlibのインストール先ディレクトリ（必須：新規・更新）
- PAPI_DIR　　# PAPIライブラリ（オプション：既存ディレクトリ、または”no”）

PMlibのインストール　Intel環境

- インストールスクリプトを実行する。

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.cmake-intel.sh  2>&1 | tee stdout.make
```

- 以下のファイルがインストールされた事を確認する

```
$  ls –go ${PMLIB_DIR}
drwxr-xr-x 2 140  7月   4 16:20 doc
drwxr-xr-x 2 117  7月   4 16:20 include
drwxr-xr-x 2  57  7月   4 16:20 lib
drwxr-xr-x 2  80  7月   4 16:20 share

$ ls –go ${PMLIB_DIR}/lib
-rw-r--r-- 1 170276  7月   4 16:20 libPM.a          # 1プロセス版PMlibライブラリ
-rw-r--r-- 1 175460  7月   4 16:19 libPMmpi.a       # MPI版PMlibライブラリ
-rw-r--r-- 1   5470  7月   4 16:20 libpapi_ext.a    # PAPI拡張ライブラリ
```

- 以上でPMlibインストール終了！
- では続いてPMlibを使ってみよう。

例題プログラムの作成（C++版）

- 適当なディレクトリ($MY_DIRとする)の下にプログラム mxm.cpp を作成する

```
$  MY_DIR=~/pmlib/my_dir
$  mkdir  -p ${MY_DIR}
$  cd  ${MY_DIR}
```

- N次の正方行列の積を計算するプログラム
  - 主プログラム：関数１と関数2を呼び出して行列積の計算を行う。
  - 関数１：正方行列[A]、[B]の各要素を値1.0で初期化する
  - 関数２：行列積[C]=[A][B]を計算する
  - シリアルプログラム（MPI不要、OpenMP不要）
- 手早く進みたい場合はパッケージのプログラム例をコピーすると良い
- 自分でソースを書いてももちろん良い　vi　mxm.cpp

```
$  vi  mxm.cpp
       あるいは
$  cp –p ${PACKAGE_DIR}/doc/src_tutorial/mxm.cpp  ./
```

行列積ソースプログラム例 C++

```
#include <stdio.h>
#include <string.h>
#include "matrix.h"
void init2d();
void mxm2d();

int main()
{
    init2d();
    mxm2d();
    return 0;
}
```

主プログラム部分

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] = (double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            matrix.c2d[j][i] = c1;
        }
    }
}
```

ヘッダーファイル matrix.h　の内容

```
#define MATSIZE 1000
struct matrix {
    double a2d[MATSIZE][MATSIZE];
    double b2d[MATSIZE][MATSIZE];
    double c2d[MATSIZE][MATSIZE];
    int nsize;
} matrix;
```

例題プログラムへのPMlibの追加

- 元の主プログラム部分

```
int main()
{

    init2d();

    mxm2d();

    return 0;
}
```

ヘッダー

初期設定

測定区間1

測定区間2

レポートを出力

- PMlibを追加した主プログラム部分

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main()
{
PM.initialize();
PM.setProperties("A:init2d", PM.CALC);
PM.setProperties("B:mxm2d", PM.CALC);
PM.start("A:init2d");
init2d();
PM.stop ("A:init2d");
PM.start("B:mxm2d");
mxm2d();
PM.stop ("B:mxm2d");
PM.print(stdout, "", "");
PM.printDetail(stdout);
return 0;
}
```

ヘッダー追加

初期設定

測定区間1

測定区間2

レポートを出力

例題プログラムのコンパイル・実行　Intel環境

- 例題プログラムをコンパイルしてPMlibをリンクするジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/Intel/x.myprog.sh
- スクリプトの内容を確認する
  - MY_DIR, INTEL_DIR, PMLIB_DIR, PAPI_DIRのパス
  - PMlibはMPI版(-lPMmpi)か1プロセス版 (-lPM) かを選択してリンクする。
  - この例題プログラムは1プロセス版なので -lPMを選択している
- ジョブスクリプトを実行する。
  - スクリプトでは対話的なコンパイル・実行を想定している。バッチジョブとして実行する場合はバッチジョブ管理ソフトに合わせてジョブ指示行を適宜追加する。
  - 最初は環境変数を指定しないユーザー申告モードで実行

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.myprog.sh
```

  - 次に環境変数HWPC_CHOOSERを指定して実行する。(HWPC自動測定モード)

```
export HWPC_CHOOSER=FLOPS
./a.out
```

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.myprog.sh
```

例題プログラムへのOpenMP指示行の追加

- ソースプログラムにOpenMP指示行を追加

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] =
(double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j,k,c1)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            …以下略…
```

- 環境変数を追加・変更して再実行
  - ジョブスクリプトで測定条件を変更して出力結果を比較する
  - OMP_NUM_THREADS=1/2/4/8など：OpenMPスレッド数
  - HWPC_CHOOSER=FLOPS/BANDWIDTH/など：HWPC測定のタイプ

10分+でできるPMlibのインストールと利用😚

FX100・京コンピュータ編

PMlibのインストールと利用実行

- PMlibのインストール
  - PMlibパッケージの入手
  - PMlibのインストール

- PMlibの利用実行
  - 例題プログラムの作成（C++言語で作成）
  - 例題プログラムへのPMlibの追加（ソースプログラムの編集）
  - 例題プログラムをコンパイルしてPMlibをリンクする
  - 例題プログラムを実行して、PMlibのレポートを確認する

- （注）この説明資料ではPMlibのインストール・例題プログラムのコンパイルはログインノード上で行い、例題プログラムの実行は計算ノード上で行われることを前提としている。

PMlibのインストールと利用実行

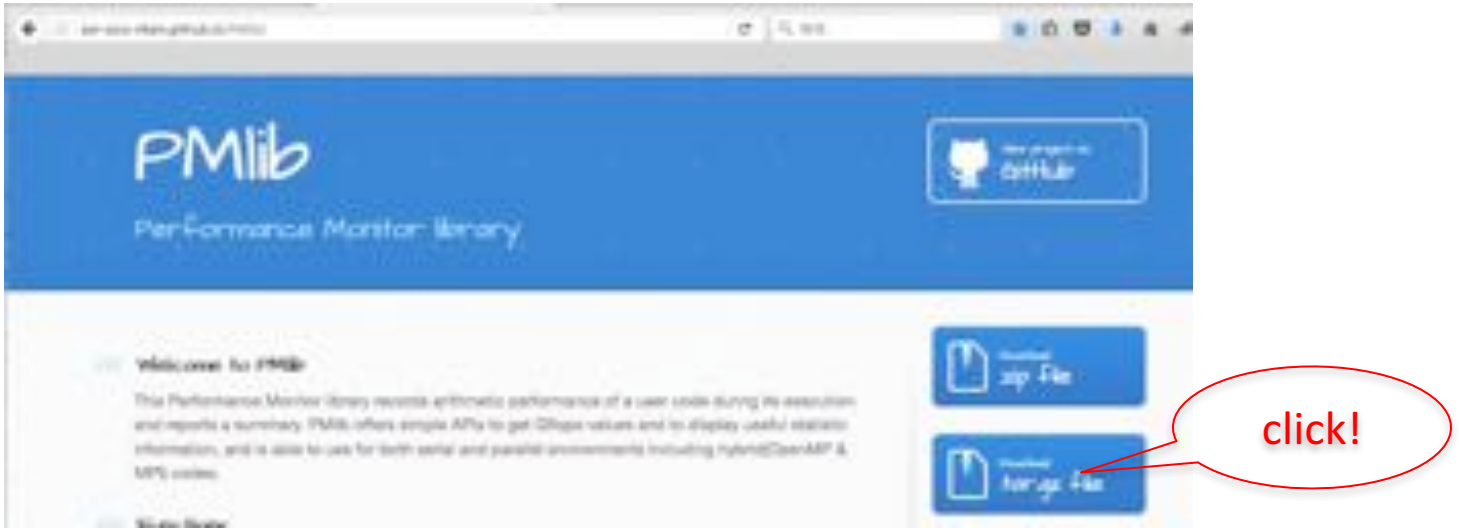
- PMlibのインストール
  - PMlibパッケージの入手
  - PMlibのインストール

- PMlibの利用実行
  - 例題プログラムの作成（C++言語で作成）
  - 例題プログラムへのPMlibの追加（ソースプログラムの編集）
  - 例題プログラムをコンパイルしてPMlibをリンクする
  - 例題プログラムを実行して、PMlibのレポートを確認する

- （注）ここではPMlibのインストールと例題プログラムの利用実行が同じ種類のIntel Xeon CPU上で行われることを想定している。
- （注）ここではIntelコンパイラ＋Intel MPIのソフトウエア環境を想定している。GNUコンパイラを用いた場合、あるいはOpenMPIを用いた場合などのインストール例についてはパッケージに含まれる doc/scripts/GNU 以下のシェルスクリプトを参照

PMlibパッケージの入手（共通）

- パッケージ公開リポジトリ
  - http://avr-aics-riken.github.io/PMlib/



- ダウンロードしたファイル名は avr-aics-riken-PMlib-*.tar.gz
  - （*の部分はバージョンにより変わる）
- ダウンロードしたファイルをインストール先のコンピュータに転送する。
  - 以降の例では ${HOME}/tmp/　下に転送したと仮定している

PMlibパッケージの展開（共通）

- インストール先のコンピュータ上で、転送したパッケージを展開する
- 展開したディレクトリにシンボリックリンクと、パスの環境変数を設定する。
- 以下の例ではログイン後ホームに pmlib ディレクトリを作って、その下に転送したパッケージのファイルを展開する。

```
$  mkdir ~/pmlib
$  cd ~/pmlib
$  tar –zxf  ~/tmp/avr-aics-riken-PMlib-*.tar.gz
$  ls –go
drwxr-xr-x  9 4096 2017-06-22 14:31 avr-aics-riken-PMlib-073c316
$  mv  avr-aics-riken-PMlib-*  package
$  export  PACKAGE_DIR=~/pmlib/package
```

- Intelサーバ/Intel環境, Intelサーバ/GNU環境、FX100システム、京コンピュータ、Macbookの各システムに対応したインストールスクリプトが以下に提供されている。

```
$ ls  –F  package/doc/scripts/
GNU/       Intel/     Kcomputer/ Mac/       fx100/
```

PMlibのインストール　FX100・京コンピュータ

- FX100・京コンピュータ用のインストールスクリプトは共通で、以下に例が提供されているので確認する

```
$ vi ${PACKAGE_DIR}/doc/scripts/K/x.cmake-K-all.sh
```

- パッケージが展開されたディレクトリ、PMlibのインストール先ディレクトリのパスが正しいか確認し、必要であれば修正する。
- 京コンピュータのコンパイラ・ライブラリパスはシステムが自動認識する。

```
N行目　PACKAGE_DIR=${HOME}/pmlib/package
N行目　PMLIB_DIR=${HOME}/pmlib/usr_local_pmlib/K
N行目　PAPI_DIR="yes”
```

- PACKAGE_DIR # PMlibパッケージを展開したディレクトリ（必須：既存）
- PMLIB_DIR　　# PMlibのインストール先ディレクトリ（必須：新規・更新）
- PAPI_DIR　　# PAPIライブラリディレクトリ（オプション：既存、”yes” で自動認識）

PMlibのインストール　FX100・京コンピュータ

- ログインノード上でインストールスクリプトを実行する。

```
$  ${PACKAGE_DIR}/doc/scripts/K/x.cmake-K-all.sh  2>&1 | tee stdout.make
```

- 以下のファイルがインストールされた事を確認する

```
$  ls –go ${PMLIB_DIR}
drwxr-xr-x 2 140  7月   4 16:20 doc
drwxr-xr-x 2 117  7月   4 16:20 include
drwxr-xr-x 2  57  7月   4 16:20 lib
drwxr-xr-x 2  80  7月   4 16:20 share

$ ls –go ${PMLIB_DIR}/lib
-rw-r--r-- 1 170276  7月   4 16:20 libPM.a          # 1プロセス版PMlibライブラリ
-rw-r--r-- 1 175460  7月   4 16:19 libPMmpi.a       # MPI版PMlibライブラリ
-rw-r--r-- 1   5470  7月   4 16:20 libpapi_ext.a    # PAPI拡張ライブラリ
```

- 以上でPMlibインストール終了！
- では続いてPMlibを使ってみよう。

例題プログラムの作成（C++版）

- 適当なディレクトリ($MY_DIRとする)の下にプログラム mxm.cpp を作成する

```
$  MY_DIR=~/pmlib/my_dir
$  mkdir  -p ${MY_DIR}
$  cd  ${MY_DIR}
```

- N次の正方行列の積を計算するプログラム
  - 主プログラム：関数1と関数2を呼び出して行列積の計算を行う。
  - 関数1：正方行列[A]、[B]の各要素を値1.0で初期化する
  - 関数2：行列積[C]=[A][B]を計算する
  - シリアルプログラム（MPI不要、OpenMP不要）
- 手早く進みたい場合はパッケージのプログラム例をコピーすると良い
- 自分でソースを書いてももちろん良い　vi　mxm.cpp

```
$  vi  mxm.cpp
       あるいは
$  cp –p ${PACKAGE_DIR}/doc/src_tutorial/mxm.cpp  ${MY_DIR}/
```

行列積ソースプログラム例 C++

```
#include <stdio.h>
#include <string.h>
#include "matrix.h"
void init2d();
void mxm2d();

int main()
{
    init2d();
    mxm2d();
    return 0;
}
```

主プログラム部分

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] = (double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            matrix.c2d[j][i] = c1;
        }
    }
}
```

ヘッダーファイル matrix.h　の内容

```
#define MATSIZE 1000
struct matrix {
    double a2d[MATSIZE][MATSIZE];
    double b2d[MATSIZE][MATSIZE];
    double c2d[MATSIZE][MATSIZE];
    int nsize;
} matrix;
```

例題プログラムへのPMlibの追加

- 元の主プログラム部分

```
int main()
{

    init2d();

    mxm2d();

    return 0;
}
```

ヘッダー

初期設定

測定区間1

測定区間2

レポートを出力

- PMlibを追加した主プログラム部分

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main()
{
PM.initialize();
PM.setProperties("A:init2d", PM.CALC);
PM.setProperties("B:mxm2d", PM.CALC);
PM.start("A:init2d");
init2d();
PM.stop ("A:init2d");
PM.start("B:mxm2d");
mxm2d();
PM.stop ("B:mxm2d");
PM.print(stdout, "", "");
PM.printDetail(stdout);
return 0;
}
```

ヘッダー追加

初期設定

測定区間1

測定区間2

レポートを出力

例題プログラムのコンパイル　FX100・京コンピュータ

- 例題プログラムをコンパイルしてPMlibをリンクするスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/K/x.myprog.sh
- スクリプトの内容を確認する
  - スクリプトはログインノード上での対話的なコンパイルを想定している。
  - MY_DIR, INTEL_DIR, PMLIB_DIR, PAPI_DIRのパス
  - PMlibはMPI版(-lPMmpi)か1プロセス版 (-lPM) かを選択してリンクする。
  - この例題プログラムは1プロセス版なので -lPMを選択している
- スクリプトを実行する。

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.myprog.sh
```

- コンパイルが終了すると${MY_DIR}に実行プログラム a.out が生成される

```
$  file ./a.out
./a.out: ELF 64-bit MSB executable, SPARC V9, total store ordering, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.12, not stripped
```

例題プログラムの実行　FX100・京コンピュータ

- 対話的ジョブセッションの開始
  - 京やFX100はバッチジョブ管理されているので、計算ノード1台を利用する対話的ジョブセッションを起動する。
  - FX100ではシステム毎にpjsub のオプションが異なる。適宜対応。

```
$  pjsub --interact --rsc-list "elapse=01:00:00" --rsc-list "node=1"
[INFO] PJM 0000 pjsub Job 2955440 submitted.
[INFO] PJM 0081 ....connected.
[INFO] PJM 0082 pjsub Interactive job 2955440 started.
```

- 計算ノード上で対話的ジョブセッションが始まったら、最初に環境設定を行う
  - FX100ではシステム毎に環境設定方法が異なることがある。適宜対応。

```
$  source  /work/system/Env_base              # 京の場合
$  /opt/FJSVXosPA/bin/xospastop               # 京の場合
$  MY_DIR=${HOME}/pmlib/mysrc
$  cd ${MY_DIR}
```

例題プログラムの実行　FX100・京コンピュータ

- 計算ノード上でプログラムを実行する。
  - デフォルトではユーザー申告モードで測定される
  - 標準出力に基本レポート・詳細レポートが出力される事を確認

```
$ ./a.out
```

- 環境変数HWPC_CHOOSERにFLOPSを指定して再度実行する
  - HWPCによる自動測定モード（計算量の自動測定）
  - 基本レポート・詳細レポートを確認

```
$  export HWPC_CHOOSER=FLOPS
$  ./a.out
```

例題プログラムへのOpenMP指示行の追加

- ソースプログラムにOpenMP指示行を追加

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] =
(double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j,k,c1)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            ...以下略...
```

例題プログラムの再実行　FX100・京コンピュータ

- ログインノード上で再度コンパイル

```
$ ${PACKAGE_DIR}/doc/scripts/Intel/x.myprog.sh
```

- 計算ノード上で対話的ジョブセッションを継続・再開
  - もし先に開始したジョブセッションが終了していればジョブの再投入と環境設定を再実施
  - 測定条件（環境変数）を追加・変更して出力結果を比較する
    - OMP_NUM_THREADS=1/2/4/8など：OpenMPスレッド数
    - HWPC_CHOOSER=FLOPS/BANDWIDTH/など：HWPC測定のタイプ

```
$ export HWPC_CHOOSER=FLOPS
$ export OMP_NUM_THREADS=4
$ ./a.out
```

10分+でできるPMlibのインストールと利用😚

Apple Mac編
（OSX10.11以降）

PMlibのインストールと利用実行

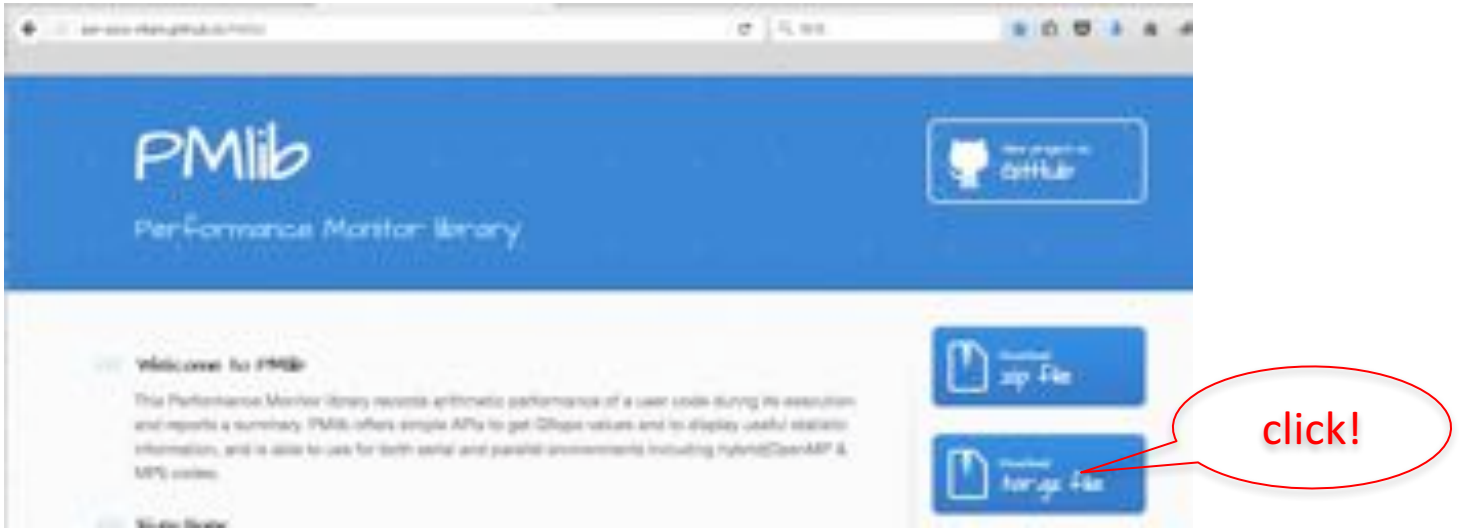
- PMlibのインストール
  - PMlibパッケージの入手
  - PMlibのインストール

- PMlibの利用実行
  - 例題プログラムの作成（C++言語で作成）
  - 例題プログラムへのPMlibの追加（ソースプログラムの編集）
  - 例題プログラムをコンパイルしてPMlibをリンクする
  - 例題プログラムを実行して、PMlibのレポートを確認する

- （注）Apple Mac環境ではCコンパイラ（Clang）は標準で備わっているがFortranコンパイラ、MPIライブラリは通常利用者がインストールする。この説明資料ではCコンパイラは標準のclang、FortranコンパイラはGNU Fortran（gfortran:GNU Cに付属）、MPIはOpenMPIがインストールされていることを前提として書かれている。
- （注）Apple Mac環境ではPAPIライブラリはサポートされていない。

PMlibパッケージの入手（共通）

- パッケージ公開リポジトリ
  - http://avr-aics-riken.github.io/PMlib/



- ダウンロードしたファイル名は avr-aics-riken-PMlib-*.tar.gz
  - （*の部分はバージョンにより変わる）
- ダウンロードしたファイルをインストール先のコンピュータに転送する。
  - 以降の例では ${HOME}/tmp/　下に転送したと仮定している

PMlibパッケージの展開（共通）

- インストール先のコンピュータ上で、転送したパッケージを展開する
- 展開したディレクトリにシンボリックリンクと、パスの環境変数を設定する。
- 以下の例ではログイン後ホームに pmlib ディレクトリを作って、その下に転送したパッケージのファイルを展開する。

```
$  mkdir ~/pmlib
$  cd ~/pmlib
$  tar –zxf  ~/tmp/avr-aics-riken-PMlib-*.tar.gz
$  ls –go
drwxr-xr-x  9 4096 2017-06-22 14:31 avr-aics-riken-PMlib-073c316
$  mv  avr-aics-riken-PMlib-*  package
$  export  PACKAGE_DIR=~/pmlib/package
```

- Intelサーバ/Intel環境, Intelサーバ/GNU環境、FX100システム、京コンピュータ、Macbookの各システムに対応したインストールスクリプトが以下に提供されている。

```
$ ls  –F  package/doc/scripts/
GNU/       Intel/     Kcomputer/ Mac/       fx100/
```

PMlibのインストール　Apple Mac環境

- Apple Mac環境用のインストールスクリプト例は以下に提供されている

```
$  vi  ${PACKAGE_DIR}/doc/scripts/Mac/x.cmake-Mac-clang.sh
```

- コンパイラ・ライブラリはシステムによってインストールされているパスが異なるので、スクリプトで設定されているパスが正しいか確認し、必要であれば修正する。

```
N行目　　PACKAGE_DIR=~/pmlib/package
N行目　　PMLIB_DIR=~/pmlib/usr_local_pmlib/intel
N行目　　MPI_DIR=/usr/local/openmpi/openmpi-1.10.2-clang
N行目　　export PATH=${PATH}:${MPI_DIR}/bin
```

- PACKAGE_DIR # PMlibパッケージを展開したディレクトリ（必須：既存）
- PMLIB_DIR　　# PMlibのインストール先ディレクトリ（必須：新規・更新）
- MPI_DIR　　# MPIディレクトリ（必須：既存）

PMlibのインストール　Apple Mac環境

- インストールスクリプトを実行する。

```
$  ${PACKAGE_DIR}/doc/scripts/Mac/x.cmake-Mac-clang.sh 2>&1 |¥
       tee stdout.make
```

- 以下のファイルがインストールされた事を確認する

```
$  ls –go ${PMLIB_DIR}
drwxr-xr-x 2 140  7月   4 16:20 doc
drwxr-xr-x 2 117  7月   4 16:20 include
drwxr-xr-x 2  57  7月   4 16:20 lib
drwxr-xr-x 2  80  7月   4 16:20 share

$ ls –go ${PMLIB_DIR}/lib
-rw-r--r-- 1 170276  7月   4 16:20 libPM.a          # 1プロセス版PMlibライブラリ
-rw-r--r-- 1 175460  7月   4 16:19 libPMmpi.a       # MPI版PMlibライブラリ
```

- 以上でPMlibインストール終了！
- では続いてPMlibを使ってみよう。

例題プログラムの作成（C++版）

- 適当なディレクトリ($MY_DIRとする)の下にプログラム mxm.cpp を作成する

```
$  MY_DIR=~/pmlib/my_dir
$  mkdir  -p ${MY_DIR}
$  cd  ${MY_DIR}
```

- N次の正方行列の積を計算するプログラム
  - 主プログラム：関数1と関数2を呼び出して行列積の計算を行う。
  - 関数1：正方行列[A]、[B]の各要素を値1.0で初期化する
  - 関数2：行列積[C]=[A][B]を計算する
  - シリアルプログラム（MPI不要、OpenMP不要）
- 手早く進みたい場合はパッケージのプログラム例をコピーすると良い
- 自分でソースを書いてももちろん良い　vi　mxm.cpp

```
$  vi  mxm.cpp
       あるいは
$  cp –p ${PACKAGE_DIR}/doc/src_tutorial/mxm.cpp  ${MY_DIR}/
```

行列積ソースプログラム例 C++

```
#include <stdio.h>
#include <string.h>
#include "matrix.h"
void init2d();
void mxm2d();

int main()
{
    init2d();
    mxm2d();
    return 0;
}
```

主プログラム部分

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] = (double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            matrix.c2d[j][i] = c1;
        }
    }
}
```

ヘッダーファイル matrix.h　の内容

```
#define MATSIZE 1000
struct matrix {
    double a2d[MATSIZE][MATSIZE];
    double b2d[MATSIZE][MATSIZE];
    double c2d[MATSIZE][MATSIZE];
    int nsize;
} matrix;
```

例題プログラムへのPMlibの追加

- 元の主プログラム部分

```
int main()
{

    init2d();

    mxm2d();

    return 0;
}
```

ヘッダー

初期設定

測定区間1

測定区間2

レポートを出力

- PMlibを追加した主プログラム部分

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main()
{
PM.initialize();
PM.setProperties("A:init2d", PM.CALC);
PM.setProperties("B:mxm2d", PM.CALC);
PM.start("A:init2d");
init2d();
PM.stop ("A:init2d");
PM.start("B:mxm2d");
mxm2d();
PM.stop ("B:mxm2d");
PM.print(stdout, "", "");
PM.printDetail(stdout);
return 0;
}
```

ヘッダー追加

初期設定

測定区間1

測定区間2

レポートを出力

例題プログラムのコンパイル・実行　Mac−clang環境

- 例題プログラムをコンパイルしてPMlibをリンクするジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/Mac/x.myprog.sh
- スクリプトの内容を確認する
  - MY_DIR, PMLIB_DIR のパス。（OTFを利用する場合はOTF_DIRも）
  - PMlibはMPI版(−lPMmpi)か1プロセス版 (−lPM) かを選択してリンクする。
  - この例題プログラムは1プロセス版なので −lPMを選択している
- ジョブスクリプトを実行する。
  - スクリプトでは対話的なコンパイル・実行を想定している。

```
$  ${PACKAGE_DIR}/doc/scripts/Mac/x.myprog.sh
```

- 実行結果の標準出力でPMlibの基本レポート・詳細レポートを確認

例題プログラムへのOpenMP指示行の追加

- ソースプログラムにOpenMP指示行を追加

```
void init2d()
{
    int i, j, nsize;
    matrix.nsize = MATSIZE;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
        matrix.a2d[i][j] =
(double)(i+j)/(double)nsize;
        matrix.b2d[i][j] = 1.0-matrix.a2d[i][j];
        matrix.c2d[i][j] = 0.0;
        }
    }
}
```

```
void mxm2d()
{
    int i, j, k, nsize;
    double c1;
    nsize = matrix.nsize;
    #pragma omp parallel for private(i,j,k,c1)
    for (i=0; i<nsize; i++){
        for (j=0; j<nsize; j++){
            c1 = 0.0;
            for (k=0; k<nsize; k++){
            c1 = c1 +
                matrix.a2d[k][i] * matrix.b2d[j][k];
            }
            ...以下略...
```

例題プログラムのコンパイルと実行　Apple Mac環境

- 再コンパイル
- 環境変数を追加・変更して再実行
  - ジョブスクリプトで測定条件を変更して出力結果を比較する
  - OMP_NUM_THREADS=1/2/4/8など：OpenMPスレッド数

```
$  ${PACKAGE_DIR}/doc/scripts/Mac/x.myprog.sh
```

以上です。

PMlibの基本的な機能と利用方法についてご紹介しました。
さらに詳細な機能やその利用方法について知りたい方は、PMlibパッケージの以下の資料をご覧ください。

doc/PMlib.pdf	PMlib利用説明書
doc/tutorial/Tutorial-slide1-overview.pdf	講習会用資料1
doc/tutorial/Tutorial-slide2-installation.pdf	講習会用資料2
doc/tutorial/PMlib-Getting-Started.pdf	簡易版利用ガイド（本資料）