10分でわかるPMlib 😚

理化学研究所 計算科学研究センター
三上和徳
2020年8月19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。
    - 富岳
    - 富士通FX100
    - Intel Xeonサーバ・PC
    - Apple Macbook（注　HWPC機能は利用不可）
  - 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
  - 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
  - ユーザが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

- 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（省略可能）
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

各APIの引数仕様についての詳しい説明はdoc/以下のファイルを参照してください

- doc/Readme.md　　Doxygenファイル生成方法の説明
- doc/html/index.html　　Doxygenが生成するWeb資料

PMlib機能API

機能API: 通常は呼び出す必要はないが、用途に応じて利用することができる。

関数名(C++)	関数名(Fortran)	機能	呼び出し位置と回数	引数
setProperties()	f_pm_setproperties()	測定区間の属性設定	start()よりも前・各区間一回	(1)ラベル、(2)測定対象タイプ、(3)排他指定
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
printThreads()	f_pm_printthreads()	指定プロセスのスレッド毎詳細情報の表示	測定終了時・一回	(1)出力ファイルポインタ、(2)指定プロセスのMPIランク番号、(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ランク毎の詳細情報表示	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 ()	f_pm_printcomm ()	MPI_Comm_split()で作成したグループ毎に詳細情報表示	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
printProgress()	f_pm_printprogress()	測定中の経過情報をスナップショット表示	任意・制約なし	(1)出力ファイルポインタ、(2)任意のコメント、(3)区間の表示順序指定
postTrace ()	f_pm_posttrace ()	ポスト処理用traceファイルの出力	測定終了後・一回	引数なし。環境変数あり。
reset()	f_pm_reset()	測定区間の経過情報をリセットする	任意・制約なし	(1)ラベル
resetAll()	f_pm_resetall()	全測定区間の経過情報をリセットする	任意・制約なし	引数なし。

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_start (”label")
call mykernel (msize,n,a,b,c)
call f_pm_stop (”label", fops, ncall)
call f_pm_print ("", isort)
end
```

PMlib初期化

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.start(”label");
mykernel();
PM.stop (”label”,fops,ncall);
PM.print(stdout, "", "");
return 0;
}
```

PMlibヘッダー

PMlib初期化

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、（オプション）詳細レポート
  - 各MPIプロセス毎のプロファイルを出力
  - 各MPIプロセス毎のHWPCイベント統計量　イベント種類を環境変数で指定
  - 指定プロセス番号のOpenMPスレッド毎のHWPCイベント統計量
- 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 7.0.1
    Linked PMlib supports: MPI, OpenMP, HWPC, OTF
    Host name : uv03
    Date      : 2020/07/27 : 22:12:15
    Mrs. Kobe
    Parallel Mode:   Hybrid (2 processes x 4 threads)
    HWPC_CHOOSER=FLOPS environment variable is provided.

    PMlib enabled elapse time (from initialize to print) = 4.465e+00 [sec]
    Exclusive sections and inclusive sections are reported below.
    Inclusive sections, marked with (*), are not added in the statistics total.

    Section         |    call  |        accumulated time[sec]           | hardware counted floating point ops.
    Label           |          |      avr    avr[%]     sdv    avr/call  |  f.p.ops      sdv    performance
    ----------------+----------+----------------------------------------+---------------------------
    Loop-section(*) :        1   4.161e+00  93.19  1.26e-01  4.161e+00    3.657e+10  2.54e+07   8.79 Gflops(*)
    Kernel-Slow     :        3   3.032e+00  67.90  9.91e-02  1.011e+00    1.223e+10  2.75e+05   4.03 Gflops
    Kernel-Fast     :        3   7.087e-01  15.87  2.72e-02  2.362e-01    1.522e+10  2.96e+07  21.47 Gflops
    Initial-section :        1   6.069e-03   0.14  9.82e-05  6.069e-03    5.833e+07  9.39e+03   9.61 Gflops
    ----------------+----------+----------------------------------------+---------------------------
    Sections per process         3.746e+00     -Exclusive HWPC sections-  2.750e+10             7.34 Gflops
    ----------------+----------+----------------------------------------+---------------------------
    Sections total job           3.746e+00     -Exclusive HWPC sections-  5.501e+10            14.68 Gflops
```

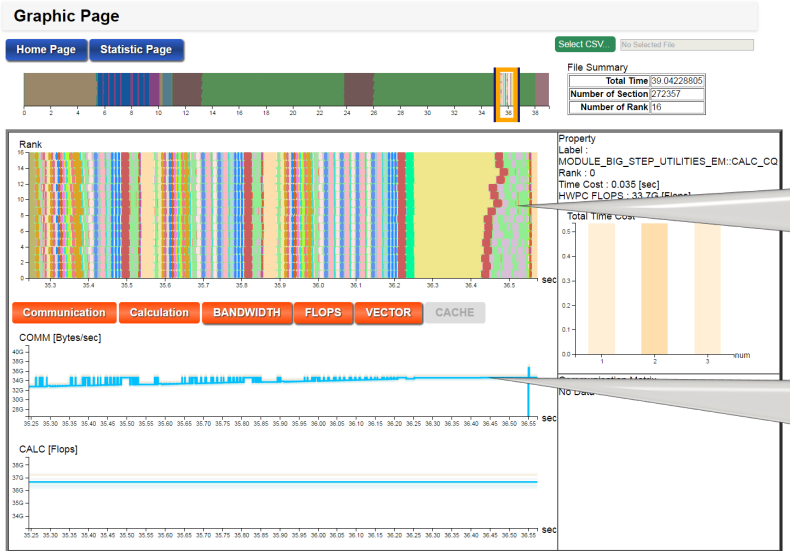
基本レポート例 (ユーザ申告モード)

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

   Timing Statistics Report from PMlib version 7.0.1
   Linked PMlib supports: MPI, OpenMP, HWPC, OTF
   Host name : uv03
   Date      : 2020/07/27 : 22:12:10
   Mrs. Kobe
   Parallel Mode:   Hybrid (2 processes x 4 threads)
   HWPC_CHOOSER is not set. User API values are reported.

   PMlib enabled elapse time (from initialize to print) = 4.656e+00 [sec]
   Exclusive sections and inclusive sections are reported below.
   Inclusive sections, marked with (*), are not added in the statistics total.

   Section         |    call  |        accumulated time[sec]           | user defined numerical performance
   Label           |          |      avr    avr[%]     sdv    avr/call  |  operations   sdv    performance
   ----------------+----------+----------------------------------------+---------------------------
   Loop-section(*) :        1   4.339e+00  93.18  3.78e-01  4.339e+00    1.920e+11  0.00e+00  44.25 GB/sec(*)
   Kernel-Slow     :        3   3.159e+00  67.86  2.83e-01  1.053e+00    1.200e+10  0.00e+00   3.80 Gflops
   Kernel-Fast     :        3   7.395e-01  15.88  7.09e-02  2.465e-01    1.200e+10  0.00e+00  16.23 Gflops
   Initial-section :        1   7.348e-03   0.16  3.09e-05  7.348e-03    2.000e+06  0.00e+00 272.17 Mflops
   ----------------+----------+----------------------------------------+---------------------------
   Sections per process         3.906e+00     -Exclusive CALC sections-  2.400e+10             6.14 Gflops
   ----------------+----------+----------------------------------------+---------------------------
   Sections total job           3.906e+00     -Exclusive CALC sections-  4.800e+10            12.29 Gflops
```

出力のオプション

- 詳細プロファイル（テキストレポート出力）
  - プロセス毎・スレッド毎の計算量・システムの実効性能諸値
  - アプリケーションと一体で利用しやすい

- OTF(Open Trace Format)ファイル出力
  - 専用のポスト処理プログラムTRAiLを用いてWebブラウザで表示
  - 計算性能を時刻歴に可視化する機能



ランク毎処理時刻グラフ
横軸：時刻
縦軸：ランク番号
実行されているラベルを
前面に表示

性能グラフ
横軸：時刻
縦軸：性能測定値
ランク毎に1つの曲線を表示

10分でできるPMlibのインストールと利用😚

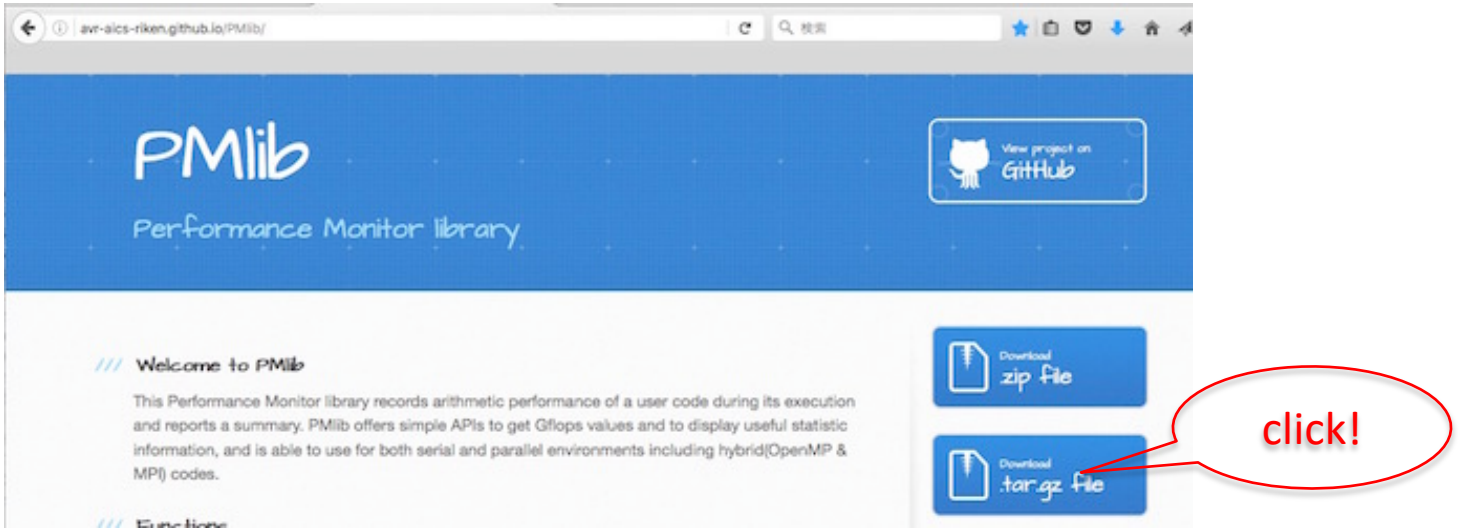
理化学研究所 計算科学研究センター
三上和徳
2020年8月19日

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のインストール例はコンピュータシステム毎に内容が異なります。適切なものを選んでお読みください。
    - Intel環境編・富岳編・FX100編・Mac編
- 例題プログラムでのPMlibの利用
  - 例題プログラムの準備（C++とFortranの例を用いる）
  - 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
    - コンパイル・実行例はコンピュータシステム毎に内容が異なります。適切なものを選んでお読みください。
    - Intel環境編・富岳編・FX100編・Mac編

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する

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     252 2020-07-15 17:42 avr-aics-riken-PMlib-e93d078
$  mv  avr-aics-riken-PMlib-*  package
$  export  PACKAGE_DIR=~/pmlib/package
```

- 各システムに対応したインストール用スクリプトが以下に含まれている。

```
$ ls  –F  package/doc/scripts/
Fugaku/    GNU/       Intel/     Kcomputer/ Mac/       fx100/     fx700/
```

PMlibのインストール

- PMlibのインストール
  - PMlibのインストール例はコンピュータシステム毎に内容が異なります。適切なものを選んでお読みください。
    - PMlibのインストール例　Intel環境編
    - PMlibのインストール例　富岳編
    - PMlibのインストール例　FX100編
    - PMlibのインストール例　Mac・OSX編

▫ PMlibのインストール　Intel環境編

- Intelコンパイラ＋Intel MPIのソフトウエア環境を想定している。
- 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
```

  - Intelコンパイラ・ライブラリを利用するための設定はシステムによって異なり、moduleの指定や、ディレクトリの直接指定などを適切に行う必要がある。上記スクリプト冒頭の２～１４行にいくつかのシステムにおける設定例をコメントで記している。
  - PMlibをインストールするためのディレクトリを指定する

```
21行目　PACKAGE_DIR=~/pmlib/package
22行目　PMLIB_DIR=~/pmlib/usr_local_pmlib/pmlib-intel
25行目　PAPI_DIR=/usr/local/papi/papi-5.6.0-intel
```

  - PACKAGE_DIR : PMlibパッケージを展開したディレクトリ（前ページで作成した）
  - PMLIB_DIR : PMlibのインストール先ディレクトリ（新規に作成・更新することになる）
  - 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”）

    PAPIはバージョン5.6.0以降を用いると良い。システムが古いPAPIしか装備していない場合は、バージョン5.6.0以降のパッケージを個別にインストールすると良い。

- 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
```

▫ PMlibのインストール　富岳編

- スーパーコンピュータ富岳用のインストールスクリプトを確認する

```
$  vi  ${PACKAGE_DIR}/doc/scripts/Fugaku/x.cmake-fugaku.sh
```

  - スーパーコンピュータ富岳用のコンパイラ・ライブラリパスはシステムが自動認識する。
  - PMlibをインストールするためのディレクトリを指定する

```
7行目     PACKAGE_DIR=${HOME}/pmlib/package
8行目     PMLIB_DIR=${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（前ページで作成した）
  - PMLIB_DIR : PMlibのインストール先ディレクトリ（新規に作成・更新することになる）
  - PAPI_DIR : PAPIライブラリ（オプション：既存、”yes” で自動認識、または”no” ）

- 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
```

▫ PMlibのインストール　FX700/FX1000編

- FX700/FX1000共通のインストールスクリプトを確認する

```
$ vi ${PACKAGE_DIR}/doc/scripts/fx700/x.cmake-fx700.sh
```

  - FX700/FX1000用のコンパイラ・ライブラリパスはシステムにより異なる場合があるが、通常は自動認識され、ユーザーが指定する必要はない(5行目 PATH変数、6行目 LD_LIBRARY_PATH変数)。
  - PMlibをインストールするためのディレクトリを指定する

```
11行目　PACKAGE_DIR=${HOME}/pmlib/package
12行目　PMLIB_DIR=${HOME}/pmlib/usr_local_pmlib/fx700
16行目　PAPI_DIR=/opt/FJSVxos/devkit/aarch64/rfs/usr
```

  - PACKAGE_DIR : PMlibパッケージを展開したディレクトリ(前ページで作成した)
  - PMLIB_DIR : PMlibのインストール先ディレクトリ(新規に作成・更新することになる)
  - PAPI_DIR : PAPIライブラリ(オプション:既存。ディレクトリはシステムにより異なる場合がある)

- 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/ fx700/x.cmake-fx700.sh 2>&1 | tee stdout.make
```

▫ PMlibのインストール　FX100・京コンピュータ編

- FX100・京コンピュータ共通のインストールスクリプトを確認する。

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

  - FX100・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。
  - PMlibをインストールするためのディレクトリを指定する

```
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（前ページで作成した）
  - PMLIB_DIR : PMlibのインストール先ディレクトリ（新規に作成・更新することになる）
  - PAPI_DIR : PAPIライブラリ（オプション：既存、”yes” で自動認識、または”no” ）

- 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/ Kcomputer /x.cmake-K-all.sh 2>&1 | tee stdout.make
```

▫ PMlibのインストール　Apple Mac環境編

- 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必要がある。ここではGNU Fortranコンパイラ（gfortran）、MPIはOpenMPIがインストールされていると仮定する。
- 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
```

  - PMlibをインストールするためのディレクトリを指定する

```
8行目      PACKAGE_DIR=~/pmlib/package
9行目      PMLIB_DIR=~/pmlib/usr_local_pmlib/intel
```

  - PACKAGE_DIR : PMlibパッケージを展開したディレクトリ（前ページで作成した）
  - PMLIB_DIR : PMlibのインストール先ディレクトリ（新規に作成・更新することになる）
  - Apple Mac環境ではPAPIライブラリが無く、HWPC機能はサポートされない。

- 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
```

PMlibインストール終了時のファイル （共通）

- 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          # PMlibライブラリ非MPI版
-rw-r--r-- 1 175460  7月  4 16:19 libPMmpi.a       # PMlibライブラリMPI対応版
-rw-r--r-- 1   5470  7月  4 16:20 libpapi_ext.a    # PAPI拡張ライブラリ
```

Libpapi_ext.aファイルはPAPIオプションを
指定した場合のみ作成される。

- 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。

例題プログラムの準備

- 適当なディレクトリ($MY_DIRとする)の下にプログラムを作成する
- パッケージのdoc/src_tutorialディレクトリ下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

```
$  MY_DIR=${HOME}/pmlib/my_dir
$  mkdir  -p ${MY_DIR}
$  cd  ${MY_DIR}
$  cp –p ${PACKAGE_DIR}/doc/src_tutorial/mxm.*    ./
```

- 次ページ以降ではC++(mxm.cpp)かFortran(mxm.f90)のどちらかを選んで進めると良い
- mxm.cpp, mxm.f90の内容：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）

例題C++プログラム（mxm.cpp）

```
#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C++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;
}
```

- 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();
PMPM.start("A:init2d");
init2d();
PM.stop ("A:init2d");
PM.start("B:mxm2d");
mxm2d();
PM.stop ("B:mxm2d");
PM.print(stdout, "", "");
return 0;
}
```

ヘッダー追加

PMlib初期化

測定区間1

測定区間2

レポート出力

例題Fortranプログラム（mxm.f90）

主プログラム部分

```
program main
    parameter(m=1000)
    real(kind=8), allocatable :: a(:,:),b(:,:),c(:,:)
    allocate (a(m,m), b(m,m), c(m,m), stat=istat)
    n=m

    call subinit2d (m,n,a,b,c)

    call submxm2d (m,n,dflop,a,b,c)

    write(*,*)  'something was computed', c(m,m)
    stop
end
```

```
subroutine subinit2d (m,n,a,b,c)
    real(kind=8) :: a(m,m), b(m,m), c(m,m)
    do j=1, n
    do i=1, n
    a(i,j)=sin(real(i)/real(n))
    b(i,j)=cos(real(i)/real(n))
    end do
    end do
    return
end
```

```
subroutine submxm2d (m,n,dflop,a,b,c)
    real(kind=8) :: a(m,m), b(m,m), c(m,m)
    real(kind=8) :: x

    dflop=2.0*dble(n)**3
    do j=1, n
    do i=1, n
    x=0
    do k=1, n
    x=x+a(i,k)*b(k,j)
    end do
    c(i,j) = x
    end do
    end do
    return
end
```

例題Fortranプログラムへの PMlibの追加

- 元の主プログラム部分

```
call subinit2d (m,n,a,b,c)

call submxm2d (m,n,dflop, a,b,c)
```

- PMlibを追加した主プログラム部分

```
call f_pm_initialize (0)
call f_pm_start ("A:subinit2d")
call subinit2d (m,n,a,b,c)
call f_pm_stop ("A:subinit2d", 0.0, 0)

call f_pm_start ("B:submxm2d")
call submxm2d (m,n,dflop,a,b,c)
call f_pm_stop ("B:submxm2d", 0.0, 0)

call f_pm_print ("","","",0)
```

PMlib初期化

測定区間1

測定区間2

レポート出力

例題プログラムのコンパイル・リンク・実行

- 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
  - コンパイル・実行例はコンピュータシステム毎に内容が異なります。適切なものを選んでお読みください。
    - Intel環境編
    - 富岳編
    - FX100編
    - Mac・OSX編

コンパイル・リンク・実行　Intel環境編

- 例題プログラムをコンパイル、PMlibをリンクし、実行するジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/Intel/x.mxm-fort.sh　　# Fortran例題
    ${PACKAGE_DIR}/doc/scripts/Intel/x.mxm-cpp.sh　　# C++例題
- スクリプトの内容を確認する
  - ジョブスクリプト例は対話的にコンパイル・実行する場合の例である。もしバッチ実行する場合はジョブ指示行を適宜追加して投入する。
  - PMlibインストール時と同様な設定がされていることを確認する
    - PMLIB_DIR, PAPI_DIR、およびINTELコンパイラ・ライブラリ利用の設定
  - PMlibはMPI版(-lPMmpi)か非MPI版 (-lPM) かを選択してリンクする。
    - この例題プログラムは1プロセスの非MPI版なので -lPMを選択する
- ジョブスクリプトを対話的に実行する。バッチ実行する場合は上記の指示行追加後投入。
  - 最初は環境変数HWPC_CHOOSERを指定しないユーザー申告モードで実行

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.mxm-cpp.sh　# C++例題の対話的実行
```

  - 次にスクリプトを編集してHWPC_CHOOSERを指定して実行し(自動測定モード)、PMlibレポートの内容を確認する

```
最下行-1　　export HWPC_CHOOSER=FLOPS
```

```
$  ${PACKAGE_DIR}/doc/scripts/Intel/x.mxm-cpp.sh　# C++例題の対話的実行
```

コンパイル・リンク・実行　富岳編

- 例題プログラムをコンパイル、PMlibをリンクし、実行するジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/Fugaku/x.mxm-fort.sh　# Fortran例題
    ${PACKAGE_DIR}/doc/scripts/Fugaku/x.mxm-cpp.sh　# C++例題
- スクリプトの内容を確認する
  - スクリプト例はバッチジョブでコンパイル・実行する例である。
  - PMlibインストール時と同様にPMLIB_DIR, PAPI_DIRが設定されていることを確認する
  - PMlibはMPI版(-lPMmpi)か非MPI版 (-lPM) かを選択してリンクする。
    - この例題プログラムは1プロセスの非MPI版なので -lPMを選択する
- スクリプトをジョブ投入する。
  - 最初は環境変数HWPC_CHOOSERを指定しないユーザー申告モードで実行

```
$ pjsub ${PACKAGE_DIR}/doc/scripts/Fugaku/x.mxm-cpp.sh  # C++例題の場合
```

  - 次にスクリプトを編集してHWPC_CHOOSERを指定して実行し(自動測定モード)、PMlibレポートの内容を確認する

最下行-1　　export HWPC_CHOOSER=FLOPS

```
$ pjsub ${PACKAGE_DIR}/doc/scripts/Fugaku/x.mxm-cpp.sh  # C++例題の場合
```

コンパイル・リンク・実行　FX100編

- 例題プログラムをコンパイル、PMlibをリンクし、実行するジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/fx100/x.mxm-fort.sh　# Fortran例題
    ${PACKAGE_DIR}/doc/scripts/fx100/x.mxm-cpp.sh　# C++例題
- スクリプトの内容を確認する
  - スクリプト例はバッチジョブでコンパイル・実行する例である。FX100ではシステム毎にジョブ指示行など必要な環境設定方法が異なり、適宜編集する必要がある。
  - PMlibインストール時と同様にPMLIB_DIR, PAPI_DIRが設定されていることを確認する
  - PMlibはMPI版(-lPMmpi)か非MPI版 (-lPM) かを選択してリンクする。
    - この例題プログラムは1プロセスの非MPI版なので -lPMを選択する
- スクリプトをジョブ投入する。
  - 最初は環境変数HWPC_CHOOSERを指定しないユーザー申告モードで実行

```
$ pjsub ${PACKAGE_DIR}/doc/scripts/fx100/x.mxm-cpp.sh　# C++例題の場合
```

  - 次にスクリプトを編集してHWPC_CHOOSERを指定して実行し(自動測定モード)、PMlibレポートの内容を確認する

最下行-1　export HWPC_CHOOSER=FLOPS

```
$ pjsub ${PACKAGE_DIR}/doc/scripts/fx100/x.mxm-cpp.sh　# C++例題の場合
```

コンパイル・リンク・実行　Mac編

- 例題プログラムをコンパイル、PMlibをリンクし、実行するジョブスクリプト例
  - ${PACKAGE_DIR}/doc/scripts/Mac/x.mxm-fort.sh　# Fortran例題
    ${PACKAGE_DIR}/doc/scripts/Mac/x.mxm-cpp.sh　# C++例題
- スクリプトの内容を確認する
  - ジョブスクリプト例は対話的にコンパイル・実行する場合の例である。
  - PMlibインストール時と同様な設定がされていることを確認する
    - PMLIB_DIRおよびFortranコンパイラ・ライブラリ利用の設定
  - PMlibはMPI版(-lPMmpi)か非MPI版 (-lPM) かを選択してリンクする。
    - この例題プログラムは1プロセスの非MPI版なので -lPMを選択する
- ジョブスクリプトを対話的に実行する。Macではユーザー申告モードでの実行モードだけがサポートされ、環境変数HWPC_CHOOSERは無効となる。

```
$ ${PACKAGE_DIR}/doc/scripts/Mac/x.mxm-cpp.sh　# C++例題の場合
```

例題プログラムの実行 スレッドレベルの詳細レポート出力

- ソースプログラムの編集
  - 興味ある測定区間にOpenMP指示行を挿入
  - 詳細レポート出力用の APIを挿入
- ジョブスクリプト x.mxm-cpp.sh/x.mxm-fort.shで環境変数を設定してジョブを再実行
  - OMP_NUM_THREADS=1/2/4/8など：OpenMPスレッド数
  - HWPC_CHOOSER=FLOPS/BANDWIDTH/VECTORなど：HWPC測定のタイプ
  - 各ケースのPMlibレポート出力内容を確認する

C++ソースプログラムの編集

- ソースプログラムの測定区間に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...
```

- 主プログラムへPMlib詳細レポート出力APIを追加

```
int main() {
… …
PM.printDetail(stdout);
PM.printThreads(stdout, 0, 0);
return 0;
}
```

詳細レポート出力

Fortranソースプログラムの編集

- ソースプログラムの測定区間にOpenMP指示行を追加

```
subroutine subinit2d (m,n,a,b,c)
   real(kind=8) :: a(m,m), b(m,m), c(m,m)
!$omp parallel do
   do j=1, n
   do i=1, n
   a(i,j)=sin(real(i)/real(n))
   b(i,j)=cos(real(i)/real(n))
   end do
   end do
   return
end
```

```
subroutine submxm2d (m,n,dflop,a,b,c)
   real(kind=8) :: a(m,m), b(m,m), c(m,m)
   real(kind=8) :: x
   dflop=2.0*dble(n)**3
!$omp parallel do private(i,j,k,x)
   do j=1, n
   do i=1, n
   x=0
   do k=1, n
   x=x+a(i,k)*b(k,j)
   end do
...以下略...
```

- 主プログラムへPMlib詳細レポート出力APIを追加

```
program main
… …
call f_pm_printdetail ("",1,0)
call f_pm_printthreads ("",0,0)
end program
```

詳細レポート出力

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Doxygen資料、英語版Readmeファイルなどをご覧ください。

doc/Readme.md	Doxygenファイル生成方法の説明
doc/html/index.html	Doxygenが生成するWeb資料
Readme.md	インストール・利用方法（英語資料）
doc/tutorial/PMlib-Getting-Started.pdf	簡易版利用ガイド（本資料）