PMlib - Performance Monitor Library

利用説明書

Ver. 4.1.4

October 2015

Advanced Visualization Research Team
Advanced Institute for Computational Science
RIKEN
http://aics.riken.jp/
7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

Release

Version 4.1.4	2015-10-27
Version 4.1	2015-10-02
Version 2.2	2014-10-30
Version 1.9.3	2013-06-27
Version 1.9.2	2013-06-26
Version 1.9.1	2013-06-25
Version 1.9	2013-06-14
Version 1.8	2013-06-13
Version 1.7	2013-05-08
Version 1.6	2013-04-13
Version 1.5	2012-12-05
Version 1.4	2012-09-06
Version 1.3	2012-07-23
Version 1.2	2012-07-11
Version 1.1	2012-05-02
Version 1.0	2012-04-28

COPYRIGHT

第 1 章　PMlib とは

1.1　概要

PMlib 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である。オープンソースソフトウエアとして理研 AICS が開発・提供している。アプリケーションのソースプログラム中に PMlib 測定区間を指定して実行し、プログラム実行時に測定区間の実行時間と計算量の集計・統計情報をレポート出力する機能をもつ。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できる。

PMlib は以下の並列プログラムモデルに対応している。

- シリアルプログラム
- OpenMP 並列プログラム* (*ただしスレッド内部からの 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

呼び出しプログラム言語は C++ と Fortran に対応している。いわゆる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している。

Pmlib の動作が確認されているシステムは以下。

- 京コンピュータ/FX10 計算ノード
  - 富士通コンパイラ＋富士通 MPI
- Intel Xeon 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
- Apple Macbook
  - Apple Clang/LLVM コンパイラ＋ OpenMPI
  - Intel コンパイラ＋ OpenMPI
  - GNU コンパイラ

Pmlib のインストールに必要なソフトウエアは以下。

- OS : Linux OS の場合は RedHat 6.1+、Suse 11.3+、kernel 2.6.32+ を推奨
- OS : Apple OS の場合は Macbook: OSX 10.8+
- OS : 富士通 OS の場合はログインノードでのクロスコンパイル環境が上記 Linux OS と同様
- C++, C, Fortran コンパイラ
- HWPC を利用したい場合は PAPI 5.0+

第 2 章　PMlib の機能と測定プロパティ

PMlib の機能は大きく分けて以下の 3 つに分類される。

- 初期化機能
- 計算性能測定機能
- レポート機能

アプリケーションプログラム中で、(1)PMlib 測定区間の初期化・プロパティ設定を行ない、(2) 指定した測定区間を実行し、(3) 終了時に各区間の統計情報をレポート出力させる、という構成で用いる。以下にこれらの機能とそれに伴うプロパティ（設定可能な属性）を示す。

尚、各機能を実現する具体的な関数の仕様は次章で説明される。

2.1　初期化機能

2.1.1　測定用の内部初期化

PMli の内部初期化を実行し、MPI 並列モード・OpenMP 並列モードの自動認識、プラットフォームの認識などを行なう。PMlib には任意の数の測定区間を設定する事が可能で、各測定区間が独立したプロパティを持つ事が可能である。

2.1.2　測定区間のプロパティ

測定する各区間は次のプロパティを持つ。

ラベル　測定区間につける名称（ラベル文字列）
測定計算量のタイプ　「通信」、「演算」、
排他測定フラグ　「排他測定」または「非排他測定」

■ラベル

測定区間はラベル名により識別される。初期化時にデフォルトの測定区間数が設定されるが、プログラム実行時に動的に測定区間を追加する事が可能である。

■測定計算量のタイプ

測定区間には経過時間と測定計算量のボリュームとが積算される。同一測定区間が複数回実行された場合、あるいは同じラベル名が付けられた計測区間群が実行された場合、経過時間と測定計算量はそれらの合計となる。

測定計算量のタイプは「通信」か「演算」かのいずれかであり、通信は CPU・メモリ間、あるいは CPU・CPU 間 (ノード間) のデータ移動（転送）処理を、演算は CPU 内での浮動小数点演算や整数演算を意図する。

備忘録 通信は転送と表現したほうがわかりやすいか?

■排他測定と非排他測定

測定区間は「排他測定」と「非排他測定」とに分類される。排他測定は区間の内部が他の区間とオーバーラップしないことを前提とし、「完成されたプログラムの実行時の性能挙動を把握するために、プログラムコードを排他的な領域に分割して各領域の実行時間を測定する」という使用方法を想定している。排他測定では統計情報出力時に、「全排他測定箇所の合計実行時間」に対する「個々の排他測定箇所の実行時間」の割合も出力する。

一方、非排他測定は区間が他の区間とオーバーラップすることを許し、「プログラムのデバッグあるいはチューニン

グ時に，一時的に興味のある対象領域の実行時間を把握する」ために使用するような場合を想定している。そのため非排他測定区間は排他測定区間を含む場合や、他の非排他測定区間と重ねることも可能で，自由に設置できる。

備忘録 MPI プロセスグループへの動的対応を可能とするため、並列実行時に「排他測定区間は同一回数だけ実行されるものと仮定している」という制約は撤廃した。

2.2　計算性能測定機能

計算性能測定機能は、プログラム実行時に対象となる測定区間の経過時間と計算量を測定し、その統計情報を集計する。計算量を測定する方法には、ユーザーが計算量を明示的に自己申告する方法と、PMlib 内部で HWPC(hardware performance counter) を用いて自動的に取得・算出する方法とがある。

2.2.1　計算量の測定：自己申告モードと自動算出モード

■自己申告モード

ユーザーが計算量を明示的に自己申告する場合は、測定区間の終了を指定する stop() メソッドへの引数としてその計算値を与える。この場合、測定区間のプロパティ登録で設定される測定計算量のタイプにより、計算量の内容 ([通信] か [計算]) が決定されることになる。

「通信」タイプを指定した場合、申告される計算量はバイト単位のデータ移動量であると解釈され、統計情報出力時に通信速度 (Byte/s 単位) が出力される。「演算」タイプを指定した場合、申告される計算量は浮動小数点演算量であると解釈され、統計情報出力時に計算速度 (FLOPS 値) が出力される。

■自動算出モード

計算量を PMlib 内部で自動算出する場合は HWPC の統計情報を用いる。実行するシステムが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 を自動的に呼び出してその統計情報を取得し、計算量を算出する。この様な算出方法を自動算出モードと呼ぶ。

自動算出モードの場合どの様な HWPC イベントグループの値を読み取るかを、プログラム実行時に環境変数 HWPC_CHOOSER で指定する。指定可能なカウンターグループは以下。

- FLOPS : 浮動小数点演算
- BANDWIDTH : バンド幅（メモリ・キャッシュ）
- CACHE : キャッシュ階層のヒット・ミス
- CYCLE : インストラクションサイクル数
- VECTOR : ベクトル命令

■自己申告モードか HWPC 自動算出モードかの決定基準

計算量測定がユーザ申告モードか HWPC 自動算出モードかの判断は、PMlib が内部的に setProperties() の type 引数の値、stop() の fP 引数の値、環境変数 HWPC_CHOOSER の値を用いて、下記表の組み合わせで決定する。

表 2.1　自己申告モードか HWPC 自動算出モードかの決定基準

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
(無指定)	CALC	指定値	時間、fP 引数による Flops	なし
(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 自動算出 Flops	FLOPS に関連する HWPC 統計情報
VECTOR	無視	無視	時間、HWPC 自動算出 SIMD 率	VECTOR に関連する HWPC 統計情報
BANDWIDTH	無視	無視	時間、HWPC 自動算出 Byte/s	BANDWIDTH に関連する HWPC 統計情報
CACHE	無視	無視	時間、HWPC 自動算出 L1, *L*2	CACHE に関連する HWPC 統計情報

計算量測定を自己申告でも HWPC 自動算出モードでも行わない場合は、経過時間だけが集計される。

2.3　レポート機能

測定が必要な区間の計算が終了した時点で統計レポートを出力する事ができる。レポートの出力を指示する箇所は、必ずしもプログラムが終了する直前である必要はない。

第 3 章　PMlib のパッケージ構成とインストール手順

3.1　PMlib 関数の全体構成

PMlib は C++ 言語で書かれたクラスライブラリと C 言語で書かれた HWPC/PAPI インタフェイスおよび Fortran インタフェイスから構成される。主要なクラスは PerfMonitor クラスと PerfWatch クラスの２つであり、ユーザーがプログラムから直接呼び出して利用するのは PerfMonitor クラスである。（下図グレーの部分）PerfWatch クラスは全て PerfMonitor クラスを経由して生成され操作されるため、ユーザーが直接 PerfWatch クラスの関数を呼ぶことはない。

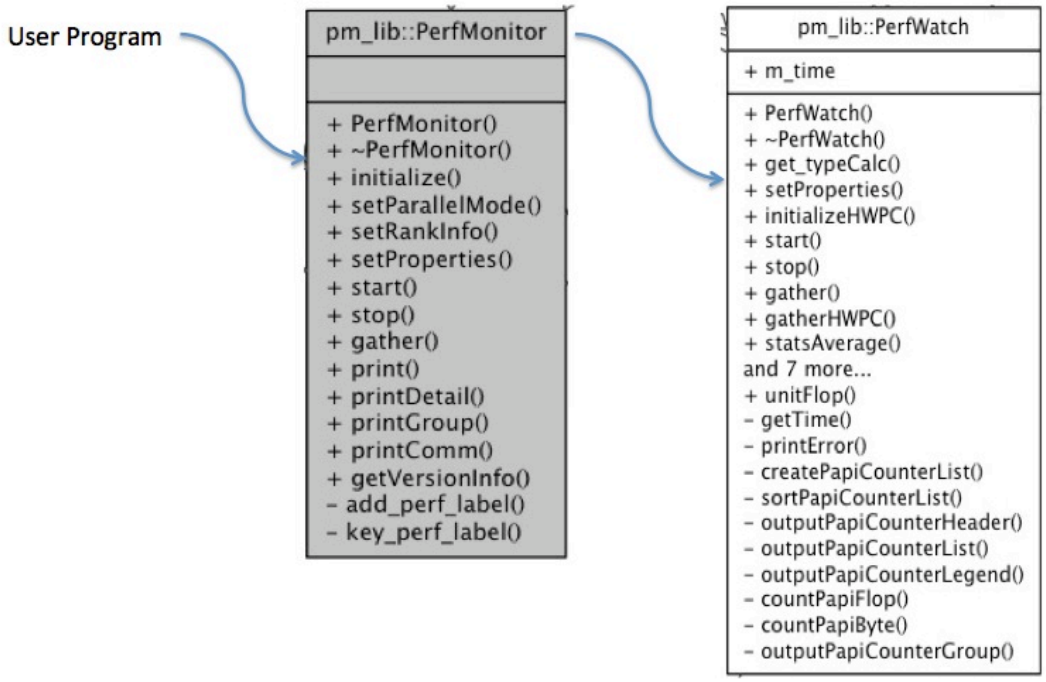

図 3.1　PMlib クラス呼び出し構図

パッケージとしての PMlib のソースプログラム構成は概ね以下である。

- クラスライブラリソースファイル
  src/: PerfCpuType.cpp PerfMonitor.cpp PerfProgFortran.cpp PerfWatch.cpp
- include ファイル
  include/: PerfMonitor.h PerfWatch.h mpi_stubs.h pmVersion.h pmlib_papi.h
- HWPC/PAPI インタフェイスライブラリソースファイル
  src_papi_ext/papi_ext.c

3.2　PMlib パッケージのインストール手順

以下の講習会実習編の資料「PMlib のインストールとテスト」に京コンピュータ、および Intel サーバ上への PMlib のインストール手順、および確認テスト方法が詳しく記載されている。実際のインストールもこの手順に従って実施するのが良い。

https://github.com/mikami3heart/PMlib-tutorials/blob/master/Tutorial-slide2-installation.pdf

本章では京コンピュータおよび Intel Xeon サーバを対象とした PMlib のインストール手順を上記資料から抜粋して基本的なシェルスクリプト形式で示す。

3.2.1　PMlib パッケージの入手方法

PMlib パッケージのソースプログラム一式（tar.gz）は以下の公開レポジトリからダウンロード可能。

http://avr-aics-riken.github.io/PMlib/

ダウンロードされる PMlib パッケージの実際のファイル名は avr-aics-riken-PMlib-バージョン名.tar.gz のように命名された長い名前であるが、以下では PMlib パッケージのファイル名が avr-PMlib.tar.gz であり、ホームディレクトリ直下の $HOME/pmlib/tar_balls/　というサブディレクトリに保存されていると仮定して説明を進める。

3.2.2　京コンピュータへの PMlib インストール

京コンピュータにおいて PMlib を利用するアプリケーション（実行プログラム）は計算ノードで実行されるが、アプリケーションの作成はログインノード上で行われる事を想定して、ログインノード上に PMlib をインストールする方法をスクリプト例で示す。シェルスクリプトファイル（ファイル名）を作成し、ログインノード上で対話的に実行する。

■京コンピュータへの PMlib インストール　スクリプト例

```
K$   cat  x.make-pmlib-K.sh
#! /bin/bash
cd ${HOME}/pmlib
tar -zxf tar_balls/avr-PMlib.tar.gz
ls -go
 # drwxr-xr-x  9 4096 2015-08-24 avr-PMlib
 # drwxr-xr-x  2 4096 2015-08-24 tar_balls
mv avr-PMlib*  PMlib

SRC_DIR=${HOME}/pmlib/PMlib
cd ${SRC_DIR}; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
autoreconf ^^e2^^80^^93i     # 初回のみ autoreconf の実行が必要な場合がある。

cd ${SRC_DIR}/BUILD_DIR; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
# make distclean 2>&1 >/dev/null  # configure と make を何度も実行する場合には奨励
CFLAGS="-std=c99 -Kopenmp,fast -Ntl_notrt"
FCFLAGS="-Cpp -Kopenmp,fast -Ntl_notrt"
CXXFLAGS="-Kopenmp,fast -Ntl_notrt "
INSTALL_DIR=${HOME}/pmlib/install_dir

../configure  CXX=mpiFCCpx  CC=mpifccpx  FC=mpifrtpx \
   CXXFLAGS="${CXXFLAGS}"  CFLAGS="${CFLAGS}"   FCFLAGS="${FCFLAGS}" \
   --with-comp=FJ  --host=sparc64-unknown-linux-gnu \
   --with-papi=yes  --with-example=yes  --prefix=${INSTALL_DIR}

make
make install

K$  ./x.make-pmlib-K.sh
```

3.2.3　Intel Xeon サーバへの PMlib インストール

■Intel コンパイラ・Intel MPI を用いたインストール　スクリプト例

```
$   cat  x.make-intel-impi.sh
#! /bin/bash
module load intel impi papi/intel   # module コマンドについては下記を参照
SRC_DIR=${HOME}/pmlib/PMlib
cd ${SRC_DIR}; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
autoreconf ^^e2^^80^^93i     # 初回のみ autoreconf の実行が必要な場合がある。

cd ${SRC_DIR}/BUILD_DIR; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
CFLAGS="-std=c99 -openmp"
FCFLAGS="-fpp -openmp "
CXXFLAGS="-openmp "
INSTALL_DIR=${HOME}/pmlib/install_develop

../configure \
    CXX=mpiicpc CC=mpiicc FC=mpiifort \
    CFLAGS="${CFLAGS}" CXXFLAGS="${CXXFLAGS}" FCFLAGS="${FCFLAGS}" \
    --with-comp=INTEL  --with-impi=${I_MPI_ROOT} \
    --with-papi=${PAPI_DIR}   --with-example=yes  -prefix=${INSTALL_DIR}

make
make install

$  ./x.make-intel-impi.sh
```

上のスクリプト第２行目の module コマンド行は Intel サーバ上でのコンパイル・リンクを用意に行うためのツールであるが、その詳細説明および module コマンドが使えない場合の対処方法は、後出の 4.3.2 節や、上記 3.2 節に示した講習会資料に説明されているので参照されたい。

3.2.4　その他の資料類

- Doxygen によるオンライン参照ファイル
  PMlib パッケージ doc/ディレクトリで doxygen コマンドを実行すると Web ブラウザで参照可能な html/index.html　などが自動生成される。関数の詳細仕様がドキュメントされている。
- PMlib 説明書（本説明書）
  PMlib パッケージ doc/ディレクトリ以下にも同じ内容のファイル PMlib.pdf が含まれている。

第 4 章　C++ プログラム用 PMlib 関数の仕様と利用例

4.1　C++ プログラム用 PMlib 関数の仕様

ユーザーが C++ プログラムから呼び出し可能な PMlib の関数である、PerfMonitor クラスの仕様を以下に説明する。

以下の説明内容と同じ内容のドキュメントが、PMlib パッケージの doc/ ディレクトリで doxygen コマンドを起動する事により作成される。

C++ API と Fortran API とでは引数の仕様が異なる場合があるので留意する。

■初期化

```
void initialize(int init_nWatch)
```

引数 `init_nWatch` は最初に確保する測定区間数を指定する。(省略可)
測定区間数がおおむね知れている場合はその値を引数として指定することが望ましい。測定区間数が不明、あるいは動的に増加する場合は引数なしで呼び出すことも可能。指定した測定区間数では不足になった時点で PMlib は必要な区間数を動的に増加させる。PerfMonitor 内部では全測定区間数＋ 1 個の PerfWatch クラス (測定時計) をインスタンスする。

関数 `initialize` はプログラム実行時に 1 度だけ呼び出す。

■測定区間の登録とプロパティ設定

```
void setProperties(string& label, Type type, bool exclusiveflag=true)
```

第 1 引数 `label` :ラベル文字列。測定区間はラベル `label` で識別される。ラベル毎に対応したキー番号が内部で自動生成される
第 2 引数 `type` :測定計算量のタイプ (COMM:通信 (省略値)、CALC:計算)
第 3 引数 `exclusiveflag` :排他測定フラグ。bool 型 false: 非排他測定区間、true: 排他測定区間

第 1 引数は必須。第 2 引数は明示的な申告モードの場合には必須、自動算出モードの場合は無視される。第 3 引数は省略可。

測定計算量のタイプ `type` の指定には以下の定数を一つ選んで使用する。

通信　`PerfMonitor::COMM` (内部値は 0)
計算　`PerfMonitor::CALC` (内部値は 1)

関数 `setProperties` は登録する測定区間数分よびだす。

■測定区間の開始

```
void start(string label)
```

第 1 引数 label :ラベル文字列。 測定区間はラベルで識別される。

ラベルで識別される測定区間の始まりを指定する。最初に start() が呼ばれる前に測定区間の登録とプロパティ設定がされていなくてはならない。

■測定区間の終了

```
void stop(string label, double flopPerTask=0.0, unsigned iterationCount=1)
```

第 1 引数 label :ラベル文字列。 測定区間はラベルで識別される。
第 2 引数 flopPerTask :計算量。演算量 (Flop) または通信量 (バイト)
第 3 引数 iterationCount :計算量に乗じる係数。測定区間を複数回実行する場合、その繰り返し数と考えて良い。

ラベル label で識別される測定区間の終わりを指定する。start() から stop() までが 1 区間となる。 1 区間が複数回呼び出される場合に加えて、別の離れた区間に同じラベル名を指定した start() から stop() で測定した場合も同一区間の値として合算される。

第 2 ・第 3 引数で与える計算量は明示的な申告モードでだけ意味を持ち、演算量または通信量が 1 区間 1 回あたりで flopPerTask*iterationCount として算出される。自動算出モードでは無視される。第 2 ・第 3 引数は省略可。

■測定結果の集約

```
void gather()
```

全プロセスの測定結果情報をマスタープロセスに集約する。もし HWPC 情報が取得可能であれば各測定区間の HWPC によるイベントカウンターの統計値を取得する。測定結果の平均値・標準偏差・経過時間降順ソートなどの基礎的な統計情報を計算する。初期化時にスタートさせた全計算時間用測定をストップさせる。

各種レポート出力ルーチンを利用する前に、本関数 gather を 1 度だけ呼び出しておく必要がある。

■レポートの出力：基本統計レポート

```
void print(FILE* fp, string hostname, string comments, int seqSections)
```

第 1 引数 fp :出力ファイルポインタ。標準出力にレポートする場合は stdout を指定。
第 2 引数 hostname :ホスト名。(省略時はマスタープロセスの実行ホスト名)
第 3 引数 comments :任意のコメント。
第 4 引数 seqSections :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)

測定結果の基本統計レポートを出力する。マスタープロセス以外では呼び出されてもなにもしない。第 2 ・ 3 ・ 4 引数は省略可。

■レポートの出力：MPI ランク別詳細レポート

```
void printDetail(FILE* fp, int legend, int seqSections)
```

第 1 引数 fp :出力ファイルポインタ。標準出力にレポートする場合は stdout を指定。
第 2 引数 legend :HWPC 記号説明の表示フラッグ (0:なし、1:表示する)
第 3 引数 seqSections :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)
MPI ランク毎に経過時間情報と計算量を出力する。計算量の自動算出モードで HWPC 統計情報を取得可能な場合は、HWPC 値も出力する。HWPC のイベント統計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。第 2 ・第 3 引数は省略可。

■レポートの出力：プロセスグループ毎の MPI ランク別詳細レポート

```
void printGroup(FILE* fp, MPI_Group p_group, MPI_Comm p_comm, int* pp_ranks,
                int group, int legend, int seqSections)
```

第 1 引数 fp :出力ファイルポインタ。標準出力にレポートする場合は stdout を指定。
第 2 引数 p_group :MPI_Group 型 group の group handle
第 3 引数 p_comm :MPI_Comm 型 group に対応する communicator
第 4 引数 pp_ranks :group を構成する rank 番号配列へのポインタ
第 5 引数 group :プロセスグループ番号（番号はユーザが任意に付けて良い）
第 6 引数 legend :HWPC 記号説明の表示フラッグ (0:なし、1:表示する)
第 7 引数 seqSections :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)
プロセスグループ毎の MPI ランク別詳細レポート、HWPC 詳細レポートを出力する。プロセスグループ p_group 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。そこで p_group とは別に 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。第 5 ・第 6 ・第 7 引数は省略可。

■レポートの出力：MPI_Comm_split 分離単位毎のグループ別詳細レポート

```
void printComm (FILE* fp, MPI_Comm new_comm, int icolor, int key, int legend,
                int seqSections)
```

第 1 引数 fp :出力ファイルポインタ。標準出力にレポートする場合は stdout を指定。
第 2 引数 new_comm :対応する communicator
第 3 引数 icolor :MPI_Comm_split() のカラー変数
第 4 引数 key :MPI_Comm_split() の key 変数
第 5 引数 legend :HWPC 記号説明の表示フラッグ (0:なし、1:表示する)
第 6 引数 seqSections :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)
MPI_Comm_split で分離されたの MPI ランクグループ毎に詳細レポート出力を行う。第 5 ・第 6 引数は省略可。

4.2　PMlib を呼び出す C++ ソースプログラム例

PMlib を呼び出して利用する C++ プログラムの書き方を説明する。
C++ プログラムから PMlib を利用するためには C++ ソースプログラムに対して以下の追加を行う。

1. PerfMonitor 用ヘッダファイル・namespace・クラス名の宣言追加
2. 初期化・測定区間の登録
3. 測定区間の実行
4. 測定結果の集計、レポートの出力

■元のソースプログラム　元のプログラムが以下の様な構成とする。単純化のため関数 compute1()/compute2() の内容や PMlib 以外のヘッダファイル類（例えば stdio.h, string など）は省略するが、関数 compute1() 内での演算量が 10^{10}、関数 compute2() 内でのデータ移動量が 2×10^{10} であるとする。

```
int main (int argc, char *argv[])
{
    compute1();      // 関数１
    compute2();      // 関数２
    return 0;
}
```

■PMlib 利用例１ソースプログラム　以下は計算量を自己申告モードで指定した場合の例である。

```
#include <PerfMonitor.h>                                  // 1.PerfMonitor 用ヘッダファイル
using namespace pm_lib;                                   //   PerfMonitor 用 namespace の宣言
PerfMonitor PM;                                           //   PerfMonitor クラス PM の宣言
int main (int argc, char *argv[])
{
    PM.initialize();                                      // 2.PerfMonitor クラス PM の初期化
    PM.setProperties("Block-1", PerfMonitor::CALC);       //   測定区間"Block-1"の登録
    PM.setProperties("Block-2", PerfMonitor::COMM);       //   測定区間"Block-2"の登録

    PM.start("Block-1");                                  // 3. 測定区間"Block-1"の実行開始
    compute1();
    float flop_count=1.0e10;
    PM.stop ("Block-1", flop_count);                      //   測定区間"Block-1"の実行終了

    PM.start("Block-2");                                  // 3. 測定区間"Block-2"の実行開始
    compute2();
    float move_count=2.0e10
    PM.stop ("Block-2", move_count);                      //   測定区間"Block-2"の実行終了

    PM.gather();                                          // 4.（全）測定結果の集計
    PM.print(stdout);                                     //   基本レポートの出力（経過時間降順）
    PM.printDetail(stdout);                               //   詳細レポートの出力
    return 0;
}
```

■PMlib 利用例２ソースプログラム　以下は計算量を自動算出モードで測定した場合の例である。

```
#include <PerfMonitor.h>                                  // 1.PerfMonitor 用ヘッダファイル
using namespace pm_lib;                                   //   PerfMonitor 用 namespace の宣言
PerfMonitor PM;                                           //   PerfMonitor クラス PM の宣言
int main (int argc, char *argv[])
```

```
{
    PM.initialize();                                    // 2.PerfMonitor クラス PM の初期化
    PM.setProperties("Block-1", PerfMonitor::CALC);  //   測定区間"Block-1"の登録
    PM.setProperties("Block-2", PerfMonitor::COMM);  //   測定区間"Block-2"の登録

    PM.start("Block-1");                                // 3. 測定区間"Block-1"の実行開始
    compute1();
    PM.stop ("Block-1");                                //   測定区間"Block-1"の実行終了

    PM.start("Block-2");                                // 3. 測定区間"Block-2"の実行開始
    compute2();
    PM.stop ("Block-2");                                //   測定区間"Block-2"の実行終了

    PM.gather();                                        // 4.（全）測定結果の集計
    PM.print(stdout);                                   //   基本レポートの出力（経過時間降順）
    PM.printDetail(stdout);                             //   詳細レポートの出力
    return 0;
}
```

4.3　プログラムのコンパイル・PMlib リンク方法

プログラムのコンパイル・リンク方法について具体的なプラットフォーム上での利用例で説明する。プラットフォームは京コンピュータ、および Intel Xeon サーバとする。

PMlib は第 3.2 節に説明した手順でインストールされているものとする。

4.3.1　京コンピュータ上でのコンパイル・リンク

ここでは京コンピュータのログインノード上で対話的にコンパイル・リンクを行うことを想定して、C++ ソースプログラムのコンパイル・PMlib のリンク方法をシェルスクリプト例によって示す。

現在のディレクトリにソースプログラムが main.cpp というファイル名で存在する場合を想定している。

シェルスクリプト 3 行目の変数 PMLIB_ROOT の値を PMlib がインストールされているディレクトリのパス名に設定した後、このシェルスクリプトを実行すると実行プログラム main.ex が生成される。

```
#! /bin/bash
source /home/system/Env_base
#         PMlib がインストールされているディレクトリ名を PMLIB_ROOT に指定する。
PMLIB_ROOT=${HOME}/pmlib/install_develop
PMLIB_INCLUDES="-I${PMLIB_ROOT}/include "
PMLIB_LDFLAGS="-L${PMLIB_ROOT}/lib -lPMmpi -lpapi_ext "
#         京コンピュータでは計算ノード用 PAPI は/lib64 以下にインストールされている
PAPI_ROOT=/
PAPI_INCLUDES="-I/include "
PAPI_LDFLAGS="-L/lib64 -lpapi -lpfm "
INCLUDES="${PMLIB_INCLUDES} ${PAPI_INCLUDES}"
LDFLAGS="${PMLIB_LDFLAGS} ${PAPI_LDFLAGS}"
CXXFLAGS="-Kopenmp,fast -Ntl_notrt -DUSE_PAPI "
mpiFCCpx ${CXXFLAGS} ${INCLUDES} -o main.ex main.cpp ${LDFLAGS}
```

4.3.2　Intel サーバ上でのコンパイル・リンク

Intel Xeon サーバではしばしば複数のコンパイラや MPI ライブラリが選択可能であり、様々な組み合わせが可能である。

C++ プログラムに PMlib をリンクした実行プログラムを作成するためには、

- コンパイラを利用するための設定
- MPI 用の設定

- PMlib がインストールされているディレクトリ名を PMLIB_ROOT に設定
- (PAPI ライブラリが利用可能な場合）PAPI ディレクトリ名を PAPI_ROOT に設定

のそれぞれを指定した上で C++ プログラムのコンパイル・リンクを行う。

以下に Intel コンパイラ、Intel MPI を利用して Intel 環境で実行プログラムを作成する例を示す。

```
#! /bin/bash
#   Intel コンパイラ用の設定（ディレクトリ名はシステム毎により異なる）
INTEL_DIR=/usr/local/intel/composer_xe_2013
source ${INTEL_DIR}/bin/compilervars.sh intel64
#   Intel MPI 用の設定（ディレクトリ名はシステム毎により異なる）
export I_MPI_ROOT=/usr/local/intel/impi/4.1.0.024
source ${I_MPI_ROOT}/bin64/mpivars.sh
export I_MPI_F90=ifort
export I_MPI_F77=ifort
export I_MPI_CC=icc
export I_MPI_CXX=icpc
export  I_HYDRA_BOOTSTRAP=ssh
export  I_HYDRA_BOOTSTRAP_EXEC=/usr/bin/ssh

#         PMlib がインストールされているディレクトリ名を PMLIB_ROOT に指定する。
PMLIB_ROOT=${HOME}/pmlib/install_develop
PMLIB_INCLUDES="-I${PMLIB_ROOT}/include "
PMLIB_LDFLAGS="-L${PMLIB_ROOT}/lib -lPMmpi "

#         PAPI がインストールされているディレクトリ名を PAPI_ROOT に指定する。
#         もし PAPI がインストールされてないシステムでは下の４行は不要（削除する）
PAPI_ROOT=/usr/local/papi/papi-5.3.2/intel
PAPI_INCLUDES="-I${PAPI_ROOT}/include "
PAPI_LDFLAGS="-L${PMLIB_ROOT}/lib -lpapi_ext -L${PAPI_ROOT}/lib -lpapi -lpfm "
export LD_LIBRARY_PATH=${PAPI_ROOT}/lib:${LD_LIBRARY_PATH}

INCLUDES="${PMLIB_INCLUDES} ${PAPI_INCLUDES}"
LDFLAGS="${PMLIB_LDFLAGS} ${PAPI_LDFLAGS}"
export LD_LIBRARY_PATH=${PAPI_ROOT}/lib:${LD_LIBRARY_PATH}
CXXFLAGS="-O3 -openmp -DUSE_PAPI "

mpicxx ${CXXFLAGS} ${INCLUDES} -o main.ex main.cpp ${LDFLAGS}
```

■(参考)Intel 環境でのコンパイラ・MPI リンカコマンド　Intel Xeon サーバではしばしば複数のコンパイラや複数の MPI ライブラリが選択可能である。コンパイラでは Intel コンパイラ・PGI コンパイラ・GNU コンパイラなどが頻用され、MPI ライブラリでは Intel MPI ・OpenMPI ・MPICH ・MVAPICH などが頻用される。全ての組み合わせについて例を提示するのは困難なため、上記では代表例として Intel コンパイラ + Intel MPI の組み合わせで PMlib を利用するプログラム例を示した。

コンパイラコマンド利用のための設定はシステム毎により異なり、上記例の様に環境変数をユーザーが明示的に指定する必要があるシステムと、module コマンドを利用して、

```
#! /bin/bash
module load intel impi
```

等の宣言で簡便に利用できる設定がされているシステムとがある。

またシステムによっては Intel コンパイラの商用ライセンスファイルの場所を環境変数で指定しなくてはいけない場合もある。

```
#! /bin/bash
export INTEL_LICENSE_FILE=/usr/local/intel/flexlm/server.lic
```

MPI をリンクするための MPI コンパイラコマンド名は Intel コンパイラの場合通常 mpiifort/ mpiicc/ mpiicpc とされる。コンパイラコマンドの利用方法はシステムの利用手引書などにより確認する。

4.4　プログラムの実行方法

プログラムはバッチジョブとして投入実行されることを想定し、プラットフォーム毎に具体的なジョブスクリプト例を説明する。対象プラットフォームは京コンピュータ、および Intel Xeon サーバとする。

4.4.1　京コンピュータ上でのプログラム実行

前節の手順で作成した実行プログラム main.ex を京コンピュータの計算ノードで実行するバッチジョブスクリプト例と、投入するコマンドを示す。

バッチジョブスクリプトにおいて、#PJM で始まる指示行の内、ファイルステージングの stgin-basedir ”ディレクトリ名” において、前節の手順で実行プログラムを作成したディレクトリ名を指定する。京コンピュータ標準のプロファイラと PMlib HWPC/PAPI とを同時に利用する事はできないため、京コンピュータ標準のプロファイラ機能を抑止するためのコマンド/opt/FJSVXosPA/bin/xospastop を起動していることに注意。

```
K$ cat x.run-test1.sh
#!/bin/bash
#PJM -N MYTEST1
#PJM --rsc-list "elapse=1:00:00"
#PJM --rsc-list "node=1"
#PJM --mpi "proc=2"
#PJM -j
#PJM -S
# stage io files
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"
#PJM --stgin-basedir "実行プログラムが生成されたディレクトリ"
#PJM --stgin "rank=* main.ex %r:./main.ex"
#
source /work/system/Env_base
set -x
date
hostname
/opt/FJSVXosPA/bin/xospastop
pwd
export HWPC_CHOOSER=FLOPS
NPROCS=2
export OMP_NUM_THREADS=4
mpiexec -n ${NPROCS} ./main.ex

K$ pjsub x.run-test1.sh
```

4.4.2　Intel サーバ上でのプログラム実行

Intel サーバでは様々なバッチジョブ管理ソフトが利用されているが、以下の例では LSF ジョブ管理ソフト用のジョブスクリプトを示す。#BSUB で始まる行は LSF への指示行である。それ以外の行はジョブの処理が行われるシェルへのコマンドである。

コンパイル・リンク時と同様に、MPI プログラムの起動コマンド呼び出しもシステム毎により異なり、module コマンドを利用する場合や環境変数を明示的に指定する場合がある。

以下は環境変数をユーザーが明示的に指定し、対話的に実行するシェルスクリプトの例である。変数 BINDIR に実行プログラム main.ex が生成されたディレクトリ名を設定後、バッチジョブ管理ソフト（この例では LSF) へ投入する。

```
intel$ cat x.run-test1.sh
#!/bin/bash
#BSUB -J MYTEST1
#BSUB -o MYTEST1.%J
#BSUB -n 2
#BSUB -R "span[ptile=1]"
#BSUB -x

INTEL_DIR=/usr/local/intel/composer_xe_2013
source ${INTEL_DIR}/bin/compilervars.sh intel64
I_MPI_ROOT=/usr/local/intel/impi/4.1.0.024
source ${I_MPI_ROOT}/bin/64/mpivars.sh
export I_MPI_F90=ifort
export I_MPI_F77=ifort
export I_MPI_CC=icc
export I_MPI_CXX=icpc
export  I_HYDRA_BOOTSTRAP=ssh
export  I_HYDRA_BOOTSTRAP_EXEC=/usr/bin/ssh

BINDIR=実行プログラム main.ex が生成されたディレクトリ
cd ${BINDIR}/

export HWPC_CHOOSER=FLOPS
NPROCS=2
export OMP_NUM_THREADS=4
mpirun -np ${NPROCS} ./main.ex


intel$ bsub < x.run-test1.sh
```

第 5 章　Fortran プログラム用 PMlib サブルーチンの仕様と利用例

5.1　Fortran プログラム用 PMlib サブルーチンの仕様

Fortran プログラムからユーザーが呼び出し可能な PMlib のサブルーチン群である、PerfMonitor クラスルーチンの仕様を以下に説明する。

Fortran API と C++ API とでは引数の型が異なる場合があるので留意する。
Fortran API では全ての引数を渡す必要がある（省略できない）またユーザープログラムが Fortran だけの場合は特に意識する必要はないが、もしユーザープログラムが Fortran と C++ の混合プログラムである場合は、Fortran サブルーチンに character 型の引数が渡される場合、コンパイラはその引数変数の長さ（character 文字数）の情報を引数並びの末尾に自動的に追加してしまう（暗黙の追加引数）ことに留意する。

■初期化

```
subroutine f_pm_initialize (init_nWatch)
```

引数 (IN) integer `init_nWatch` は最初に確保する測定区間数。登録予定の測定区間数を引数として指定する。測定区間数が不明、あるいは動的に増加する場合は引数 `init_nWatch` の値を 1 と指定するとよい。指定した測定区間数では不足になった時点で PMlib は必要な区間数を動的に増加させる。

サブルーチン `f_pm_initialize` はプログラム実行時に 1 度だけ呼び出す。

■測定区間の登録とプロパティ設定

```
subroutine f_pm_setproperties (fc, ftype, fexclusive)
```

第 1 引数 (IN) character*(*) `fc` :ラベル文字列。測定区間はラベル `fc` で識別される。ラベル毎に対応したキー番号が内部で自動生成される
第 2 引数 (IN) integer `ftype` :測定計算量のタイプ (0:通信, 1:計算)。明示的な申告モードの場合には必須、自動算出モードの場合は無視される。
第 3 引数 (IN) integer `fexclusive` :測定の排他フラグ。(0:非排他測定区間, 1:排他測定区間)

サブルーチン `f_pm_setproperties` は登録する測定区間数分よびだす。

■測定区間の開始

```
subroutine f_pm_start (fc)
```

第 1 引数 (IN) character*(*) `fc` :ラベル文字列。測定区間を識別するために用いる。

ラベルで識別される測定区間の始まりを指定する。最初に f_pm_start () が呼ばれる前に測定区間の登録とプロパティ

設定がされていなくてはならない。

■測定区間の終了

```
subroutine f_pm_stop (fc, fpt, tic)
```

第 1 引数 (IN) character*(*) fc :ラベル文字列。測定区間を識別するために用いる。
第 2 引数 (IN) real*8 fpt :計算量。演算量 (Flop) または通信量 (バイト)。
第 3 引数 (IN) integer tic :計算量に乗じる係数。測定区間を複数回実行する場合、その繰り返し数と考えて良い。
ラベル fc で識別される測定区間の終わりを指定する。
f_pm_start () から f_pm_stop () までが 1 区間となる。 1 区間が複数回呼び出される場合に加えて、別の離れた区間に同じラベル名を指定した f_pm_start () から f_pm_stop () の測定結果も同一区間の値として合算される。
Fortran PMlib サブルーチンでは引数を省略する事はできない。
第 2・第 3 引数で与える計算量は明示的な申告モードでだけ意味を持ち、演算量または通信量が 1 区間 1 回あたりで fpt*tic として算出される。自動算出モードでは無視される。

■測定結果の集約

```
subroutine f_pm_gather ()
```

全プロセスの全測定結果情報をマスタープロセスに集約する。もし HWPC 情報が取得可能であれば各測定区間の HWPC によるイベントカウンターの統計値を取得する。測定結果の平均値・標準偏差・経過時間降順ソートなどの基礎的な統計情報を計算する。初期化時にスタートさせた全計算時間用測定をストップさせる。
各種レポート出力ルーチンを利用する前に、本サブルーチン f_pm_gather を 1 度だけ呼び出しておく必要がある。

■レポートの出力：基本統計レポート

```
subroutine f_pm_print (fp, psort)
```

第 1 引数 (IN) character*(*) fp :出力ファイル名。もし fp の値が NULL(””) の場合は標準出力が選択される。
第 2 引数 (IN) integer psort :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)
測定結果の基本統計レポートを出力する。マスタープロセス以外では呼び出されてもなにもしない。

■レポートの出力：MPI ランク別詳細レポート

```
subroutine f_pm_printdetail (fc, legend, psort)
```

第 1 引数 (IN) character*(*) fp :出力ファイル名。もし fp の値が NULL(””) の場合は標準出力が選択される。
第 2 引数 (IN) integer legend :HWPC 記号説明の表示 (0:表示する、1:しない) HWPC 情報が取得可能な場合にのみ有効
第 3 引数 (IN) integer psort :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)
MPI ランク毎に経過時間情報と計算量を出力する。計算量の自動算出モードで HWPC 統計情報を取得可能な場合は、HWPC 値も出力する。HWPC のイベント統計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。

■レポートの出力：プロセスグループ毎の MPI ランク別詳細レポート

```
subroutine f_pm_printgroup (fc, p_group, p_comm, pp_ranks, group, legend, psort)
```

第 1 引数 (IN) character*(*) fp :出力ファイル名。もし fp の値が NULL(””) の場合は標準出力が選択される。
第 2 引数 (IN) integer p_group :MPI Group の group handle
第 3 引数 (IN) integer p_comm :MPI Group に対応する communicator
第 4 引数 (IN) integer pp_ranks(:) :MPI Group を構成する rank 番号の一次元配列
第 5 引数 (IN) integer group :プロセスグループの番号（番号はユーザが任意に付けて良い）
第 6 引数 (IN) integer legend :HWPC 記号説明の表示 (0:表示する、1:しない)
第 7 引数 (IN) integer psort :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)

プロセスグループ毎の MPI ランク別詳細レポート、HWPC 詳細レポートを出力する。プロセスグループ p_group 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。そこで p_group とは別に 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。

C++ MPI での MPI_Group, MPI_Comm 型は呼び出す Fortran MPI では integer 型で宣言される。

■レポートの出力：MPI_Comm_split 分離単位毎のグループ別詳細レポート

```
subroutine f_pm_printcomm (fp, new_comm, icolor, key, legend, psort)
```

第 1 引数 (IN) character*(*) fp :出力ファイル名。もし fp の値が NULL(””) の場合は標準出力が選択される。
第 2 引数 (IN) integer new_comm :対応する communicator
第 3 引数 (IN) integer icolor :MPI_Comm_split() のカラー変数
第 4 引数 (IN) integer key :MPI_Comm_split() の key 変数
第 5 引数 (IN) integer legend :HWPC 記号説明の表示 (0:表示する、1:しない)
第 6 引数 (IN) integer psort :測定区間の表示順 (0:経過時間順にソート後表示、1:登録順で表示)

MPI_Comm_split で分離された MPI ランクグループ毎に詳細レポート出力を行う。

5.2　PMlib を呼び出す Fortran ソースプログラム例

PMlib を呼び出して利用する Fortran プログラムの書き方を説明する。

Fortran プログラムから PMlib を利用するためには Fortran ソースプログラムに対して以下の追加を行う。

1. PMlib 初期化・測定区間の登録
2. 測定区間の実行
3. 測定結果の集計・レポートの出力

■元のソースプログラム　元のプログラムが以下の様な構成とする。単純化のためサブルーチン compute1()/compute2() の内容や変数の型宣言などは省略するが、サブルーチン *compute*1() 内での演算量が 10^{10}、*compute*2() 内でのデータ移動量が 2×10^{10}であると仮定する。

```
program check
call compute1 ()                                    ! サブルーチン compute1
call compute2 ()                                    ! サブルーチン compute2
end
```

■PMlib 利用例１ソースプログラム

以下は計算量を自己申告モードで指定した場合の例である。

```
program check
icalc=1                                             ! 測定量のタイプ 1:演算
imove=0                                             ! 測定量のタイプ 0:通信
iexcl=1                                             ! 排他的測定区間
call f_pm_initialize (2)                            ! PMlib 用の初期化
call f_pm_setproperties ("Block-1", icalc, iexcl)   ! 測定区間"Block-1"の登録
call f_pm_setproperties ("Block-2", imove, iexcl)   ! 測定区間"Block-2"の登録

call f_pm_start ("Block-1")                         ! 測定区間"Block-1"の開始
call compute1 ()
call f_pm_stop ("Block-1", 1.0e10, 0)               ! 測定区間"Block-1"の終了

call f_pm_start ("Block-2")                         ! 測定区間"Block-2"の開始
call compute2 ()
call f_pm_stop ("Block-2", 2.0e10, 0)               ! 測定区間"Block-2"の終了

call f_pm_gather ()                                 ! 全測定結果の集計
call f_pm_print ("", 0)                             ! 基本レポートの出力
call f_pm_printdetail ("", 0, 0)                    ! 詳細レポートの出力
end
```

■PMlib 利用例２ソースプログラム

以下は計算量を自動算出した場合の例である。

```
program check
icalc=1                                             ! 自動算出モードでは無視される
iexcl=1                                             ! 排他的測定区間
call f_pm_initialize (2)                            ! PMlib 用の初期化
call f_pm_setproperties ("Block-1", icalc, iexcl)   ! 測定区間"Block-1"の登録
call f_pm_setproperties ("Block-2", icalc, iexcl)   ! 測定区間"Block-2"の登録
```

```
call f_pm_start ("Block-1")                          ! 測定区間"Block-1"の開始
call compute1 ()
call f_pm_stop ("Block-1", 0.0, 0)                   ! 測定区間"Block-1"の終了

call f_pm_start ("Block-2")                          ! 測定区間"Block-2"の開始
call compute2 ()
call f_pm_stop ("Block-2", 0.0, 0)                   ! 測定区間"Block-2"の終了

call f_pm_gather ()                                  ! 全測定結果の集計
call f_pm_print ("", 0)                              ! 基本レポートの出力
call f_pm_printdetail ("", 0, 0)                     ! 詳細レポートの出力
end
```

5.3　プログラムのコンパイル・PMlib リンク方法

プログラムのコンパイル・リンク方法について具体的なプラットフォーム上での利用例で説明する。プラットフォームは京コンピュータ、および Intel Xeon サーバとする。

PMlib は第 3.2 節に説明した手順でインストールされているものとする。

5.3.1　京コンピュータでの PMlib ライブラリのリンク

ここでは京コンピュータのログインノード上で対話的にコンパイル・リンクを行うことを想定して、Fortran ソースプログラムのコンパイル・PMlib のリンク方法をシェルスクリプト例によって示す。

現在のディレクトリにソースプログラムが main.f90 というファイル名で存在する場合を想定している。

シェルスクリプト 3 行目の変数 PMLIB_ROOT の値を PMlib がインストールされているディレクトリのパス名に設定した後、このシェルスクリプトを実行すると実行プログラム main.ex が生成される。

```
#! /bin/bash
source /home/system/Env_base
#    PMlib がインストールされているディレクトリ名を PMLIB_ROOT に指定する。
PMLIB_ROOT=${HOME}/pmlib/install_develop
PMLIB_INCLUDES="-I${PMLIB_ROOT}/include "
PMLIB_LDFLAGS="-L${PMLIB_ROOT}/lib -lPMmpi -lpapi_ext "
#    京コンピュータでは計算ノード用 PAPI は/lib64 以下にインストールされている
PAPI_ROOT=/
PAPI_INCLUDES="-I/include "
PAPI_LDFLAGS="-L/lib64 -lpapi -lpfm "
INCLUDES="${PMLIB_INCLUDES} ${PAPI_INCLUDES}"
LDFLAGS="${PMLIB_LDFLAGS} ${PAPI_LDFLAGS} --linkfortran "
CXXFLAGS="-Kopenmp,fast -Ntl_notrt -DUSE_PAPI "
FCFLAGS="-Cpp -Kopenmp,fast -Ntl_notrt -DUSE_PAPI "
mpifrtpx ${FCFLAGS} ${INCLUDES} -c main.f90
mpiFCCpx ${CXXFLAGS} ${INCLUDES} -o main.ex main.o ${LDFLAGS}
```

5.3.2　Intel サーバ上でのコンパイル・リンク

Intel 環境で Fortran プログラムに PMlib をリンクした実行プログラムを作成するためには、

- コンパイラを利用するための設定
- MPI 用の設定
- PMlib がインストールされているディレクトリ名を PMLIB_ROOT に設定
- (PAPI ライブラリが利用可能な場合）PAPI ディレクトリ名を PAPI_ROOT に設定

のそれぞれを設定した上で Fortran プログラムのコンパイル・リンクを行う。

以下に Intel コンパイラ、Intel MPI を利用して実行プログラムを作成するシェルスクリプト例を示す。

```
#! /bin/bash
#         Intel コンパイラ用の設定（ディレクトリ名はシステム毎により異なる）
INTEL_DIR=/usr/local/intel/composer_xe_2013
source ${INTEL_DIR}/bin/compilervars.sh intel64
#         Intel MPI 用の設定（ディレクトリ名はシステム毎により異なる）
export I_MPI_ROOT=/usr/local/intel/impi/4.1.0.024
source ${I_MPI_ROOT}/bin64/mpivars.sh
export I_MPI_F90=ifort
export I_MPI_F77=ifort
export I_MPI_CC=icc
export I_MPI_CXX=icpc
export  I_HYDRA_BOOTSTRAP=ssh
export  I_HYDRA_BOOTSTRAP_EXEC=/usr/bin/ssh
#         PMlib がインストールされているディレクトリ名を PMLIB_ROOT に指定する。
PMLIB_ROOT=${HOME}/pmlib/install_develop
PMLIB_INCLUDES="-I${PMLIB_ROOT}/include "
PMLIB_LDFLAGS="-L${PMLIB_ROOT}/lib -lPMmpi "
#         PAPI がインストールされているディレクトリ名を PAPI_ROOT に指定する。
#         もし PAPI がインストールされてないシステムでは下の４行は不要（削除する）
PAPI_ROOT=/usr/local/papi/papi-5.3.2/intel
PAPI_INCLUDES="-I${PAPI_ROOT}/include "
PAPI_LDFLAGS="-L${PMLIB_ROOT}/lib -lpapi_ext -L${PAPI_ROOT}/lib -lpapi -lpfm "
export LD_LIBRARY_PATH=${PAPI_ROOT}/lib:${LD_LIBRARY_PATH}

INCLUDES="${PMLIB_INCLUDES} ${PAPI_INCLUDES}"
LDFLAGS="${PMLIB_LDFLAGS} ${PAPI_LDFLAGS}"
FCFLAGS="-cpp -O3 -openmp"
LDFLAGS="${LDFLAGS} -lstdc++ "
mpiifort ${FCFLAGS} ${INCLUDES} -o main.ex main.f90 ${LDFLAGS}
```

■(参考)Intel 環境でのコンパイラ・MPI リンカコマンド　Intel Xeon サーバではしばしば複数のコンパイラや複数の MPI ライブラリが選択可能である。コンパイラでは Intel コンパイラ・PGI コンパイラ・GNU コンパイラなどが頻用され、MPI ライブラリでは Intel MPI ・OpenMPI ・MPICH ・MVAPICH などが頻用される。全ての組み合わせについて例を提示するのは困難なため、上記では代表例として Intel コンパイラ + Intel MPI の組み合わせで PMlib を利用するプログラム例を示した。

コンパイラコマンド利用のための設定はシステム毎により異なり、上記例の様に環境変数をユーザーが明示的に指定する必要があるシステムと、module コマンドを利用して、

```
#! /bin/bash
module load intel impi
```

等の宣言で簡便に利用できる設定がされているシステムとがある。

またシステムによっては Intel コンパイラの商用ライセンスファイルの場所を環境変数で指定しなくてはいけない場合もある。

```
#! /bin/bash
export INTEL_LICENSE_FILE=/usr/local/intel/flexlm/server.lic
```

MPI をリンクするための MPI コンパイラコマンド名は Intel コンパイラの場合通常 mpiifort/ mpiicc/ mpiicpc とされる。コンパイラコマンドの利用方法はシステムの利用手引書などにより確認する。

5.4　プログラムの実行方法

プログラムはバッチジョブとして投入実行されることを想定し、プラットフォーム毎に具体的なジョブスクリプト例を説明する。対象プラットフォームは京コンピュータ、および Intel Xeon サーバとする。

5.4.1　京コンピュータ上でのプログラム実行

前節の手順で作成した実行プログラム main.ex を京コンピュータの計算ノードで実行するバッチジョブスクリプト例と、投入するコマンドを示す。

バッチジョブスクリプトにおいて、#PJM で始まる指示行の内、ファイルステージングの stgin-basedir ”ディレクトリ名” において、前節の手順で実行プログラムを作成したディレクトリ名を指定する。京コンピュータ標準のプロファイラと PMlib HWPC/PAPI とを同時に利用する事はできないため、京コンピュータ標準のプロファイラ機能を抑止するためのコマンド/opt/FJSVXosPA/bin/xospastop を起動していることに注意。

```
K$ cat x.run-test1.sh
#!/bin/bash
#PJM -N MYTEST1
#PJM --rsc-list "elapse=1:00:00"
#PJM --rsc-list "node=1"
#PJM --mpi "proc=2"
#PJM -j
#PJM -S
# stage io files
#PJM --stg-transfiles all
#PJM --mpi "use-rankdir"
#PJM --stgin-basedir "実行プログラムが生成されたディレクトリ"
#PJM --stgin "rank=* main.ex %r:./main.ex"
#
source /work/system/Env_base
set -x
date
hostname
/opt/FJSVXosPA/bin/xospastop
pwd
export HWPC_CHOOSER=FLOPS
NPROCS=2
export OMP_NUM_THREADS=4
mpiexec -n ${NPROCS} ./main.ex

K$ pjsub x.run-test1.sh
```

5.4.2　Intel サーバ上でのプログラム実行

Intel サーバでは様々なバッチジョブ管理ソフトが利用されているが、以下の例では LSF ジョブ管理ソフト用のジョブスクリプトを示す。#BSUB で始まる行は LSF への指示行である。それ以外の行はジョブの処理が行われるシェルへのコマンドである。

コンパイル・リンク時と同様に、MPI プログラムの起動コマンド呼び出しもシステム毎により異なり、module コマンドを利用する場合や環境変数を明示的に指定する場合がある。

以下は環境変数をユーザーが明示的に指定し、対話的に実行するシェルスクリプトの例である。変数 BINDIR に実行プログラム main.ex が生成されたディレクトリ名を設定後、バッチジョブ管理ソフト（この例では LSF) へ投入する。

```
intel$ cat x.run-test1.sh
#!/bin/bash
```

```
#BSUB -J MYTEST1
#BSUB -o MYTEST1.%J
#BSUB -n 2
#BSUB -R "span[ptile=1]"
#BSUB -x

INTEL_DIR=/usr/local/intel/composer_xe_2013
source ${INTEL_DIR}/bin/compilervars.sh intel64
I_MPI_ROOT=/usr/local/intel/impi/4.1.0.024
source ${I_MPI_ROOT}/bin64/mpivars.sh
export I_MPI_F90=ifort
export I_MPI_F77=ifort
export I_MPI_CC=icc
export I_MPI_CXX=icpc
export  I_HYDRA_BOOTSTRAP=ssh
export  I_HYDRA_BOOTSTRAP_EXEC=/usr/bin/ssh

BINDIR=実行プログラム main.ex が生成されたディレクトリ
cd ${BINDIR}/

export HWPC_CHOOSER=FLOPS
NPROCS=2
export OMP_NUM_THREADS=4
mpirun -np ${NPROCS} ./main.ex


intel$ bsub < x.run-test1.sh
```

第 6 章　PMlib の出力レポート

6.1　出力レポートの種類

PMlib が出力するレポートの種類は 3 種類ある。

- 基本統計レポート:
  全プロセスの平均した基本情報。　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がレポートされる。
- MPI ランク別詳細レポート:
  MPI ランク（プロセス）毎の情報。　各 MPI プロセス毎のプロファイルをレポート。　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。
- HWPC 統計情報詳細レポート:
  HWPC（ハードウエア性能カウンタ）イベントグループの統計情報。　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算される。

主要な統計量は測定区間毎に累積された経過時間と測定計算量のボリュームである。計算量には「通信」と「演算」のタイプがあり、通信（転送）は CPU・メモリ間、あるいは CPU・CPU 間 (ノード間) のデータ移動処理を、演算は CPU 内での浮動小数点演算や整数演算を意味する。

6.2　各出力レポートの情報

6.2.1　基本統計レポート

基本統計レポートは `print(C++)` および `f_pm_print(Fortran)` が出力する。

■基本統計レポートの出力例

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

Timing Statistics Report from PMlib version 4.1.4
Linked PMlib supports: MPI, OpenMP, HWPC
Host name : vsp21
Date      : 2015/10/27 : 19:59:18
Mr. Bean
Parallel Mode:   Hybrid (4 processes x 8 threads)
The environment variable HWPC_CHOOSER is not provided. No HWPC report.

Total execution time            = 2.615132e-01 [sec]
Total time of measured sections = 2.589099e-01 [sec]

Exclusive Sections statistics per process and total job.

Section         |    call  |        accumulated time[sec]           | [flop counts or byte counts ]
Label           |          |      avr   avr[%]      sdv   avr/call  |      avr       sdv   speed
----------------+----------+----------------------------------------+---------------------------
First location  :        3   1.569e-01  60.60  4.64e-04  5.230e-02    0.000e+00  0.00e+00   0.00 Mflops
Third location  :        1   5.101e-02  19.70  1.20e-04  5.101e-02    1.601e+10  0.00e+00 313.82 GB/sec
Second location :        1   5.100e-02  19.70  1.27e-04  5.100e-02    4.000e+09  0.00e+00  78.42 Gflops
```

```
---------------+----------+---------------------------------------+----------------------------
Sections per process         2.079e-01                                4.000e+09           19.24 Gflops
Sections per process         5.101e-02                                1.601e+10          313.82 GB/sec
---------------+----------+---------------------------------------+----------------------------
Sections total job           2.079e-01                                1.600e+10           76.96 Gflops
Sections total job           5.101e-02                                6.403e+10            1.26 TB/sec
```

経過時間および計算量の測定結果を表示する。プログラムが MPI 並列化されている場合は全 MPI プロセスの平均値を表示する。各プロセスがスレッド並列化されている場合は元プロセスに帰属するスレッドの計算量が合算されて元プロセスの値となる。

出力される各レコードの内容を説明する。

第 1 行目　Timing Statistics Report from PMlib version バージョン名

第 2 行目　リンクされている PMlib がサポートする並列モデル (MPI, OpenMP, HWPC)

第 3 行目　マスタープロセスが起動されるホスト名

第 4 行目　プログラム実行日時

第 5 行目　コメント（省略される場合有り）

第 6 行目　プログラムが実行した並列プログラムモデル

第 7 行目　環境変数 HWPC_CHOOSER の指定有無

空行

第 9 行目　Total execution time = マスタープロセスの経過時間。`initialize(C++)`,`f_pm_initialize(Fortran)` の呼び出しから `gather(C++)`,`f_pm_gather(Fortran)` の呼び出しまでの経過時間

第 10 行目　Total time of measured sections = 全測定区間の合計経過時間。測定区間毎に全プロセスの平均経過時間を算出し、全測定区間の合計値を出力

空行

第 12 行目　Exclusive Sections statistics per process and total job.

空行

第 13 行目・第 14 行目　数値の内容説明ヘッダ行

- Section Label: 測定区間のラベル
- call：呼び出し回数
- accumulated time[sec]：累積経過時間
  - avr ：累積経過時間（秒）
  - avr[%] ：全体に占める %
  - sdv ：標準偏差
  - avr/call ：計測区間 1 回あたりの経過時間（秒）
- flop counts or byte counts : 演算数または移動バイト数
  - avr ：累積経過時間（秒）
  - sdv ：標準偏差
  - speed ：計算量の速度 (Flops 又は Bytes/sec）

第 15 行目以降　測定区間毎の各数値

末尾 2 行　Sections per process：全測定区間の 1 プロセス平均経過時間、計算量と速度

末尾 2 行　Sections total job：全測定区間の全プロセス合計計算量と速度

6.2.2　MPI ランク別詳細レポート

MPI ランク別詳細レポートは `printDetail(C++)` あるいは `f_pm_printdetail(Fortran)` が出力する。

各測定区間のラベル行、Header 行に続いて、測定対象タイプ type で指定した測定値が各プロセス毎に出力される。その測定値の平均値が基本統計レポートの出力値となる。

■MPI ランク別詳細レポートの出力例

```
# 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  1.573e-01   60.8  0.000e+00  5.244e-02  0.000e+00  0.000e+00 Flops
Rank     1 :        3  1.567e-01   60.5  6.387e-04  5.223e-02  0.000e+00  0.000e+00 Flops
Rank     2 :        3  1.563e-01   60.4  9.909e-04  5.211e-02  0.000e+00  0.000e+00 Flops
Rank     3 :        3  1.572e-01   60.7  1.106e-04  5.241e-02  0.000e+00  0.000e+00 Flops
Label  Third location
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   flop|msg    speed
Rank     0 :        1  5.107e-02   19.7  2.980e-05  5.107e-02  1.601e+10  3.135e+11 Bytes/sec
Rank     1 :        1  5.110e-02   19.7  0.000e+00  5.110e-02  1.601e+10  3.133e+11 Bytes/sec
Rank     2 :        1  5.083e-02   19.6  2.639e-04  5.083e-02  1.601e+10  3.149e+11 Bytes/sec
Rank     3 :        1  5.104e-02   19.7  5.293e-05  5.104e-02  1.601e+10  3.136e+11 Bytes/sec
Label  Second location
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   flop|msg    speed
Rank     0 :        1  5.107e-02   19.7  3.409e-05  5.107e-02  4.000e+09  7.833e+10 Flops
Rank     1 :        1  5.110e-02   19.7  0.000e+00  5.110e-02  4.000e+09  7.828e+10 Flops
Rank     2 :        1  5.082e-02   19.6  2.820e-04  5.082e-02  4.000e+09  7.871e+10 Flops
Rank     3 :        1  5.104e-02   19.7  6.413e-05  5.104e-02  4.000e+09  7.838e+10 Flops
```

以下の様なレコードセットの繰り返しとなっている。
第 1 行目（Label で始まる行）：測定区間のラベル文字列
第 2 行目　数値の内容説明ヘッダ行
第 3 行目　以下の数値の並び

- Header ID: MPI プロセスのランク番号
- call：測定区間の呼び出し回数
- time[s]：当プロセスの経過時間
- time[%]：全測定区間の経過時間に対する当区間の経過時間比率 (%)
- t_wait[s]：経過時間が最大のプロセスと当プロセスの経過時間の差（待ち時間）
- t[s]/call； 1 呼び出し回数あたりの経過時間
- flop | msg：計算量（演算量またはデータ移動量）
- speed：計算量の速度

6.2.3　HWPC 統計情報詳細レポート

環境変数 HWPC_CHOOSER の値が指定されていて測定する計算量が自動算出モードである場合に printDetail(C++) あるいは f_pm_printdetail(Fortran) が呼ばれた場合、MPI ランク別詳細レポートに続いて HWPC 統計情報詳細レポートを出力する。

環境変数 HWPC_CHOOSER の値に応じて HWPC が自動算出した内容が出力される。

各測定区間のラベル行、Header 行に続いて、各プロセス毎の HWPC 測定値および統計値が出力される。出力される内容はプラットフォームにより、また HWPC_CHOOSER の値により異なる。

6.2.3.1　Intel Xeon サーバでの出力例

Intel Xeon サーバで PMlib の HWPC 統計情報詳細レポート出力例を HWPC_CHOOSER=FLOPS, BANDWIDTH, CACHE, CYCLE の各場合について示す。

■HWPC **統計情報詳細レポート**: Intel HWPC_CHOOSER=FLOPS **指定例**

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.438e+10  4.400e+01  9.167e+10
Rank     1 :  1.438e+10  2.900e+01  9.145e+10
Rank     2 :  1.438e+10  2.800e+01  9.148e+10
Rank     3 :  1.437e+10  2.800e+01  9.187e+10
Label   Second location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  4.753e+09  8.000e+00  9.306e+10
Rank     1 :  4.753e+09  1.000e+01  9.309e+10
Rank     2 :  4.753e+09  1.200e+01  9.310e+10
Rank     3 :  4.753e+09  1.100e+01  9.306e+10
Label   Third location
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  4.753e+09  1.100e+01  9.310e+10
Rank     1 :  4.754e+09  1.400e+01  9.309e+10
Rank     2 :  4.753e+09  1.200e+01  9.311e+10
Rank     3 :  4.753e+09  9.000e+00  9.308e+10
```

■HWPC **統計情報詳細レポート**: Intel HWPC_CHOOSER=BANDWIDTH **指定例**

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     LD_INS     SR_INS     L1_HIT    HIT_LFB L2_DRD_REQ L2_DRD_HIT L2_PF_MISS L2_RFO_MIS   [HW B/s]
Rank     0 :  3.103e+09  9.339e+06  2.540e+09  5.501e+08  3.200e+08  1.844e+08  4.403e+08  3.217e+04  2.364e+11
Rank     1 :  3.103e+09  9.296e+06  2.539e+09  5.503e+08  3.202e+08  1.844e+08  4.399e+08  3.257e+04  2.356e+11
Rank     2 :  3.097e+09  8.414e+06  2.535e+09  5.489e+08  3.204e+08  1.847e+08  4.390e+08  3.122e+04  2.341e+11
Rank     3 :  3.097e+09  8.443e+06  2.534e+09  5.499e+08  3.203e+08  1.845e+08  4.393e+08  3.111e+04  2.352e+11
Label   Second location
Header  ID :     LD_INS     SR_INS     L1_HIT    HIT_LFB L2_DRD_REQ L2_DRD_HIT L2_PF_MISS L2_RFO_MIS   [HW B/s]
Rank     0 :  1.028e+09  1.504e+06  8.404e+08  1.837e+08  1.066e+08  6.146e+07  1.469e+08  7.660e+02  2.405e+11
Rank     1 :  1.028e+09  1.458e+06  8.401e+08  1.837e+08  1.067e+08  6.149e+07  1.469e+08  6.910e+02  2.408e+11
Rank     2 :  1.028e+09  1.449e+06  8.400e+08  1.837e+08  1.066e+08  6.148e+07  1.469e+08  7.320e+02  2.407e+11
Rank     3 :  1.028e+09  1.463e+06  8.400e+08  1.838e+08  1.067e+08  6.146e+07  1.469e+08  7.850e+02  2.407e+11
Label   Third location
Header  ID :     LD_INS     SR_INS     L1_HIT    HIT_LFB L2_DRD_REQ L2_DRD_HIT L2_PF_MISS L2_RFO_MIS   [HW B/s]
Rank     0 :  1.029e+09  1.546e+06  8.408e+08  1.836e+08  1.066e+08  6.147e+07  1.469e+08  7.720e+02  2.403e+11
Rank     1 :  1.028e+09  1.473e+06  8.403e+08  1.836e+08  1.067e+08  6.149e+07  1.469e+08  7.770e+02  2.407e+11
Rank     2 :  1.028e+09  1.448e+06  8.402e+08  1.835e+08  1.066e+08  6.150e+07  1.469e+08  7.280e+02  2.407e+11
Rank     3 :  1.028e+09  1.454e+06  8.402e+08  1.835e+08  1.066e+08  6.149e+07  1.469e+08  7.690e+02  2.408e+11
```

■HWPC **統計情報詳細レポート**: Intel HWPC_CHOOSER=CACHE **指定例**

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     L1_TCM     L2_TCM     L3_TCM    OFFCORE
Rank     0 :  3.754e+08  1.759e+08  1.131e+05  3.918e+08
Rank     1 :  3.754e+08  1.760e+08  1.114e+05  3.914e+08
Rank     2 :  3.754e+08  1.759e+08  1.147e+05  3.917e+08
Rank     3 :  3.754e+08  1.758e+08  1.128e+05  3.916e+08
Label   Second location
Header  ID :     L1_TCM     L2_TCM     L3_TCM    OFFCORE
Rank     0 :  1.251e+08  5.855e+07  7.610e+02  1.307e+08
Rank     1 :  1.251e+08  5.859e+07  7.610e+02  1.306e+08
Rank     2 :  1.251e+08  5.856e+07  7.160e+02  1.307e+08
Rank     3 :  1.251e+08  5.852e+07  7.100e+02  1.307e+08
Label   Third location
```

```
Header  ID :     L1_TCM     L2_TCM     L3_TCM    OFFCORE
Rank     0 :  1.251e+08  5.854e+07  5.610e+02  1.307e+08
Rank     1 :  1.251e+08  5.855e+07  5.270e+02  1.306e+08
Rank     2 :  1.251e+08  5.852e+07  5.280e+02  1.307e+08
Rank     3 :  1.251e+08  5.850e+07  5.110e+02  1.307e+08
```

■HWPC 統計情報詳細レポート: Intel HWPC_CHOOSER=CYCLE **指定例**

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :    TOT_CYC    TOT_INS
Rank     0 :  3.262e+09  5.930e+09
Rank     1 :  3.282e+09  5.936e+09
Rank     2 :  3.277e+09  5.929e+09
Rank     3 :  3.218e+09  5.919e+09
Label   Second location
Header  ID :    TOT_CYC    TOT_INS
Rank     0 :  1.057e+09  1.963e+09
Rank     1 :  1.057e+09  1.963e+09
Rank     2 :  1.056e+09  1.963e+09
Rank     3 :  1.051e+09  1.960e+09
Label   Third location
Header  ID :    TOT_CYC    TOT_INS
Rank     0 :  1.057e+09  1.964e+09
Rank     1 :  1.056e+09  1.963e+09
Rank     2 :  1.056e+09  1.963e+09
Rank     3 :  1.051e+09  1.960e+09
```

■HWPC 統計情報詳細レポートの HWPC **記号説明**: Intel Xeon **サーバ**

legend 引数を 1 に指定した場合に出力される。

```
    Detected CPU architecture:
        GenuineIntel
        Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
        The available PMlib HWPC events for this CPU are shown below.
        The values for each process as the sum of threads.
    HWPC events legend:
        FP_OPS: floating point operations
        SP_OPS: single precision floating point operations
        DP_OPS: double precision floating point operations
        VEC_SP: single precision vector floating point operations
        VEC_DP: double precision vector floating point operations
        LD_INS: memory load instructions
        SR_INS: memory store instructions
        L1_HIT: level 1 cache hit
        L2_HIT: level 2 cache hit
        L3_HIT: level 3 cache hit
        HIT_LFB cache line fill buffer hit
        L1_TCM: level 1 cache miss
        L2_TCM: level 2 cache miss
        L3_TCM: level 3 cache miss by demand
        OFFCORE: demand and prefetch request cache miss
        TOT_CYC: total cycles
        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
```

6.2.3.2　京コンピュータでの出力例

京コンピュータで PMlib の HWPC 統計情報詳細レポート出力例を HWPC_CHOOSER=FLOPS, BANDWIDTH, CACHE, CYCLE の各場合について示す。

■HWPC 統計情報詳細レポート: 京コンピュータ HWPC_CHOOSER=FLOPS 指定例

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     FP_OPS    [Flops]
Rank     0 :  1.216e+10  1.738e+10
Rank     1 :  1.216e+10  1.743e+10
Label   Second location
Header  ID :     FP_OPS    [Flops]
Rank     0 :  4.033e+09  1.732e+10
Rank     1 :  4.033e+09  1.732e+10
Label   Third location
Header  ID :     FP_OPS    [Flops]
Rank     0 :  4.033e+09  1.729e+10
Rank     1 :  4.033e+09  1.737e+10
```

■HWPC 統計情報詳細レポート: 京コンピュータ HWPC_CHOOSER=BANDWIDTH 指定例

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     L2_TCM   L2_WB_DM   L2_WB_PF   [HW B/s]
Rank     0 :  1.195e+08  3.810e+02  1.533e+05  2.190e+10
Rank     1 :  1.180e+08  9.800e+02  3.767e+05  2.174e+10
Label   Second location
Header  ID :     L2_TCM   L2_WB_DM   L2_WB_PF   [HW B/s]
Rank     0 :  4.443e+07  5.300e+01  3.990e+04  2.434e+10
Rank     1 :  3.719e+07  4.900e+01  3.194e+04  2.037e+10
Label   Third location
Header  ID :     L2_TCM   L2_WB_DM   L2_WB_PF   [HW B/s]
Rank     0 :  3.968e+07  2.200e+01  3.744e+04  2.188e+10
Rank     1 :  3.846e+07  3.200e+01  3.476e+04  2.124e+10
```

■HWPC 統計情報詳細レポート: 京コンピュータ HWPC_CHOOSER=CACHE 指定例

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   First location
Header  ID :     L1_TCM     L2_TCM
Rank     0 :  2.433e+08  1.209e+08
Rank     1 :  2.433e+08  1.045e+08
Label   Third location
Header  ID :     L1_TCM     L2_TCM
Rank     0 :  8.106e+07  5.108e+07
Rank     1 :  8.105e+07  2.987e+07
Label   Second location
Header  ID :     L1_TCM     L2_TCM
Rank     0 :  8.106e+07  4.804e+07
Rank     1 :  8.106e+07  3.178e+07
```

■HWPC 統計情報詳細レポート: 京コンピュータ HWPC_CHOOSER=CYCLE 指定例

```
# PMlib hardware performance counter (HWPC) Report -------------------------
```

```
Label   First location
Header  ID :    TOT_CYC    TOT_INS     LD_INS     SR_INS
Rank     0 :  5.612e+09  1.194e+10  6.007e+09  1.805e+07
Rank     1 :  5.593e+09  1.194e+10  6.007e+09  1.805e+07
Label   Third location
Header  ID :    TOT_CYC    TOT_INS     LD_INS     SR_INS
Rank     0 :  1.861e+09  3.972e+09  2.002e+09  5.016e+06
Rank     1 :  1.857e+09  3.972e+09  2.002e+09  5.016e+06
Label   Second location
Header  ID :    TOT_CYC    TOT_INS     LD_INS     SR_INS
Rank     0 :  1.848e+09  3.972e+09  2.002e+09  5.016e+06
Rank     1 :  1.843e+09  3.972e+09  2.002e+09  5.016e+06
```

■HWPC 統計情報詳細レポートの HWPC 記号説明: 京コンピュータ

legend 引数を 1 に指定した場合に出力される。

```
    Detected CPU architecture:
        Sun
        Fujitsu SPARC64 VIIIfx
        The available PMlib HWPC events for this CPU are shown below.
        The values for each process as the sum of threads.
    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
```