PMlib - Performance Monitor Library

利用説明書

Ver. 5.1

July 2016

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

Release

Version 5.1	2016-07-07
Version 5.0	2016-05-16
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

目次

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などがあるが、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 を利用して統計情報をレポートするには以下の手順をふむ。

1. PMlib をインストールする。
   利用者が自分用のライブラリとして個別にインストールすることも、
   管理者がシステムライブラリとしてインストールすることも可能である。
2. アプリケーションへ PMlib を組み込む。
   利用者アプリケーションのソースプログラムに PMlib API を組み込む。
3. アプリケーションを実行する。
   アプリケーションを実行する際に環境変数を用いて PMlib の出力をコントロールすることが可能。

PMlib は以下のプログラムモデルに対応し、C++ および Fortran プログラム言語 API が準備されている。

- 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

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/FX100 計算ノード
  - 富士通コンパイラ＋富士通 MPI
- Intel Xeon クラスタ (RedHat 6.4 以降、Suse 11.3 以降を推奨)
  - Intel コンパイラ＋ IntelMPI
  - GNU コンパイラ +OpenMPI/gnu
  - PGI コンパイラ +OpenMPI/pgi
- Apple Macbook (Mac OSX 10.11 以降を推奨)
  - Apple Clang/LLVM コンパイラ＋ OpenMPI
  - GNU コンパイラ＋ OpenMPI

Pmlib のインストールオプションにより以下のソフトウエアが必要になる。

- HWPC 測定機能を利用したい場合は PAPI ライブラリ 5.0 以降
- OTF 出力機能を利用したい場合は OTF ライブラリ 1.12 以降

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。

1. 初期化機能測定区間の初期化・プロパティ設定を行う。
2. 計算性能測定機能指定した測定区間を実行して計算量を測定する。
3. レポート機能各区間の統計情報をレポート出力する。

ソースプログラム中に各機能に対応する PMlib API（関数・サブルーチン）を加えて PMlib とリンクしたアプリケーションを実行することにより、性能統計情報を取得する。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、各機能に対応する具体的な関数名と仕様は第 4 章、第 5 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、区間の経過時間はそれらの合計となる。区間の計算量も同様である。後出のトレース情報分析オプションを用いると、区間の情報を合計するのではなく、各測定値を全て記録・出力することができる。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もしれない。備忘録 以降の章でも同様

■排他測定と非排他測定
測定区間は「排他測定区間」と「非排他測定区間」とに分類される。排他測定では対象とする区間が他の区間とオーバーラップしないことを前提とし、主として「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、「全排他測定区間の実行時間合計」に対する「個々の排他測定区間の実行時間」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。

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、計算量の内容 ([通信] か [計算]) は、測定区間のプロパティ登録時に設定する計算量タイプ指定により決定される。

「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ハードウエア性能カウンター (HWPC) の統計情報を用いる。実行するシステムが (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カウンターグループ環境変数 HWPC_CHOOSER で指定する。HWPC_CHOOSER で指定可能な値は以下。

- 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選択は、

- 環境変数 HWPC_CHOOSER の値、
- PMlib 関数 setProperties() の type 引数の値、
- PMlib 関数 stop() の fP 引数の値、

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

HWPC が利用できないシステムにおいては常に自己申告モードが選択される。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。レポートの種類にはテキストベースの基本レポート、詳細レポートがある。また、測定情報の時刻歴トレース可視化のためのポスト処理用ファイル出力機能がある。

- テキストベースの基本レポート・詳細レポート
- ポスト処理可視化用ファイル

レポートやポスト処理ファイル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。

テキストレポート機能の詳細については第 6 章、ポスト処理可視化用ファイルの出力は、第 7 章可視化処理ソフトウエアとの連携については第 8 章で説明される。

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（下図グレーの部分）である。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。

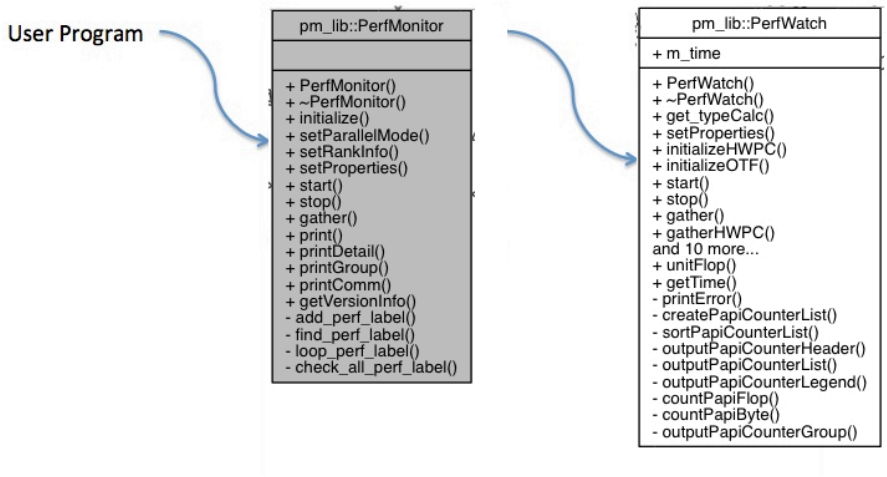

図 3.1　PMlib クラス呼び出し構図

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_otf.h
  - pmlib_papi.h
- HWPC/PAPI インタフェイスソースファイル src_papi_ext/
  - papi_ext.c
- ポスト処理可視化用 OTF インタフェイスソースファイル src_otf_ext/
  - otf_ext.c

3.2　PMlib パッケージのインストール手順

定期的に開催されている計算科学研究機構主催の PMlib 講習会の実習編資料「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。次節に示す PMlib パッケージに同じ内容の説明用スライド資料が含まれており、実際のインストールもその手順に従って実施するのが良い。

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 パッケージはソースプログラム形式で参考資料と共に公開されている。
PMlib のリリース版は以下の公開レポジトリから入手可能。
`http://avr-aics-riken.github.io/PMlib/`

tar.gz 形式のファイルでダウンロードする場合、ダウンロードされる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。

また、最新の開発版パッケージは以下の開発用リポジトリから入手可能。
`https://github.com/avr-aics-riken/PMlib`

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
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・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
PAPI_DIR=${PAPI_ROOT}
MPI_DIR=${I_MPI_ROOT}
OTF_DIR=/usr/local/otf/otf-1.12-intel
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${OTF_DIR}/lib

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 \
    --with-comp=INTEL  --with-impi=${I_MPI_ROOT} \
    CXX=mpiicpc CC=mpiicc FC=mpiifort \
    CFLAGS="${CFLAGS}" CXXFLAGS="${CXXFLAGS}" FCFLAGS="${FCFLAGS}" \
    --with-papi=${PAPI_DIR}  --with-otf=${OTF_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。またインストール先となる -prefix=$INSTALL_DIR ディレクトリへは書き込み権限が必要である。

3.2.4　configure コマンドのオプション

上記２つのインストール例で configure コマンドに与えたオプションの内容は以下である。

■configure コマンドのオプションとその内容

```
--prefix=INSTALL_DIR
   Install directory path.  If omitted, default is /usr/local/PMlib.

--with-comp=(INTEL|FJ|GNU)
   If Intel compiler is used, choose INTEL, and if Fujitsu compiler, specify FJ.
   This option is mandatory when using these compilers.

--with-example=(no|yes)
```

```
   Set 'yes' to build example programs. Default setting is 'no'.

--with-impi=IntelMPI_DIR
   If you use IntelMPI library, specify this option with IntelMPI_DIR value
   pointing to the IntelMPI library installed directory.
   This option is required only when using native compilers.
   If you use MPI wrapper compiler for CXX, CC, FC, the MPI library is
   automatically detected, and this options can be omitted.

--with-papi=PAPI_DIR or --with-papi=yes
   If you use PAPI library, specify this option with PAPI_DIR value
   pointing to the PAPI library installed directory.
   On K computer compute node, use --with-papi=yes form. K computer has
   different paths between front-end nodes and compute nodes, and PMlib
   configure script automatically detects the path on compute nodes.

--with-otf=OTF_DIR
   If you use OTF library, specify this option with OTF_DIR value
   pointing to the OTF library installed directory.


CXX=CXX_COMPILER
   Specify a native C++ compiler, e.g., g++, icpc, c++ or MPI wrapper
   compiler, e.g. mpicxx.

CXXFLAGS=CXX_OPTIONS
   Specify compiler options.

CC=CC_COMPILER
   Specify a native C compiler, e.g., gcc, icc, clang or MPI wrapper
   compiler, e.g. mpicc.

CFLAGS=CC_OPTIONS
   Specify compiler options.

FC=FORTRAN_COMPILER
   Specify a native fortran compiler, e.g., ifort, gfortran, or MPI wrapper
   compiler, e.g. mpif90.

FCFLAGS=FORTRAN_OPTIONS
   Specify compiler options.
```

configure コマンドに指定可能なオプションはさらに多数ある。全オプションとその内容、さらに各種のインストール・パターンの例が INSTALL ファイルに説明されているので必要に応じて適宜参照いただきたい。

3.2.5 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。C++ プログラムが MPI 並列プログラムの場合には各関数は全てのプロセスから呼び出すことが基本となる。

以下の説明内容を含む詳しいドキュメントが、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)
```

第 1 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)
```

第 1 引数 `label` :ラベル文字列（必須)。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（省略可）(COMM:通信 (省略値)、CALC:計算)

第 3 引数 `exclusiveflag` :測定の排他指定フラグ（省略可)。bool 型 true: 排他測定区間 (省略値)、false: 非排他測定区間

第 2 引数である `type` は明示的な申告モードの場合にのみ意味を持ち、自動算出モードの場合は無視される。`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（必須）。 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, unsigned iterationCount)
```

第 1 引数 label :ラベル文字列（必須）。 測定区間はラベルで識別される。

第 2 引数 flopPerTask :計算量（省略可）。演算量 (Flop) 又は通信量 (バイト)

第 3 引数 iterationCount :計算量に乗じる係数（省略可）。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。同一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。

■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（必須）。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（省略可）(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（省略可）

第 4 引数 seqSections :測定区間の表示順（省略可）(0:(省略時) 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。必要に応じて内部で全プロセスの測定結果情報をマスタープロセスに集約し、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。基本統計レポート、あるいは下記の詳細レポートを出力した時点で全ての測定区間の PMlib 測定は終了する。

■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（必須）。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（省略可）(0:なし、1:表示する)

第 3 引数 seqSections :測定区間の表示順 （省略可）(0:経過時間順にソート後表示、1:登録順で表示)

各排他的測定区間毎に、全 MPI ランクの経過時間情報と測定された計算量の統計レポートを出力する。必要に応じて内部で全プロセスの測定結果情報をマスタープロセスに集約し、計算量が自動算出モードで HWPC 統計情報が取得された場合は、HWPC 値も出力する。HWPC のイベント統計値は各プロセス毎に OpenMP・自動並列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（必須）。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（必須）
第 3 引数 p_comm :MPI_Comm 型 group に対応する communicator（必須）
第 4 引数 pp_ranks :group を構成する rank 番号配列へのポインタ（必須）
第 5 引数 group :プロセスグループ番号（必須）（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（省略可）(0:なし、1:表示する)
第 7 引数 seqSections :測定区間の表示順（省略可）(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。

■レポートの出力：MPI_Comm_split で自動グループ化した単位毎の MPI ランク別詳細レポート

```
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（必須）。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 （必須）
第 3 引数 icolor :MPI_Comm_split() のカラー変数（必須）
第 4 引数 key :MPI_Comm_split() の key 変数（必須）
第 5 引数 legend :HWPC 記号説明の表示フラッグ（省略可）(0:なし、1:表示する)
第 6 引数 seqSections :測定区間の表示順 （省略可）(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。

■レポートの出力：途中経過レポート

```
void printProgress(FILE* fp, const std::string comments, int seqSections);
```

第 1 引数 fp :出力ファイルポインタ（必須）。標準出力にレポートする場合は stdout を指定。
第 2 引数 comments :任意のコメント
第 3 引数 seqSections :測定区間の表示順（省略可）(0:(省略時) 経過時間順にソート後表示、1:登録順で表示)
基本レポートと同様なフォーマットで排他的測定区間の途中経過をレポート出力する。この API は繰り返し呼び出して利用することができる。多数回の反復計算を行う様なプログラムにおいて初期の経過状況を少数回モニターする場合などに利用することを想定している。

■ポスト処理用 trace ファイルの出力

```
void postTrace(void);
```

ポスト処理用 trace ファイルを出力する。現在サポートしている trace ファイルフォーマットは OTF(Open Trace Format) v1.1。trace ファイルの出力はプログラム実行中一回のみ可能である。

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。

```
int main (int argc, char *argv[])
{
    for (int i=0; i<3; i++)
    {
      compute1();                  // 関数１：(測定区間"Block-1")
      compute2();                  // 関数２：(測定区間"Block-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。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 4×10^{10} であると知られている場合は以下の様に引数指定する。

```
#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

    for (int i=0; i<3; i++)
    {
      PM.start("Block-1");                                  // 3. 測定区間"Block-1"の実行開始
      compute1();
      PM.stop ("Block-1", 1.0e10);                    //   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", 4.0e10);                    //   測定区間"Block-2"の実行終了
    }

    PM.print(stdout);                                     // 4.  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

    for (int i=0; i<3; i++)
    {
      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.print(stdout);                                     // 4. 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　C++ 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 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
- (HWPC/PAPI ライブラリを利用する場合）PAPI ディレクトリ名を PAPI_ROOT に設定
- (OTF ライブラリを利用する場合）OTF ディレクトリ名を OTF_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} ${PAPI_LDFLAGS}"
export LD_LIBRARY_PATH=${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}/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 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 プログラムが MPI 並列プログラムの場合には各関数は全てのプロセスから呼び出すことが基本となる。

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 では全ての引数を省略せずに渡す必要がある。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のみ意味を持ち、自動算出モードの場合は無視される。

第 3 引数 (IN) integer `fexclusive` :測定の排他指定フラグ。(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。同一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。

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。

■レポートの出力：基本統計レポート

```
subroutine f_pm_print (fc, psort)
```

第 1 引数 (IN) character*(*) fc :出力ファイル名。もし fc 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。必要に応じて内部で全プロセスの測定結果情報をマスタープロセスに集約し、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。基本統計レポート、あるいは下記の詳細レポートを出力した時点で全ての測定区間の PMlib 測定は終了する。

■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*(*) fc :出力ファイル名。もし fc 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:なし、1:表示する) 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)

各排他的測定区間毎に、全 MPI ランクの経過時間情報と測定された計算量の統計レポートを出力する。必要に応じて内部で全プロセスの測定結果情報をマスタープロセスに集約し、計算量が自動算出モードで HWPC 統計情報が取得された場合は、HWPC 値も出力する。HWPC のイベント統計値は各プロセス毎に OpenMP・自動並列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*(*) fc :出力ファイル名。もし fc 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:なし、1:表示する)

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 で自動グループ化した単位毎の MPI ランク別詳細レポート

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

第 1 引数 (IN) character*(*) fc :出力ファイル名。もし fc 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:なし、1:表示する)

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。

■レポートの出力：測定途中経過レポートを出力

```
subroutine f_pm_printprogress (fc, comments, psort)
```

第 1 引数 (IN) character*(*) fc :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) character*(*) comments :任意のコメント (文字列)

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)

基本レポートと同様なフォーマットで排他的測定区間の途中経過をレポート出力する。この API は繰り返し呼び出して利用することができる。多数回の反復計算を行う様なプログラムにおいて初期の経過状況を少数回モニターする場合などに利用することを想定している。

■ポスト処理用 trace ファイルの出力

```
subroutine f_pm_posttrace ()
```

引数なし。ポスト処理用 trace ファイルを出力する。現在サポートしている trace ファイルフォーマットは OTF(Open Trace Format) v1.1。trace ファイルの出力はプログラム実行中一回のみ可能である。

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。

```
program check
do i=1,3
  call compute1 ()                                    ! サブルーチン compute1
  call compute2 ()                                    ! サブルーチン compute2
end do
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。サブルーチン compute1() 内での演算量が 10^{10}、compute2() 内でのデータ移動量が 4×10^{10} であると知られている場合は以下の様に引数指定する。

```
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

do i=1,3
  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", 4.0e10, 0)                 ! 測定区間"Block-2"の終了
end do

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

do i=1,3
  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
end do

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　Fortran 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

- (HWPC/PAPI ライブラリを利用する場合）PAPI ディレクトリ名を PAPI_ROOT に¿設定
- (OTF ライブラリを利用する場合）OTF ディレクトリ名を OTF_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　テキストレポート出力

PMlib が出力する統計情報には、テキストレポートとポスト処理用ファイルとがあり、それらの出力は API、あるいは環境変数で指定する。
この章ではテキストレポートの機能を説明する。

6.1　テキストレポートの種類

PMlib はプログラムの全測定区間終了後に測定結果統計情報をテキストレポートとして出力することができる。テキストレポートの種類は 4 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。
- 経過状況レポート:
  計算時間が非常に長いジョブで、途中の早い時点で測定経過を確認したい　場合などに有効。排他的測定区間の経過状況が、基本統計レポートと同様なフォーマットでレポートされる。

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
```

第 7 章　ポスト処理用ファイル出力

PMlib は統計情報の詳細な可視化処理を行うためのポスト処理用ファイル出力機能を持っている。この章ではポスト処理用ファイルの出力指定方法を説明する。

7.1　ポスト処理ファイルの種類

PMlib は以下の種類のポスト処理ファイルをサポートしている。

- Open Trace Format (OTF) バージョン 1:

OTF は Dresden 工科大学などが中心となってトレース情報フォーマットの共通化を図ったものである。PMlib は http://wwwpub.zih.tu-dresden.de/ jurenz/otf/api/current/ および http://www.paratools.com/otf/specification.pdf で公開されている Open Trace Format API Specification Version 1.1 仕様による出力をサポートしている。以下の説明では PMlib が OTF オプション (–with-otf) つきでインストール済みであることを前提としている。

7.2　OTF ファイル出力の指定方法

PMlib をリンクした利用者プログラムから OTF ファイルを出力するためにはプログラム内で、第 4 章で示した void postTrace(void); あるいは第 5 章で示した subroutine f_pm_posttrace () を呼び出す。プログラムを実行する場合に以下の環境変数を指定して OTF ファイル出力の制御をすることが可能である。

- 環境変数 OTF_TRACING
- 環境変数 OTF_FILENAME

これらの環境変数が取り得る値とその意味合いを以下に示す。

- OTF_TRACING=[off | on | full]
  OTF_TRACING の値により OTF ファイルの出力内容が決定される。
  ”off” : OTF ファイルを出力しない。
  ”on” : 測定区間の全呼び出し回数に対して開始・終了の経過時刻を含む OTF ファイルを出力する。
  ”full” : 測定区間の全呼び出し回数に対して開始・終了の経過時刻、終了時刻における計算量情報を含む OTF ファイルを出力する。
  PMlib から出力される OTF ファイルには以下の 3 種類がある。
  – ${OTF_FILENAME}.otf
  – ${OTF_FILENAME}.0.def
  – ${OTF_FILENAME}.[1|2|..|N].events
  OTF_TRACING=”full” を指定した場合、測定区間の呼び出し回数に対応して計算量を抽出・ファイル出力するための時間的なオーバーヘッドが過剰にかかるケースがありうるので、注意を要する。
- OTF_FILENAME=”ファイル名”
  OTF_FILENAME は上記の OTF ファイル名を決定するために用いられる。
  前述の環境変数 OTF_TRACING の値が”on”又は”full”の時にのみ意味を持ち、ファイル名の先頭部分として用いられる。

OTF_FILENAME の デフォルト値は ”pmlib_optional_otf_files” である。

前述した postTrace() 関数、f_pm_posttrace () サブルーチンの動作はこれらの環境変数のみによってその動作が制御されるため、多くの場合はプログラム中に呼び出しを記述しておいて、OTF ファイルを出力したい時だけ環境変数 OTF_TRACING を設定してプログラムを実行するというような使い方が可能である。

7.3　Intel サーバ上での OTF ファイル出力ジョブ例

■OTF 出力ジョブ例 1、 測定区間の経過時間出力

```
OTF_DIR=/usr/local/otf/otf-1.12-intel
LDFLAGS+=" -L${OTF_DIR}/lib -lopen-trace-format -lotfaux "
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${OTF_DIR}/lib

SRC_DIR=${HOME}/pmlib/scripts/src_tests
WKDIR=/media/dali/data1/mikami/check_pmlib
mkdir -p $WKDIR
cd $WKDIR; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi

cp $SRC_DIR/main_pmlib.cpp main.cpp
cp $SRC_DIR/sub_kernel.c .
cp $SRC_DIR/sub_copy.c .
CFLAGS="-O3 -openmp ${REPORTS}"
mpicxx -c ${CFLAGS} ${INCLUDES} main.cpp
mpicc  -c ${CFLAGS} ${INCLUDES} sub_kernel.c
mpicc  -c ${CFLAGS} ${INCLUDES} sub_copy.c
mpicxx    ${CFLAGS} ${INCLUDES} main.o sub_kernel.o sub_copy.o ${LDFLAGS}

NPROCS=4
export OMP_NUM_THREADS=2
export OTF_TRACING=on
sleep 1
mpirun -np ${NPROCS} ./a.out
ls -go
```

■OTF 出力ジョブ例 2、 測定区間の経過時間、HWPC 統計計算量 (Flops) 情報の詳細な出力

```
PAPI_DIR=/usr/local/papi/papi-5.3.2/intel
LDFLAGS+=" -L${PAPI_DIR}/lib -lpapi -lpfm "
OTF_DIR=/usr/local/otf/otf-1.12-intel
LDFLAGS+=" -L${OTF_DIR}/lib -lopen-trace-format -lotfaux "
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${OTF_DIR}/lib:${PAPI_DIR}/lib

SRC_DIR=${HOME}/pmlib/scripts/src_tests
WKDIR=/media/dali/data1/mikami/check_pmlib
mkdir -p $WKDIR
cd $WKDIR; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi

cp $SRC_DIR/main_pmlib.cpp main.cpp
cp $SRC_DIR/sub_kernel.c .
cp $SRC_DIR/sub_copy.c .
CFLAGS="-O3 -openmp ${REPORTS}"
mpicxx -c ${CFLAGS} ${INCLUDES} main.cpp
mpicc  -c ${CFLAGS} ${INCLUDES} sub_kernel.c
mpicc  -c ${CFLAGS} ${INCLUDES} sub_copy.c
mpicxx    ${CFLAGS} ${INCLUDES} main.o sub_kernel.o sub_copy.o ${LDFLAGS}

NPROCS=4
export OMP_NUM_THREADS=2
export OTF_TRACING=full
export OTF_FILENAME="extra_otf_files"
export HWPC_CHOOSER=FLOPS
sleep 1
```

```
mpirun -np ${NPROCS} ./a.out
ls -go
```

第 8 章　可視化処理ソフトウエアとの連携

第 7 章で示した方法により Open Trace Format (OTF) バージョン 1 フォーマットのポスト処理用ファイルを出力することができる。OTF 1.0 をサポートする各種の可視化ソフトウエアを用いてこのポスト処理用ファイルを処理すると、測定区間の開始時刻、終了時刻、測定量（計算量）の種類、計算量の情報、を全ての区間呼び出しイベント毎のトレース情報（時刻歴情報）として詳細に可視化することが可能である。

これら可視化ソフトウエアを用いた具体的な可視化手順は可視化ソフトウエア各々の利用者向けマニュアルに詳細に記載されており、本書で詳しい説明を加えることはないが、この章ではその代表的な例として、Vampir、および TRAiL を用いたトレース情報の可視化手順の概略のみを紹介する。

8.1　Vampir によるトレース情報可視化の概要

Vampir は高機能な性能統計可視化ソフトウエアとして知られ Dresden 工科大学が開発・ライセンス販売する。

プログラムの起動時に PMlib が出力した OTF マスターファイルを引数で指定するとデフォルトで Master Timeline が表示された状態で GUI が立ち上がるため、その後簡単な手順で性能統計情報の可視化を行うことができる。

■Vampir 起動スクリプト例

```
#! /bin/bash
export PATH="/usr/local/Vampir/8.4.0/bin/:$PATH"
cd ${HOME}/pmlib_result_dir
if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
vampir pmlib_optional_otf_files.otf
```

このスクリプトを実行すると下図の様に Master Timeline が表示された状態で Vampir の GUI が立ち上がる。この例は PMlib のリリースパッケージに含まれる example/test1 プログラムを実行した結果得られる OTF ファイルを入力として用いた場合である。

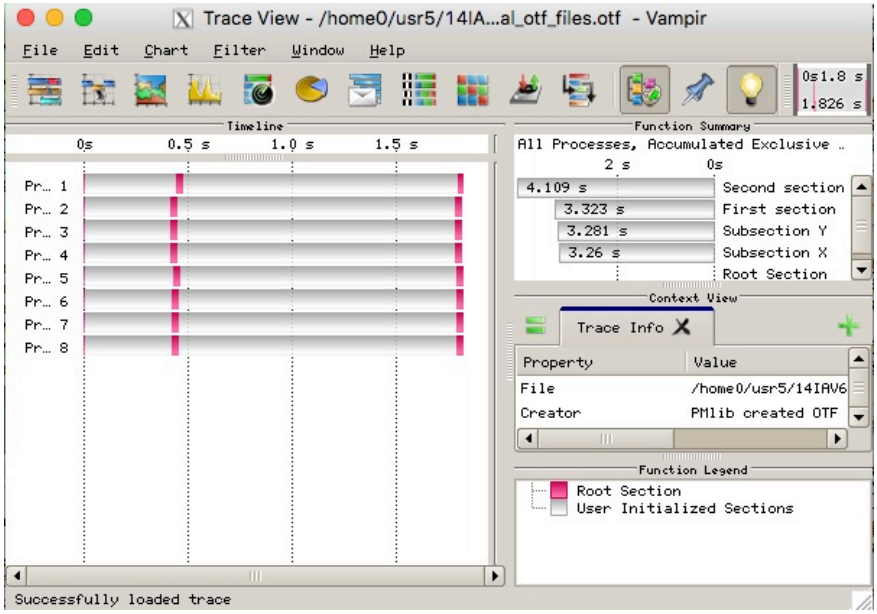

図 8.1　Vampir の GUI 起動

GUI 上部のアイコンメニュー (下図) から以下のサブパネルを適宜選択表示する。

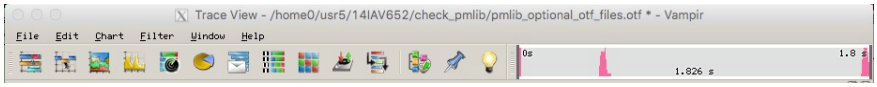

図 8.2　GUI 上部のアイコンメニュー

選択したサブパネル:

- Add Process Timeline
- Add Summary Timeline
- Add Counter Data Timeline
- Add Performance Rader

Counter Data Timeline が表示されたサブパネルの丈夫テキストエリアで右クリックするとカスケードメニューが表示される。その中から Select Meric がをクリックすると、PMlib が出力したカウンター情報として

- User Defined CALC sections
- User Defined COMM sections

が選択可能な状態で表示されるので、どちらかを選択する。MPI プロセス 0 に対して Add Counter Data Timeline サブパネルを 2 つ表示し、その内一つを CALC(Flops) 表示、他の一つを COMM（Bandwidth) 表示した場合の可視化結果が下図である。

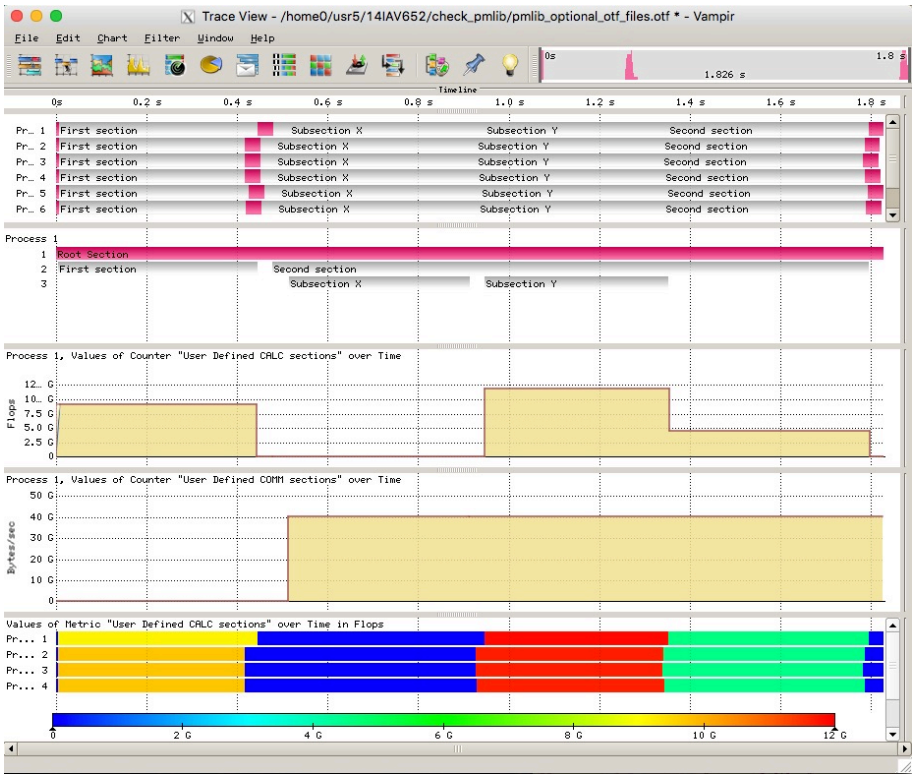

図 8.3　PMlib 出力の CALC/COMM 計算量表示

尚 OTF ではプロセス番号は 1 から付与されるため Vampir ではプロセス 1 と表示されることに注意する。

8.2　TRAiL によるトレース情報可視化の概要

TRAiL は Open Trace Format (OTF) バージョン 1 のファイル入力が可能なトレース情報可視化ソフトウエアであり、Web ブラウザ上での可視化が可能である。オープンソースソフトウエアとして現在理研 AICS が開発を進めている。

図 8.4　TRAiL による Web ブラウザ上での可視化