PMlib 講習会資料

理化学研究所 計算科学研究機構
可視化技術研究チーム
2017年3月8日更新

はじめに

- ゲスト無線LANの利用について（別資料）
- 本日使用する資料（説明スライドなど）
  - https://github.com/mikami3heart/PMlib-tutorials
    - Tutorial-slide1-overview.pdf
    - Tutorial-slide2-installation.pdf
    - PMlib-5.0-API-html.tar.gz
    - scripts.K.tar.gz
- PMlibパッケージの入手・利用方法は、後半の実習編で説明
- （公式な）PMlibパッケージ公開リポジトリ
  - http://avr-aics-riken.github.io/PMlib/
- （最新版が維持される）開発用リポジトリ
  - https://github.com/avr-aics-riken/PMlib

講習会の内容

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

PMlib概要説明

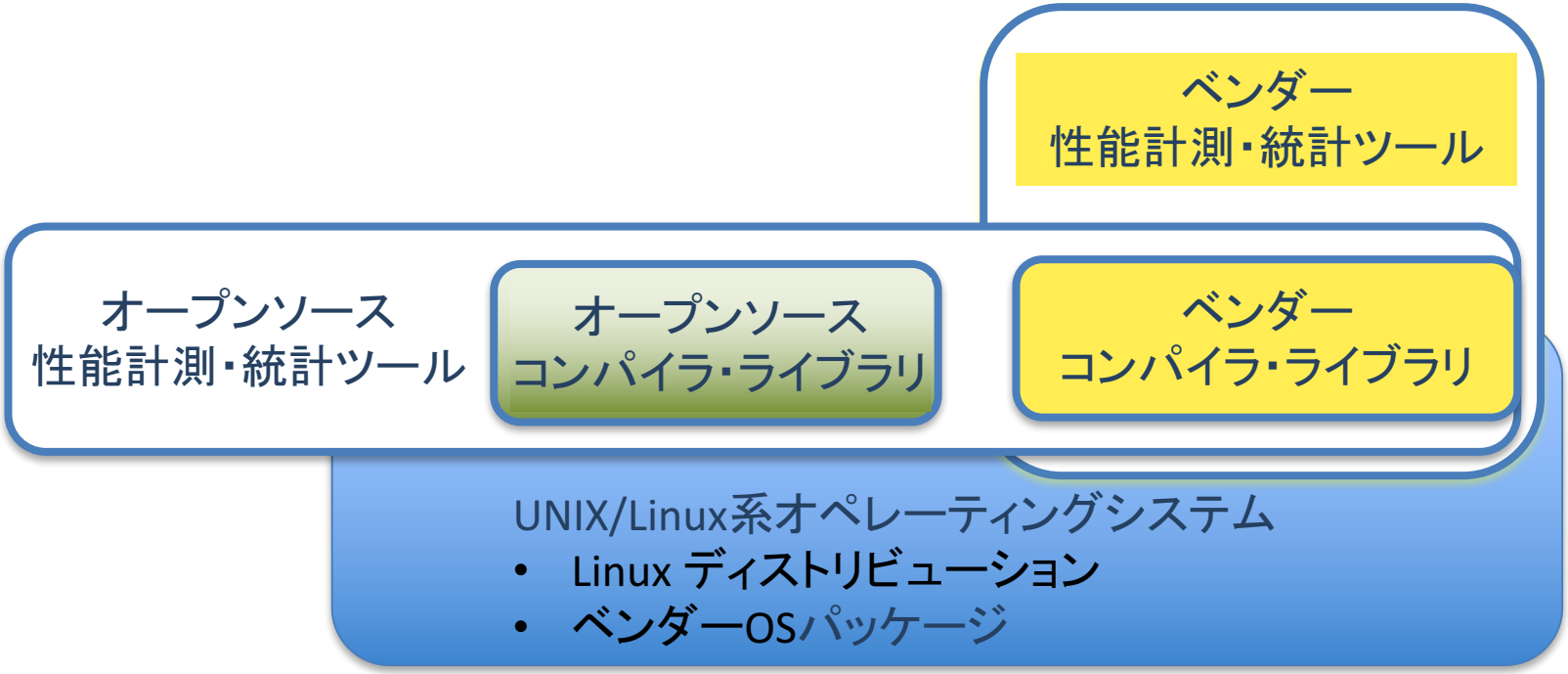
- 性能の可視化
- 開発プロダクツ：PMlib, TRAiL
- ツールの位置づけ：性能統計ツールの位置取りと分類
- PMlibは：性能評価の視点とPMlibの機能概要を説明
- TRAiLは：TRAiLの機能概要を説明
- パーサーツールによるサポート
- 現在の達成状況と今後の計画

性能の可視化

- 性能測定・評価の観点　PMlib
  - 計算科学的観点での性能
    - ソースプログラムで記述された数値計算量を評価
    - アルゴリズムに主眼がおかれる。
  - システム評価的観点での性能
    - HWPCを基にした実行性能(FLOPS)、実効性能(%peak)
    - 特定のシステム・アーキテクチャ・言語処理系SWでの性能に主眼がおかれる

- 性能情報の可視化　TRAiL
  - 静的な統計情報の可視化
    - ジョブの経過時間で平均化された値
  - 動的な性能挙動の可視化
    - ジョブの経過時刻に沿った過渡的な挙動
    - プロセス毎の過渡的な挙動
    - プロセス間の計算負荷バランスの挙動

性能統計ツール



ベンダー
ツール

X86系
- Intel
- PGI

Sparc系
- 富士通
- Oracle

Power系
- 日立/IBM

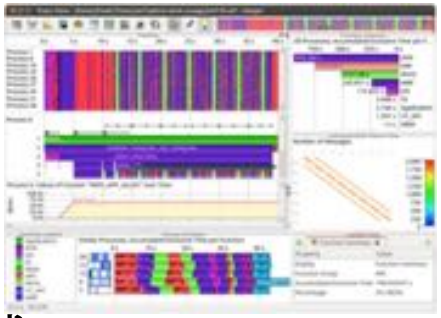
ARM系
- お楽しみ

その他
- Cray
- NVIDIA
- Phi/KNL
- SX

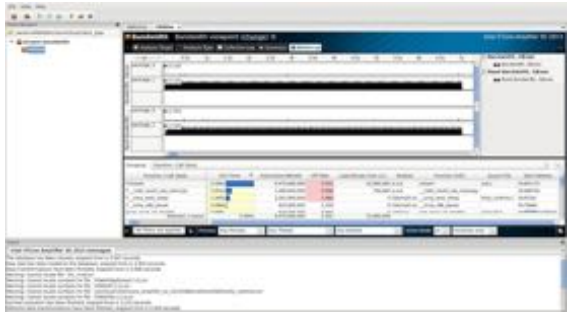
- オープンソース性能統計ツール類
  - Gprof: 簡易機能、コンパイラに制約
  - Scalasca: 高機能、Score-P共通インフラ
  - PAPI : HWPCへのアクセス
  - 他 : 多数

.. and .. PMlib + TRAiL

ベンダー製品とオープンソース



- ベンダー製品の性能計測・統計ツール
  - ○豊富な機能が統合化されたインタフェイス
  - ○パッケージの完成度が高く、インストールが容易
  - ○ 詳しいドキュメント、ベンダーによるサポート、安心感
  - △一般に高価格で相当の習熟期間が必要
  - △選択できるツールがシステム毎に制限される



↑マニュアルHTML 60MB以上

- オープンソースの性能計測・統計ツール
  - ○各ツール毎に特徴・機能が明確
  - ○様々なシステムへの移植可能
  - ○価格の心配が不要
  - △ユーザーインタフェイスが個性的
  - △ものによりインストール・利用がそれなりに大変

PMlib

- アプリケーションの計算性能モニター用のライブラリ
- オープンソース　http://avr-aics-riken.github.io/PMlib
- 主な用途
  - タイマー、計算負荷の概要把握、HWPCへの簡易アクセスなど
- 利用方法
  - アプリのソース中に測定区間を指定。終了時に統計情報を出力
- 利用のモード
  - アプリに常時組み込み、プロダクション利用することを想定
  - 性能測定・性能モデル化の支援にも期待
  - C++とFortranに対応するAPI
- 特徴
  - インストールが簡単。講習会での実績平均10分
  - テキストレポートを基本とするコンパクトなツール
  - 利用のための学習バリアが低い

PMlibによる計算性能のモニターとは？

- 測定区間は少数の属性を持つ
  - ラベル：任意の名称
  - 測定する計算量の種類：「演算量」、「通信量」など＊1
  - 排他性：「排他的」か「非排他的」＊2
- 計算量は時間情報とともに記録される
- 出力情報の評価
  - テキストレポートの数値
  - オプションにより詳細な性能統計情報を出力
    →ポスト処理パッケージ化時刻歴可視化：TRAiL
- そもそも計算量（性能情報）はどう定義するか←ポイント
  - (1) ユーザーが明示的に申告
  - (2) HWPCを用いて自動算出

＊2　他の区間とカブっていないかという意味

計算量: (1) 明示的な申告モード

- 計算科学的視点での性能
- 測定区間の計算量を値や計算式でPMlib APIの引数としてユーザーが明示的に与える
- ソースプログラムの計算式に忠実に沿って評価する場合、あるいは特定の演算種類や粒度毎に重さを設定したい場合に適している
  - 四則演算・基本関数・アルゴリズムのブロックなど
  - ソースプログラムからの計算量の計上方法
    - ユーザーが自分でソースプログラムから計上する
    - パーサーツールを用いて(半自動的に)計上する

計算量：(2) HWPCによる自動算出モード

- システム・アーキテクチャ視点での性能
  - コンパイラによる命令最適化の効果やプロセッサ・階層的メモリレイアウトなどの特性効果を含み、個別のシステム性能の評価に適する
- システムがハードウエア性能カウンタ(HWPC)を装備し、そのイベント統計情報をPAPIライブラリでアクセス可能な場合は、PMlibがPAPI低レベルAPI経由で情報を採取・整理する

- HWPCイベントの種類毎にカウンターグループを定義＊1
  - 実行時の環境変数HWPC_CHOOSERで選択する
    - FLOPS | BANDWIDTH | VECTOR | CACHE | CYCLE

＊1　浮動小数点演算量、メモリ階層間データ移動量、その他HW固有の統計量が選べる

PMlib基本APIの一覧

関数名(C++)	関数名(Fortran)	機能	呼び出し位置と回数	引数
initialize()	f_pm_initialize()	PMlib全体の初期化	冒頭・一回	(1)測定区間数
setProperties()	f_pm_setproperties()	測定区間のラベル化	任意・各区間一回	(1)ラベル、(2)測定対象タイプ、(3)排他指定
start()	f_pm_start()	測定の開始	任意(startとstopでペア)・任意	(1)ラベル
stop()	f_pm_stop()	測定の停止	任意(startとstopでペア)・任意	(1)ラベル、(2)計算量、(3)任意の係数
print()	f_pm_print()	測定区間毎の基本統計結果表示	測定終了時・一回	(1)出力ファイルポインタ、(2)ホスト名、(3)任意のコメント、(4)区間の表示順序指定
printDetail()	f_pm_printdetail()	MPIランク毎の詳細性能情報の表示	測定終了時・一回	(1)出力ファイルポインタ、(2)記号説明の表示、(3)区間の表示順序指定

これだけでとりあえずPMlibの機能を利用できる😀

PMlibを利用するプログラムの構成例（Fortran）

- 元のソース

```
program main

call mykernel()

end
```

注目箇所

- PMlib組み込み後のソース

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

初期設定

測定区間

レポート出力

PMlibを利用するプログラムの構成例（C++）

- 元のソース

```
int main(int argc, char *argv[])
{
mykernel(); //注目箇所
return 0;
}
```

- PMlib組み込み後のソース

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main(int argc, char *argv[])
{
PM.initialize();
PM.setProperties("Koko!", PM.CALC);
PM.start("Koko!");
mykernel();
PM.stop ("Koko!");
PM.print(stdout, "", "");
PM.printDetail(stdout);
return 0;
}
```

PMlibヘッダー

初期設定

測定区間

レポート出力

出力テキストレポート

- １、基本レポート
  - 各測定区間のプロセス平均性能プロファイルを登録順、または経過時間順に出力
  - ジョブあたりの総合性能プロファイル
- ２、詳細レポート
  - 2-1 各MPIプロセス毎のプロファイルを出力
    - プロセスがOpenMPスレッドを発生した場合、各スレッドの計算量は元プロセスに合算する。
  - 2-2各MPIプロセス毎のHWPCイベント統計量
    - HWPCイベントグループを環境変数で指定した場合に出力する。
    - 各スレッドの計算量は元プロセスに合算する(2-1と同様)

基本レポート例 ユーザ申告モード

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

      Timing Statistics Report from PMlib version 5.0.4
      Linked PMlib supports: MPI, OpenMP, HWPC, no-OTF
      Host name : g05-040
      Date      : 2016/06/22 : 01:27:48
      Mrs. Kobe
      Parallel Mode:   Hybrid (2 processes x 4 threads)
      The environment variable HWPC_CHOOSER is not provided. No HWPC report.

      Total execution time            = 2.008230e+00 [sec]
      Total time of measured sections = 2.000537e+00 [sec]

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

      Section           |    call   |        accumulated time[sec]           | [user defined counter values ]
      Label             |           |       avr   avr[%]      sdv   avr/call |       avr       sdv   speed
      ------------------+-----------+----------------------------------------+-------------------------------
      Second section(*) :         1   1.736e+00   86.80  8.58e-03  1.736e+00    2.800e+10  0.00e+00  16.12 Gflops(*)
      Subsection Y      :         3   7.010e-01   35.04  1.05e-03  2.337e-01    1.200e+10  0.00e+00  17.12 Gflops
      Subsection X      :         3   6.988e-01   34.93  7.66e-04  2.329e-01    4.800e+10  0.00e+00  68.69 GB/sec
      First section     :         1   2.311e-01   11.55  6.98e-04  2.311e-01    4.000e+09  0.00e+00  17.31 Gflops
      ------------------+-----------+----------------------------------------+-------------------------------
      Sections per process          9.321e-01     -Exclusive CALC sections-   1.600e+10            17.17 Gflops
      Sections per process          6.988e-01     -Exclusive COMM sections-   4.800e+10            68.69 GB/sec
      ------------------+-----------+----------------------------------------+-------------------------------
      Sections total job            9.321e-01     -Exclusive CALC sections-   3.200e+10            34.33 Gflops
      Sections total job            6.988e-01     -Exclusive COMM sections-   9.600e+10            37.38 GB/sec
```

基本レポート例 HWPCによる自動測定モード

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

      Timing Statistics Report from PMlib version 5.0.4
      Linked PMlib supports: MPI, OpenMP, HWPC, no-OTF
      Host name : g05-040
      Date      : 2016/06/22 : 01:53:20
      Mrs. Kobe
      Parallel Mode:   Hybrid (2 processes x 4 threads)
      The environment variable HWPC_CHOOSER=FLOPS is provided.

      Total execution time            = 2.005677e+00 [sec]
      Total time of measured sections = 1.996694e+00 [sec]

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

      Section           |    call  |       accumulated time[sec]           | [hardware counter byte counts]
      Label             |          |     avr    avr[%]      sdv   avr/call |       avr      sdv   speed
      ------------------+----------+---------------------------------------+-------------------------------
      Second section(*) :        1   1.733e+00  86.77  9.56e-03  1.733e+00    4.603e+09  7.07e-01   2.66 Gflops (*)
      Subsection X      :        3   6.975e-01  34.94  1.18e-03  2.325e-01    1.438e+10  2.12e+00  20.61 Gflops
      Subsection Y      :        3   6.962e-01  34.87  1.60e-03  2.321e-01    1.381e+10  2.12e+00  19.83 Gflops
      First section     :        1   2.309e-01  11.56  1.50e-05  2.309e-01    4.090e+09  4.77e-07  17.71 Gflops
      ------------------+----------+---------------------------------------+-------------------------------
      Sections per process          1.625e+00    -Exclusive CALC sections-  3.228e+10           19.87 Gflops
      ------------------+----------+---------------------------------------+-------------------------------
      Sections total job            1.625e+00    -Exclusive CALC sections-  6.455e+10           39.73 Gflops
```

詳細レポート例(プロセス毎)

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

Label  Subsection Y
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   counter     speed
Rank     0 :        3  3.111e-01   31.7  7.485e-03  1.037e-01  1.568e+10  5.039e+10 Flops (HWPC)
Rank     1 :        3  3.116e-01   31.7  7.000e-03  1.039e-01  1.568e+10  5.032e+10 Flops (HWPC)
Rank     2 :        3  3.186e-01   32.5  0.000e+00  1.062e-01  1.568e+10  4.921e+10 Flops (HWPC)
Rank     3 :        3  3.094e-01   31.5  9.198e-03  1.031e-01  1.567e+10  5.066e+10 Flops (HWPC)
Label  Subsection X
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   counter     speed
Rank     0 :        3  3.111e-01   31.7  4.311e-04  1.037e-01  1.614e+10  5.189e+10 Flops (HWPC)
Rank     1 :        3  3.116e-01   31.7  0.000e+00  1.039e-01  1.615e+10  5.182e+10 Flops (HWPC)
Rank     2 :        3  3.104e-01   31.6  1.143e-03  1.035e-01  1.615e+10  5.202e+10 Flops (HWPC)
Rank     3 :        3  3.094e-01   31.5  2.169e-03  1.031e-01  1.614e+10  5.217e+10 Flops (HWPC)
Label  First section
Header ID  :     call   time[s] time[%]  t_wait[s]  t[s]/call   counter     speed
Rank     0 :        1  1.059e-01   10.8  0.000e+00  1.059e-01  4.809e+09  4.542e+10 Flops (HWPC)
Rank     1 :        1  1.033e-01   10.5  2.578e-03  1.033e-01  4.806e+09  4.653e+10 Flops (HWPC)
Rank     2 :        1  1.034e-01   10.5  2.490e-03  1.034e-01  4.807e+09  4.650e+10 Flops (HWPC)
Rank     3 :        1  1.031e-01   10.5  2.789e-03  1.031e-01  4.805e+09  4.661e+10 Flops (HWPC)
```

詳細レポート例(HWPC/PAPI)

```
# PMlib hardware performance counter (HWPC) Report ------------------------
Label   Subsection Y
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.568e+10  9.500e+01  5.039e+10
Rank     1 :  1.568e+10  9.200e+01  5.032e+10
Rank     2 :  1.568e+10  9.600e+01  4.921e+10
Rank     3 :  1.567e+10  1.020e+02  5.066e+10
Label   Subsection X
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.614e+10  1.030e+02  5.189e+10
Rank     1 :  1.615e+10  1.070e+02  5.182e+10
Rank     2 :  1.615e+10  1.080e+02  5.202e+10
Rank     3 :  1.614e+10  1.130e+02  5.217e+10
Label   First section
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  4.809e+09  2.600e+01  4.542e+10
Rank     1 :  4.806e+09  2.300e+01  4.653e+10
Rank     2 :  4.807e+09  2.400e+01  4.650e+10
Rank     3 :  4.805e+09  2.400e+01  4.661e+10
```

HWPC legend 京コンピュータ

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

HWPC legend Intel Xeon E5系

```
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
```

出力のオプション：ポスト処理用ファイル

- 計算性能情報を時刻歴可視化処理するための機能
- Open Trace Format API Version 1.1 (OTF1) 仕様
  - Dresden工科大学などによるトレース情報フォーマットの共通化
  - http://wwwpub.zih.tu-dresden.de/~jurenz/otf/api/current/
  - http://www.paratools.com/otf/specification.pdf

TRAiL

- Open Trace Format 1 に準拠する性能トレースデータの可視化ツール
- 主な用途
  - PMlibで採取した詳細情報の可視化
  - OTF1準拠データファイルの汎用可視化ツール
- 利用方法
  - PMlib組み込みプログラムは以下の環境変数を指定してOTFファイルを出力可能
    - OTF_TRACING=[off | on | full]
    - OTF_FILENAME=出力ファイル名ヘッダ
  - OTFデータファイルを、Webブラウザ上でポスト処理・表示する

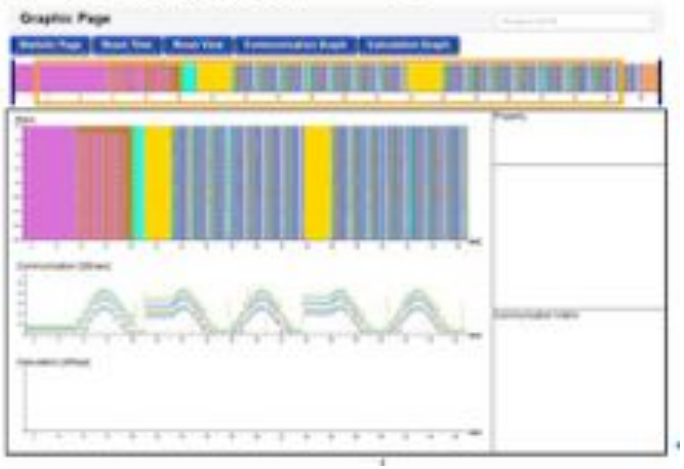
- 利用のモード
  - ユーザー自身のPCにローカルなツールとしてインストール・利用
  - （サーバー上にインストールして分散処理も可能）

TRAiLによる性能情報の可視化

- Webブラウザ上で可視化
  - ジョブの経過時間で平均化された静的な統計情報
  - 動的な性能挙動の可視化
    - ジョブの経過時刻に沿った過渡的な挙動
    - プロセス毎の過渡的な挙動
    - プロセス間の計算負荷バランスの挙動（今後）

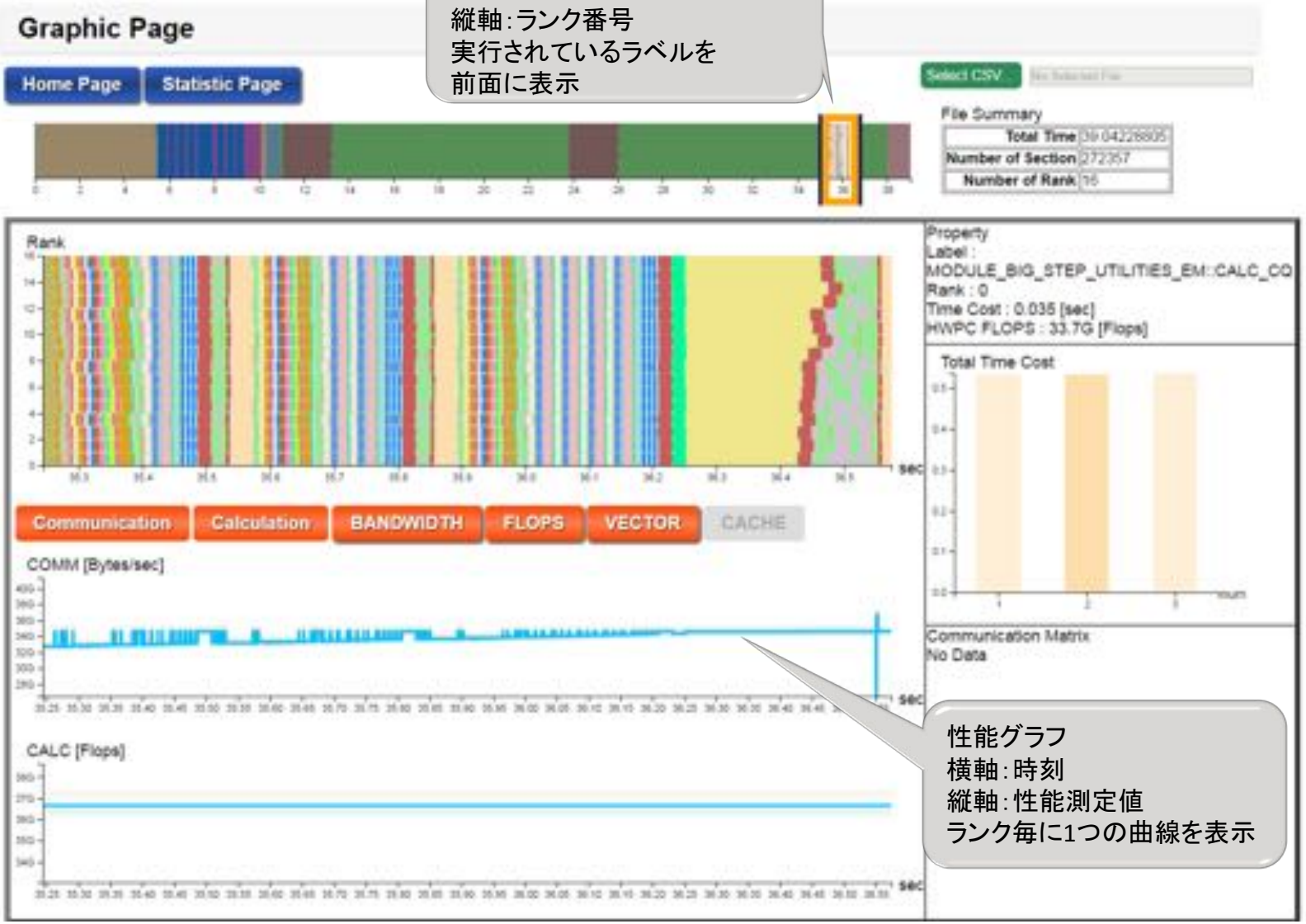
例）統計情報表示ページ

例）ランク毎グラフページ



ランク毎グラフ

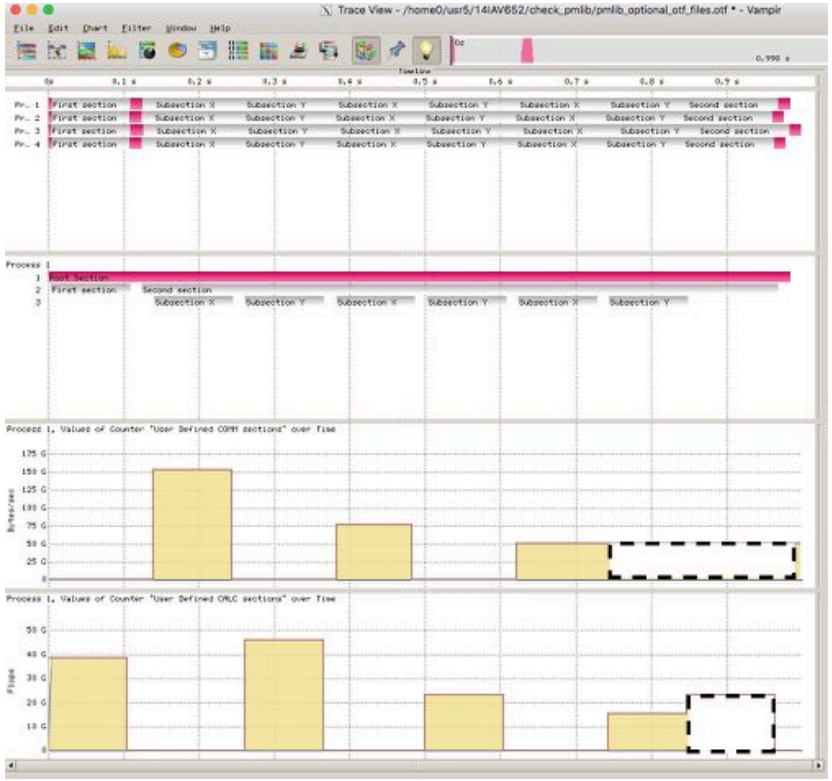
ランク毎処理時刻グラフ
横軸：時刻
縦軸：ランク番号
実行されているラベルを
前面に表示



性能グラフ
横軸：時刻
縦軸：性能測定値
ランク毎に1つの曲線を表示

出力のオプション：ポスト処理用ファイル

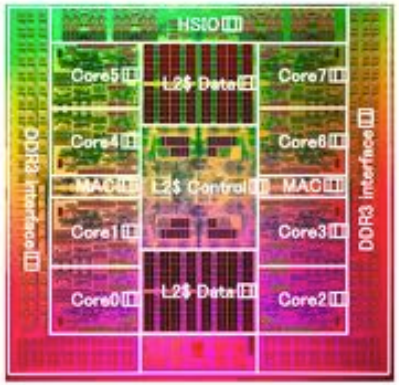
- OTF1をサポートする各種の可視化ソフトウエアで処理　Vampir



- TRAiL（可視化チームで開発中のオープンソースSW）

ハードウエアカウンタ(京計算ノード)

SPARC64™VIIIfx



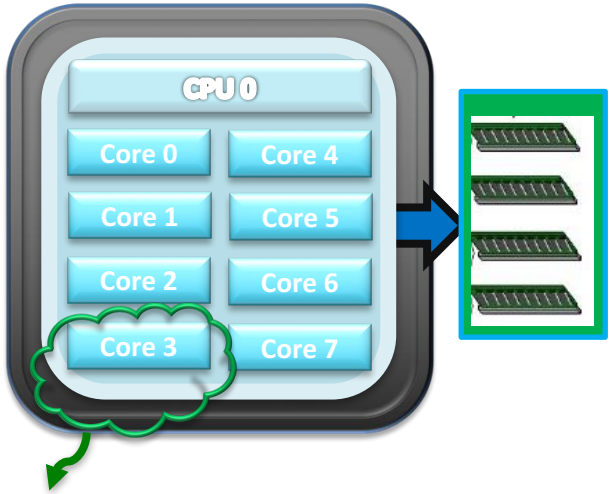
- **Specification**
  - 8 Cores
  - 6 MB Shared L2 Cache
  - FMA×4 (2 SIMD)/core
  - 256 (64bit) DP Reg. /core
  - 2GHz
- **Peak Performance**
  - FP Performance 128GFlop/s
  - Memory Bandwidth 64GB/s
- **Power Consumption**
  - 58W (LINPACK Max)

```
$ papi_avail  -a
Available events and hardware information.
-------------------------------------------------------
Vendor string and code  : Sun (7)
Model string and code  : Fujitsu SPARC64 IXfx (141)
CPU Revision            : 0.000000
CPU Megahertz           : 1650.000000
CPU Clock Megahertz : 1650
CPU's in this Node      : 16
Nodes in this System   : 1
Total CPU's             : 16
Number Hardware Counters : 8
Max Multiplex Counters   : 512
-------------------------------------------------
Of 30 available events, 15 are derived.
```

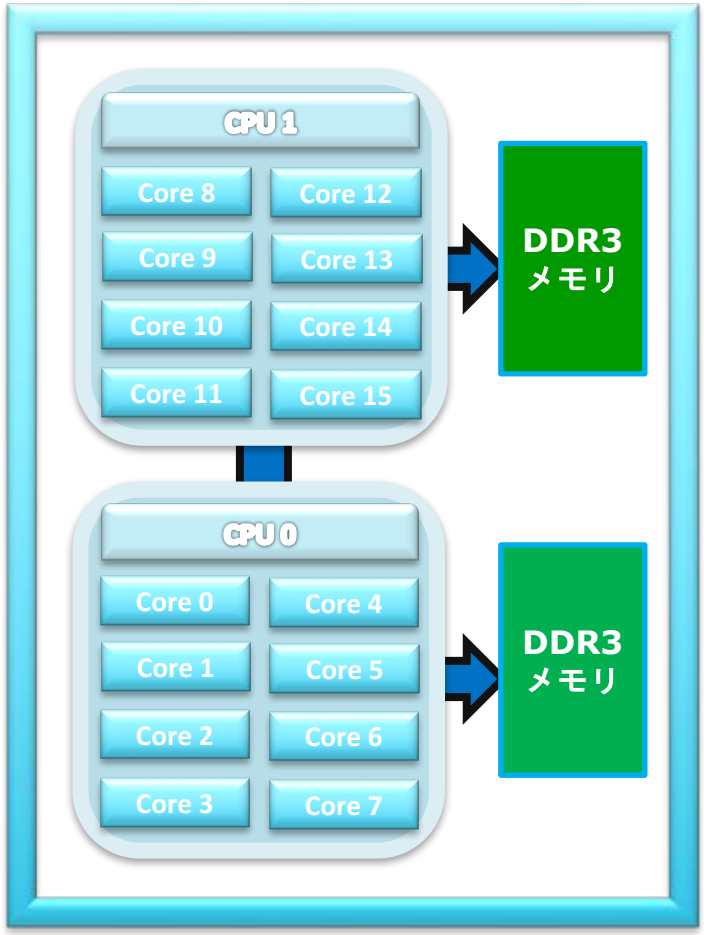
```
    Name        Code    Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No   Level 1 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  Yes  Level 2 cache misses
PAPI_CA_INV  0x8000000c  No   Requests for cache line invalidation
PAPI_CA_ITV  0x8000000d  No   Requests for cache line intervention
PAPI_TLB_DM  0x80000014  No   Data translation lookaside buffer misses
PAPI_TLB_IM  0x80000015  No   Instruction translation lookaside buffer misses
PAPI_TLB_TL  0x80000016  Yes  Total translation lookaside buffer misses
PAPI_MEM_SCY 0x80000022  No   Cycles Stalled Waiting for memory accesses
PAPI_STL_ICY 0x80000025  No   Cycles with no instruction issue
PAPI_FUL_ICY 0x80000026  No   Cycles with maximum instruction issue
PAPI_STL_CCY 0x80000027  Yes  Cycles with no instructions completed
PAPI_FUL_CCY 0x80000028  Yes  Cycles with maximum instructions completed
PAPI_HW_INT  0x80000029  No   Hardware interrupts
PAPI_BR_MSP  0x8000002e  No   Conditional branch instructions mispredicted
PAPI_BR_PRC  0x8000002f  Yes  Conditional branch instructions correctly
predicted
PAPI_FMA_INS 0x80000030  Yes  FMA instructions completed
PAPI_TOT_IIS 0x80000031  Yes  Instructions issued
PAPI_TOT_INS 0x80000032  No   Instructions completed
PAPI_FP_INS  0x80000034  Yes  Floating point instructions
PAPI_LD_INS  0x80000035  Yes  Load instructions
PAPI_SR_INS  0x80000036  Yes  Store instructions
PAPI_BR_INS  0x80000037  No   Branch instructions
PAPI_VEC_INS 0x80000038  Yes  Vector/SIMD instructions
PAPI_TOT_CYC 0x8000003b  No   Total cycles
PAPI_LST_INS 0x8000003c  No   Load/store instructions completed
PAPI_L2_TCH  0x80000056  Yes  Level 2 total cache hits
PAPI_L2_TCA  0x80000059  Yes  Level 2 total cache accesses
PAPI_FP_OPS  0x80000066  Yes  Floating point operations
```

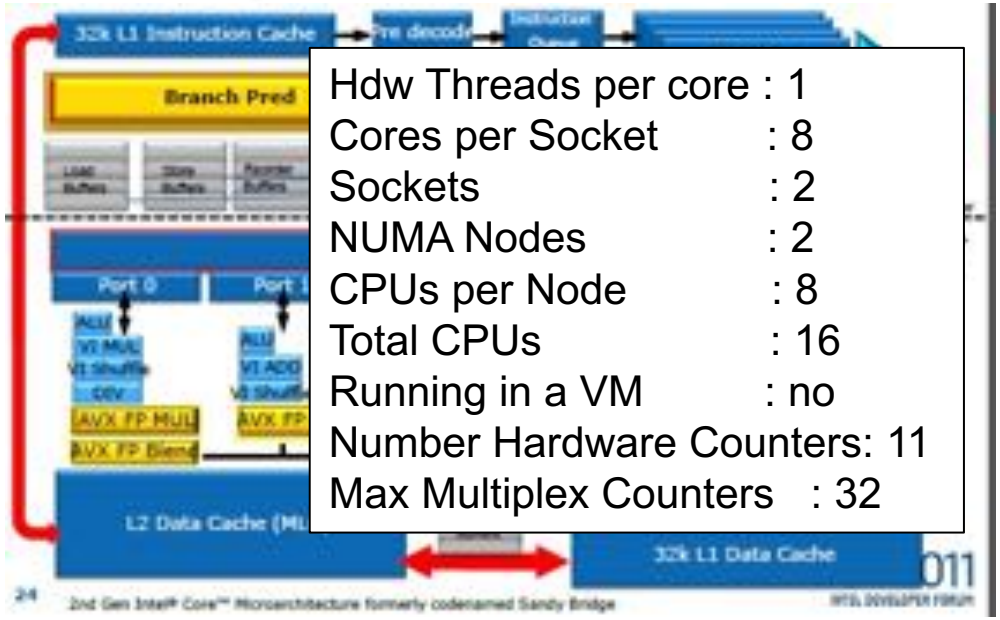
ハードウエアカウンタ（Intel Xeon E5）

CPU : Xeon E5-2670



uncore: L3, QPI, PCIe, …



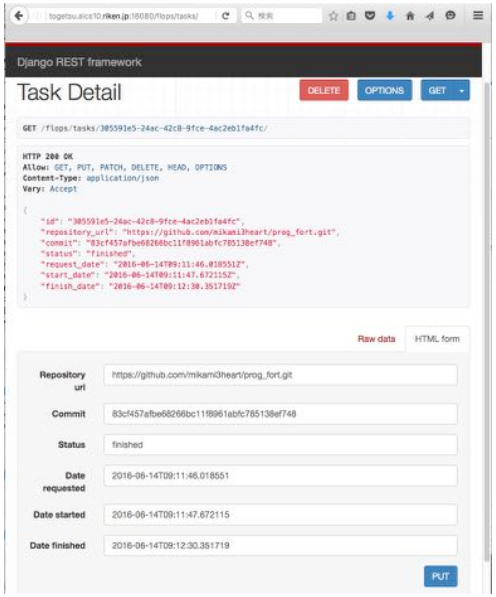


Hdw Threads per core : 1
Cores per Socket : 8
Sockets : 2
NUMA Nodes : 2
CPUs per Node : 8
Total CPUs : 16
Running in a VM : no
Number Hardware Counters: 11
Max Multiplex Counters : 32

Of 50 available events, 17 are derived.

計算量を明示的に申告する場合
👉パーサーツールを利用してPmlib API組み込みを簡便化

- 計算量を値や計算式でPMlib APIの引数としてユーザーが明示的に与える場合、ソースプログラムで記述された演算量や配列アクセス量をパースするツールがとりあえず欲しい。



- Web版（FLOPS-API）
  - 前田チームの試験的Webサービス
  - ソースプログラムのリポジトリ名を入力
  - 分析結果をJSON出力Zipダウンロード

- スタンドアロン版（CCA/EBT）
  - Docker containerを用いてFLOPS APIの機能をパッケージ
  - 前田チーム橋本さんが開発中
  - PMlibの適用に必要な機能盛り込みを協力いただいている

FLOPS-API
（CCA/EBT）

JSONスキーマ

変数名	内容
nfadd	浮動小数点演算数（加算）（+）

```
 1 subroutine sub_fcount(a,b,c,n)
 2 integer :: n
 3 real*8 a(n), b(n), c(n)
 4
 5 a123=a(1)+a(2)+a(3)
 6 b123=b(1)+b(2)+b(3)
 7
 8 do i=1,n-1
 9 c(i)=0.1*(a(i)+b(i))
10 end do
11
12 c(n)=c(n-1)+a123+b123
13 return
14 end
```

元Fortranソース

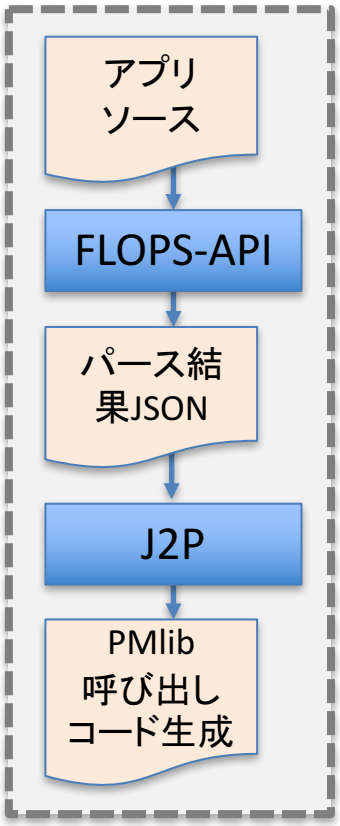
```
"pu": "sub_fcount",
"cat": "subroutine-external-subprogram",
"end_line": 14,
"start_line": 1,
"type": "subroutine",
  途中略
{
    "loc": "check_counts.f90",
    "pu": "sub_fcount",
    "niter": "n - 1",
    "cat": "do-construct",
    "end_line": 10,
    "start_line": 8,
    "type": "loop",
    "children": [
        {
            "loc": "check_counts.f90",
            "pu": "sub_fcount",
            "cat": "do-block",
            "metrics": {
                "nfadd": 1,
                "narefr": 2,
                "nfmul": 1,
                "narefl": 1
            },
  以降略
```

JSONデータ

```
! subroutine sub_fcount(a,b,c,n)
! J2P produced the PMlib API
! for the following source block
!   "start_line": 1,
!   "end_line": 14,
!
! FLOPS counts in the body
    f_pm_start("koko!")
    f_pm_stop("koko!", 6+(n-1)*(1+1), 1)
!
! ARRAY access counts in the body
    f_pm_start("koko!")
    f_pm_stop("koko!", 7+1+(n-1)*(2+1), 1)
```

PMlib APIを出力

J2P




```
 1 subroutine sub_fcount(a,b,c,n)
 2 integer :: n
 3 real*8 a(n), b(n), c(n)
 4
 5 a123=a(1)+a(2)+a(3)
 6 b123=b(1)+b(2)+b(3)
 7
 8 do i=1,n-1
 9 c(i)=0.1*(a(i)+b(i)
10 end do
11
12 c(n)=c(n-1)+a123+b1
13 return
14 end
```

元Fortranソース

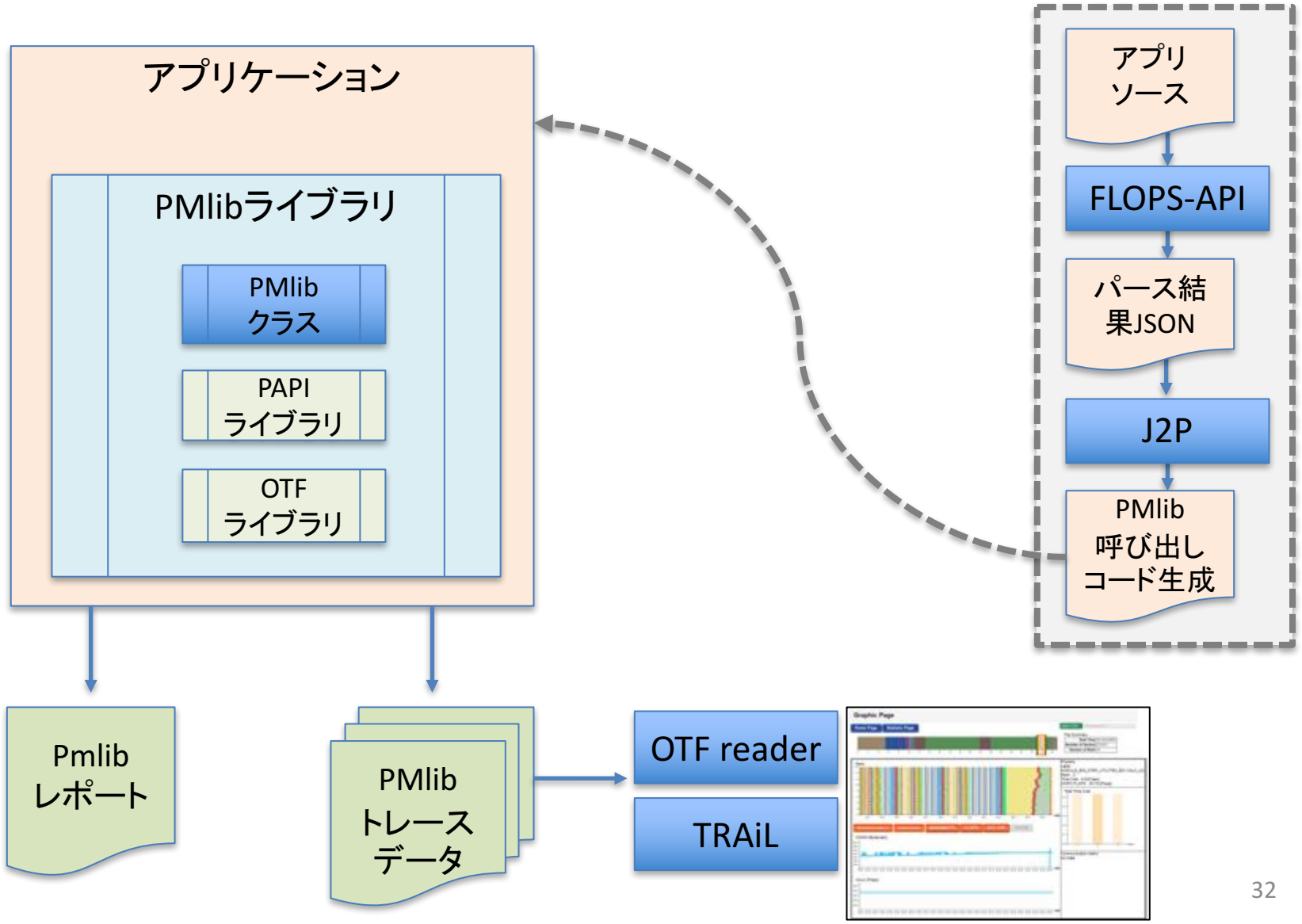
```
! subroutine sub_fcount(a,b,c,n)
! J2P produced the PMlib API argur
! for the following source block
!   "start_line": 1,
!   "end_line": 14,
!
! FLOPS counts in the body
   f_pm_start("location-1")
   f_pm_stop(" location-1", 6+(n-1)*(1+1), 1)
!
! ARRAY access counts in the body
   f_pm_start(" location-1")
   f_pm_stop(" location-1", 7+1+(n-1)*(2+1), 1)
```

PMlib API

```
 1 subroutine sub_fc
 2 integer :: n
 3 real*8 a(n), b(n), c(n)
 4 call f_pm_start("location-1")
 5 a123=a(1)+a(2)+a(3)
 6 b123=b(1)+b(2)+b(3)
 7
 8 do i=1,n-1
 9 c(i)=0.1*(a(i)+b(i))
10 end do
11
12 c(n)=c(n-1)+a123+b123
13 call f_pm_stop("location-1", 6+(n-1)*(1+1), 1)
14 return
15 end
```

PMlibを利用するソース

ツール間の関係


アプリケーション
PMlibライブラリ
PMlib
クラス
PAPI
ライブラリ
OTF
ライブラリ
アプリ
ソース
FLOPS-API
パース結
果JSON
J2P
PMlib
呼び出し
コード生成
Pmlib
レポート
PMlib
トレース
データ
OTF reader
TRAiL

PMlib関数一覧

関数名（C++）	関数名（Fortran）	機能	呼び出し位置と回数	引数
initialize()	f_pm_initialize()	PMlib全体の初期化	冒頭・一回	(1)測定区間数
setProperties()	f_pm_setproperties()	測定区間のラベル化	任意・各区間一回	(1)ラベル、(2)測定対象タイプ、(3)排他指定
start()	f_pm_start()	測定の開始	任意(startとstopでペア)	(1)ラベル
stop()	f_pm_stop()	測定の停止	任意(startとstopでペア)	(1)ラベル、(2)計算量、(3)計算のタスク数
printProgress()	f_pm_printprogress()	測定中の経過情報を表示	任意・制約なし	(1)出力ファイルポインタ、(2)任意のコメント、(3)区間の表示順序指定
print()	f_pm_print()	測定区間毎の基本統計結果表示	測定終了時・一回	(1)出力ファイルポインタ、(2)ホスト名、(3)任意のコメント、(4)区間の表示順序指定
printDetail()	f_pm_printdetail()	MPIランク毎の詳細性能情報の表示	測定終了時・一回	(1)出力ファイルポインタ、(2)記号説明の表示、(3)区間の表示順序指定
printGroup ()	f_pm_printgroup ()	指定プロセスグループに属するMPIランク毎の詳細性能情報の表示	測定終了後・グループ数回	(1)出力ファイルポインタ、(2)group handle、(3)communicator、(4)rank番号配列、(5)グループ番号、(6) 記号説明の表示、(7)区間の表示順序指定
printComm ()	f_pm_printcomm ()	MPI_Comm_split()で作成したグループ毎にprintGroup ()をよびだす	測定終了後・一回	(1)出力ファイルポインタ、(2)communicator handle、(3)カラー変数、(4)key変数、(5) 記号説明の表示、(6)区間の表示順序指定
postTrace ()	f_pm_posttrace ()	ポスト処理用traceファイルの出力	測定終了後・一回	引数なし。環境変数あり。

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

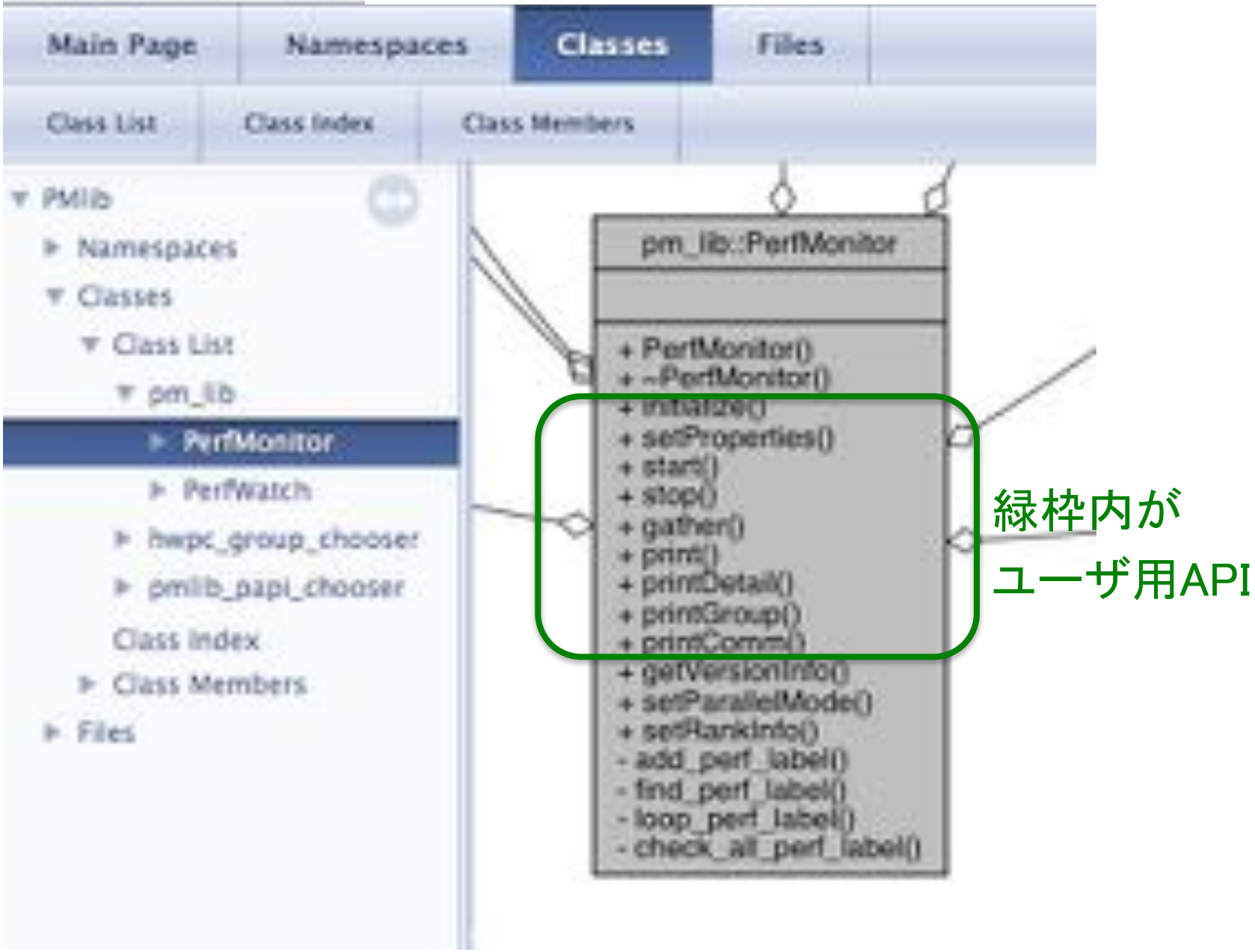
PMlib関数の仕様

- 以降のスライドはPMlibパッケージのdocディレクトリでコマンドを実行して生成することができ、各自のPC上でWebブラウザ表示すると見やすい。

```
$ tar –zxf PMlib-master.tar.gz
$ cd PMlib-master/doc
$ doxygen
$ open html/index.html
```

```
$ tar –zxf PMlib-5.0-API-html.tar.gz
$ open html/index.html
```

PMlib 5.0 各関数の仕様(C++ API)



緑枠内が
ユーザ用API

各関数の仕様　(fortran API)

PMlib 5.0

Main Page | Namespaces | Classes | Files

File List | File Members

- PMlib
  - Namespaces
  - Classes
  - Files
    - File List
      - api_fortran
        - PMlib_Fortran_api.F90
          - f_fortran_api_gene
          - f_pm_gather
          - f_pm_initialize
          - f_pm_print
          - f_pm_printdetail
          - f_pm_printgroup
          - f_pm_setproperties
          - f_pm_start
          - f_pm_stop
      - include
    - File Members

program f_fortran_api_general ( )

PMlib Fortran インタフェイス一般の注意

Note

PMlib Fortran インタフェイスでは引数を省略する事はできない。 またFortran とC++のデータタイプの違いなどにより、 引数仕様が異なるものがある

Fortranコンパイラは character文字列fc に対してその長さ（文字数） を示す引数を内部で自動的に追加する。例えばfcがcharacter*(fc_size) fc と定義されている場合、 call f_pm_api (fc)というインタフェイスに対して実際に生成される コードは右のCコードに相当する。 void f_pm_api_ (char* fc, int fc_size)

呼び出すFortranプログラムからこの追加引数fc_sizeを意識する必要はない。

subroutine f_pm_gather ( )

PMlib Fortran 用 全プロセスの全測定結果情報をマスタープロセス(0)に集約

subroutine f_pm_initialize (nWatch)

PMlib Fortran インタフェイスの初期化

Parameters

[in] integer nWatch 最初に確保する測定区間数。

Note

測定区間数が不明、あるいは動的に増加する場合は nWatch の値を 1 と指定してよい。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。

各関数の仕様　initialize()

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

PMlibの内部初期化

測定区間数分の内部領域を確保しする。並列動作モード、サポートオプション の認識を行い、実行時のオプションにより HWPC、OTF出力用の初期化も行う。

Parameters

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

Note

測定区間数分の内部領域を最初にinit_nWatch区間分を確保する。 測定区間数が不足したらその都度動的に init_nWatch追加する。

各関数の仕様　setProperties()

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

測定区間にプロパティを設定.

Parameters

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

Note

labelラベル文字列は測定区間を識別するために用いる。 各ラベル毎に対応した区間番号を内部で自動生成する 最初に確保した区間数init_nWatchが不足したら動的にinit_nWatch追加する 第１引数は必須。第２引数は明示的な自己申告モードの場合に必須。 第３引数は省略可

各関数の仕様　start()/stop()

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

測定区間スタート

Parameters

〔in〕 label ラベル文字列。測定区間を識別するために用いる。

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

測定区間ストップ

Parameters

〔in〕 label　ラベル文字列。測定区間を識別するために用いる。
〔in〕 flopPerTask　測定区間の計算量(演算量Flopまたは通信量Byte):省略値0
〔in〕 iterationCount 計算量の乗数（反復回数）:省略値1

Note

計算量のボリュームは次のように算出される。
(A) ユーザ申告モードの場合は 1 区間 1 回あたりでflopPerTask*iterationCount
(B) HWPCによる自動算出モードの場合は引数とは関係なくHWPC内部値を利用
出力レポートに表示される計算量は測定のモード・引数の組み合わせで 以下の規則により決定される。

```
/**
(A) ユーザ申告モード
  - HWPC APIが利用できないシステムや環境変数HWPC_CHOOSERが指定
    されていないジョブでは自動的にユーザ申告モードで実行される。
  - ユーザ申告モードでは(1):setProperties() と(2):stop()への引数により
    出力内容が決定される。
  - (1) ::setProperties(区間名, type, exclusive)の第2引数typeが
    計算量のタイプを指定する。演算(CALC)タイプか通信(COMM)タイプか。
  - (2) ::stop (区間名。fPT, iC)の第2引数fPTは測定計算量。
    演算（浮動小数点演算）あるいは通信（MPI通信やメモリロードストア
    などデータ移動)の量を数値や式で与える。

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

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

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

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

各関数の仕様　gather()

void pm_lib::PerfMonitor::gather (void)

全プロセスの測定結果をマスタープロセス(0)に集約.

Note

以下の処理を行う。 各測定区間の全プロセスの測定結果情報をノード０に集約。 測定結果の平均値・標準偏差などの基礎的な統計計算。 経過時間でソートした測定区間のリストm_order[m_nWatch] を作成する。 各測定区間のHWPCイベントの統計値を取得する。 OTFポスト処理ファイルの終了処理。

各関数の仕様　print()

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

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

Parameters

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

Note

基本統計レポートは排他測定区間, 非排他測定区間をともに出力する。 MPIの場合、rank0プロセスの測定回数が1以上の区間のみを表示する。

各関数の仕様　printDetail()

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

MPIランク別詳細レポート、HWPC詳細レポートを出力。

Parameters

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

Note

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

詳細レポートは排他測定区間のみを出力する

各関数の仕様　printGroup()

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

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

Parameters

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

Note

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

各関数の仕様　printComm()

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

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

Parameters

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

PMlib利用プログラム例(再掲)

- 元のソース

```
int main (int argc, char *argv[])
{
     subkernel(); //演算を行う関数

     return 0;
}
```

PMlib組み込み後のソース

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main (int argc, char *argv[])
{
     PM.initialize();
     PM.setProperties(" Kokodayo", 1);
     PM.start(" Kokodayo");
     subkerel();
     PM.stop (" Kokodayo", 0.0, 1);
     PM.gather();
     PM.print(stdout, " Mr.Bean");
     PM.printDetail(stdout);
     return 0;
}
```

ヘッダー部追加

初期設定

測定区間

レポートを出力

前半の資料説明終了