

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

2016年6月22日

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>

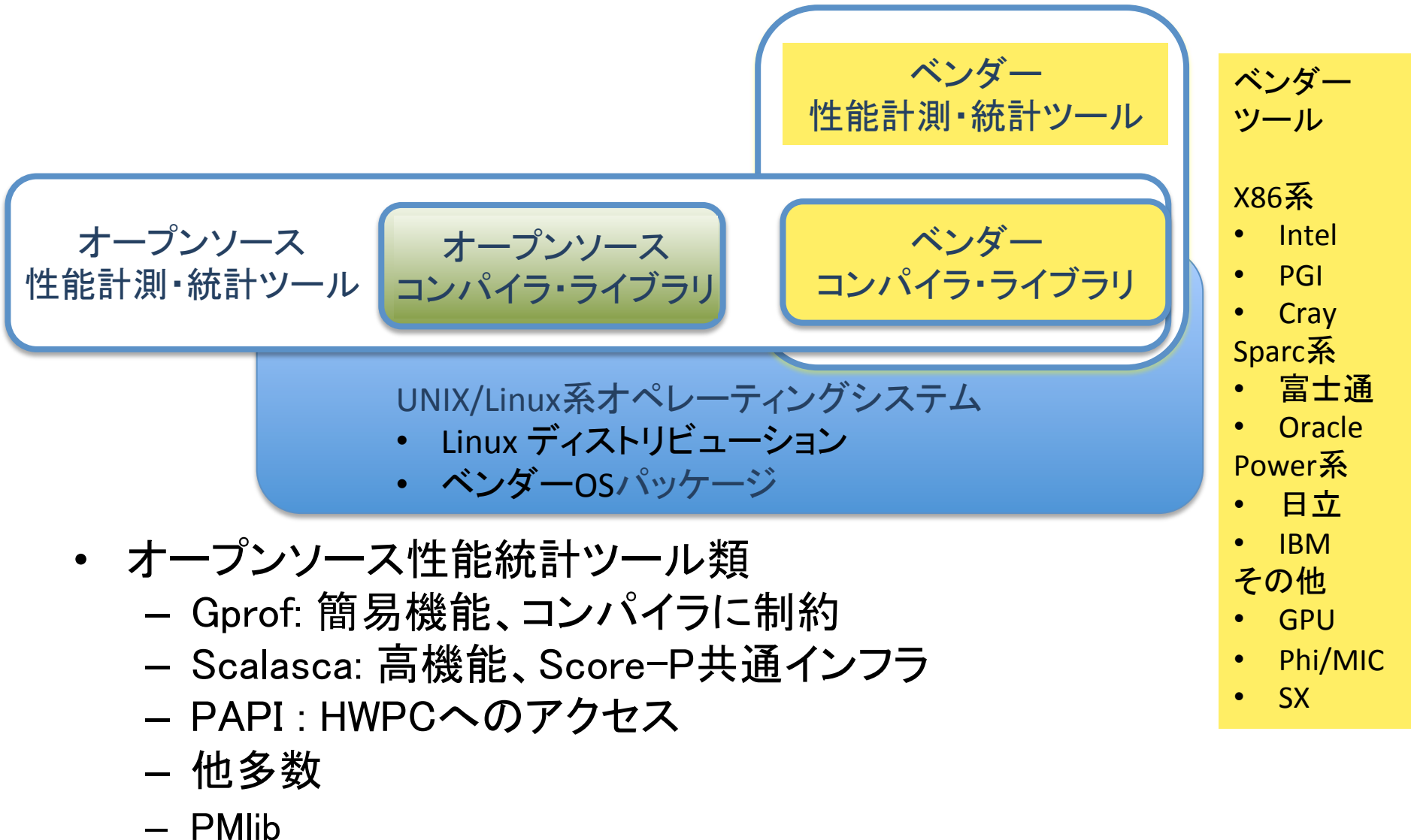
講習会の内容

- 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とは

- 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
- オープンソースソフトウェア(理研 AICSが開発・提供)
- ユーザーライブラリとしてもシステムライブラリとしても利用可
- アプリケーションのソースプログラム中に測定区間を指定。実行終了時に区間の統計情報を出力
- 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を想定
- 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
- C++とFortranに対応したAPI

性能統計ツールの位置づけ

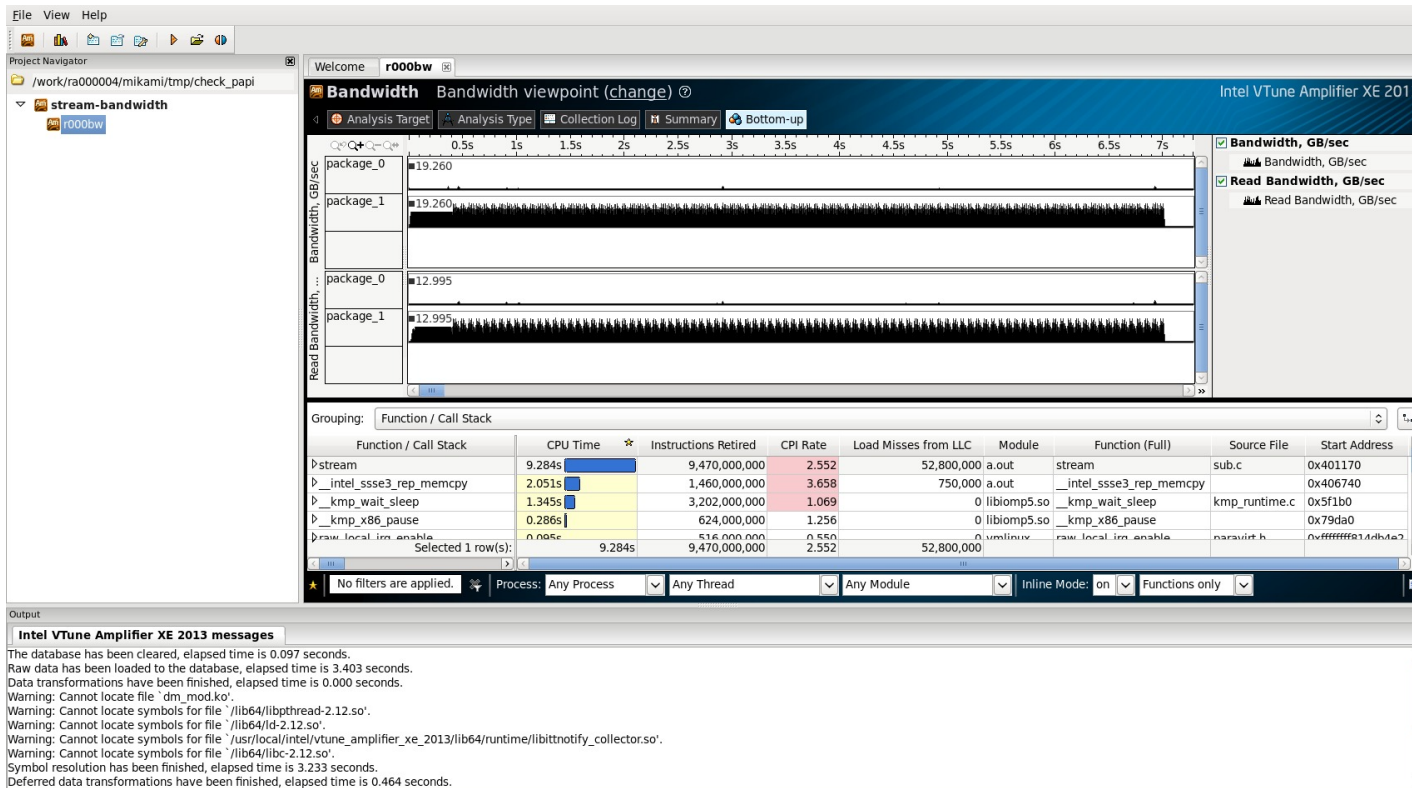


各ツールの位置づけ

- ベンダー性能計測・統計ツール
 - ○豊富な機能、高度なインタフェイス、システムに統合化された安心感、詳しいドキュメント、ベンダーによるサポート
 - △習熟に相当期間が必要、システム機種毎にツールが決まってしまう、それなりの価格
- オープンソース性能計測・統計ツール
 - ○各ツール毎に高機能、無料
 - △ユーザーインタフェイスが個性的、インストールの手間・利用方法の習熟がそれなりに大変→周囲にツールをよく知っている人がいないとハードルは高い
- PMlib
 - オープンソース
 - テキストレポートを基本とした簡易なツール

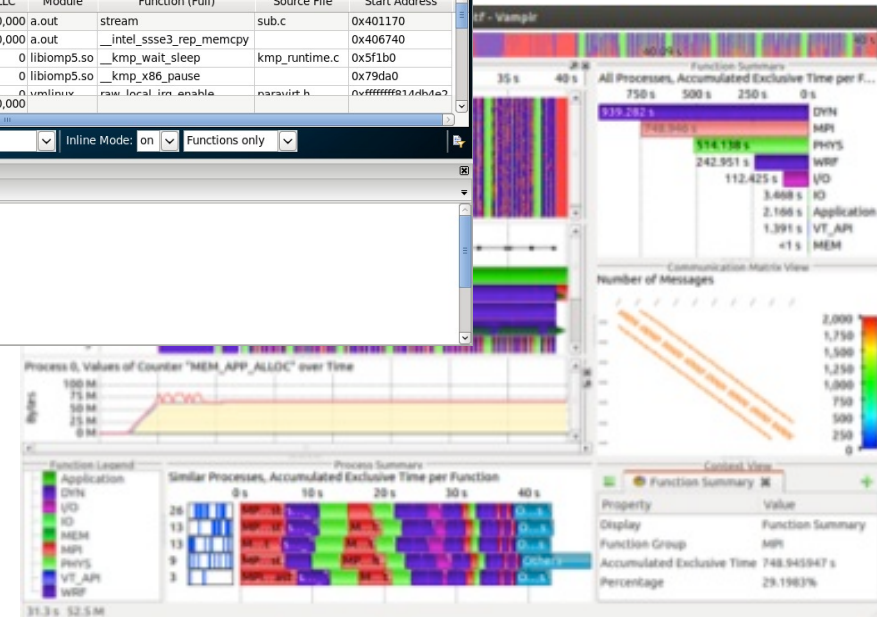
高機能GUIベースツールのハードル

- 見た目の豪華さ ∞ 利用に必要な習熟期間の長さ



↑マニュアル 60MB online HTML

マニュアル100ページ➤



PMlibの特徴

- PMlibの特徴
 - テキストレポートを基本としたコンパクトなツール
 - 機能・出力情報を絞って容易に利用
 - インストールがとても簡単
 - この後の実習
 - アプリケーション中に測定区間を指定してトレーシング測定
 - 測定する情報(計算性能統計情報)とその採取方法を選択可能
 - 計算量を明示的に申告(アルゴリズム性能)
 - 計算量をHWPCを用いて自動的に算出(システム性能)
 - アプリケーションが性能統計レポートを直接出力
 - 指定区間毎の経過時間と計算量
 - 時刻歴情報のポスト処理可視化ツールを別パッケージ化

PMlibが対応する並列プログラムモデル

- シリアルプログラム
- OpenMP (SMPスレッド) 並列プログラム
 - 測定区間内にOpenMPループを含む場合に相当
 - ただしスレッド自身からの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
- APIはC++とFortranに対応

PMlibの動作確認がとれているシステム

- 京/FX10/FX100:
 - 計算ノード用(ログインノードでのクロスコンパイル可能)
 - 富士通コンパイラ+富士通MPI
- Intel Xeon クラスタ: RedHat 6.1+、Suse 11.3+、Linux kernel 2.6.32+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: OSX 10.8以降
 - Apple Clang/LLVMコンパイラ+OpenMPI
 - Intelコンパイラ+OpenMPI
 - GNUコンパイラ
- 必要なソフトウェア環境
 - C++, C, Fortranコンパイラ
 - (必要に応じて)MPIライブラリ、PAPIライブラリ、OTFライブラリ

PMlibの利用方法

- PMlibライブラリのインストール
- アプリケーション中に測定区間を定義する
 - ソースプログラム中の注目箇所にPMlib APIを追加
- アプリケーションを実行する
 - 実行時に性能統計情報がレポートされる(標準出力または指定ファイル)
 - オプションによりポスト処理用トレーシングOTF(詳細時刻歴)データファイルが生成される
- 出力情報の評価
 - テキストレポートの数値を評価する
 - オプションにより生成されたデータファイルを、ポスト処理可視化ツールを用いて可視化を行い時刻歴評価する

PMlib利用プログラム例

- 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(" Koko!", 1);
    PM.start(" Koko!");
    subkernel();
    PM.stop (" Koko!", 100.0, 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

PMlib計算性能モニター機能

- 指定した測定区間毎に性能統計情報を蓄積・記録
- 各測定区間は少数のプロパティを持つ
 - ラベル: 測定区間につける名称。任意の文字列
 - 測定する計算量: 「演算量」、「通信量」、「実行時に動的決定」
 - 排他性: 「排他的測定区間」または「非排他的測定区間」
- 測定区間毎の性能統計値の種類
 - 経過時間の累積値・毎回値
 - 計算量: 演算量(主として浮動小数点演算量)・通信量(データ移動量)・その他HW固有の評価可能な統計量
- 測定区間の計算量の決定・算出方法
 - ユーザが明示的に申告する場合
 - PMlibとシステムにHWPC値を自動採取させる場合

計算量：明示的な申告モード

- 計算量をユーザが明示的に申告する場合
 - 測定区間の計算量を値あるいは計算式でPMlib APIの引数として与える
 - ソースプログラムで記述された計算量に基づいた(アルゴリズムベースでの)性能を評価したい場合に適している
 - 演算の種類(四則演算・基本関数・アルゴリズムのブロック)などの粒度に応じて自分で決定できる
 - ではその値・計算式はどのように算出するかという
 - ソースからを自分で数え上げる・机上で設定する
 - ソースのパースーツールを用いて算出する

ソースのパーサーツール例1

- 例1: KSCOPE
 - Kscope (<http://www.aics.riken.jp/ungi/soft/kscope/>)
 - 対話的な変数スコープ分析
 - 分析情報をCSV出力

The screenshot displays the Kscope application window. On the left, a 'Structure tree' shows the program's hierarchy, with the 'do k = 1, n, 1' loop selected. The main window shows the Fortran source code for 'fmxm.f90', highlighting a loop from line 58 to 64. The bottom panel shows the '演算数カウント' (Operation Count) tab, displaying a table of FLOP counts for the selected loop block.

Block	FLOP	add(F)	sub(F)	mul(F)	div(F)	intrinsic(F)
DO 59	2	1	0	1	0	0

At the bottom of the window, status bars indicate: 'ツリー情報のエクスポート: 完了' (Tree information export: completed), '58 : 1' (current line), and '構造解析: 終了' (Structure analysis: completed).

ソースのパーサーツール例2

- 例2: FLOPS-API (AICS利用高度化チームの試験的Webサービス)

- ソースプログラムを含むリポジトリを指定してPOST
- 分析情報をJSON出力

The screenshot displays a web browser window at the URL `togetsu.aics10.riken.jp:18080/flops/tasks/`. The page is titled "Task Detail" and is part of a "Django REST framework". It features three buttons at the top right: "DELETE" (red), "OPTIONS" (blue), and "GET" (blue with a dropdown arrow). Below these buttons, the HTTP method "GET" and the full URL `/flops/tasks/305591e5-24ac-42c8-9fce-4ac2eb1fa4fc/` are shown. The main content area displays the HTTP response status "HTTP 200 OK" and headers: "Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS", "Content-Type: application/json", and "Vary: Accept". The response body is a JSON object containing task details. At the bottom right, there are tabs for "Raw data" and "HTML form". The "HTML form" tab is active, showing a form with fields for "Repository url", "Commit", "Status", "Date requested", "Date started", and "Date finished", each with its corresponding value from the JSON response. A "PUT" button is located at the bottom right of the form.

```
{
  "id": "305591e5-24ac-42c8-9fce-4ac2eb1fa4fc",
  "repository_url": "https://github.com/mikami3heart/prog_fort.git",
  "commit": "83cf457afbe68266bc11f8961abfc785138ef748",
  "status": "finished",
  "request_date": "2016-06-14T09:11:46.018551Z",
  "start_date": "2016-06-14T09:11:47.672115Z",
  "finish_date": "2016-06-14T09:12:30.351719Z"
}
```

Repository url	https://github.com/mikami3heart/prog_fort.git
Commit	83cf457afbe68266bc11f8961abfc785138ef748
Status	finished
Date requested	2016-06-14T09:11:46.018551
Date started	2016-06-14T09:11:47.672115
Date finished	2016-06-14T09:12:30.351719

ソースのパースーツール例2

FLOPS-API (Webサービス)

```
subroutine sub_fcount(a,b,c,n)
integer :: n
real*8 a(n), b(n), c(n)
do i=1,n
c(i)=0.1/(a(i)+b(i))
end do
do i=1,n
c(i)=c(i)+a(i)*b(i)
end do
return
end
```

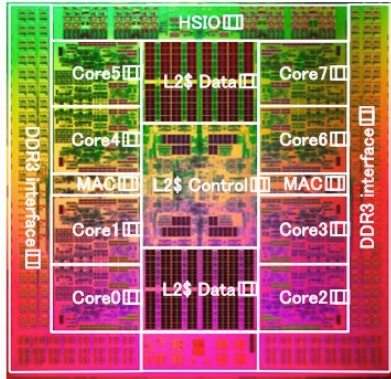
```
{
  "loc": "check_counts.f90",
  "pu": "sub_fcount",
  "niter": "n",
  "cat": "do-construct",
  "metrics": {
    "nfadd": 1,
    "nfddiv": 1,
    "naref": 3
  },
  "end_line": 7,
  "start_line": 5,
  "type": "loop"
}, {"loc": "check_counts.f90",
  "pu": "sub_fcount",
  "niter": "n - 1",
  "cat": "do-construct",
  "metrics": {
    "nfadd": 1,
    "nfmul": 1,
    "naref": 4
  },
  "end_line": 10,
  "start_line": 8,
  "type": "loop"
}
```

計算量：HWPCによる自動算出モード

- システムがハードウェア性能カウンタ(HWPC)を内部に持ち、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経由で情報を採取する
 - コンパイラによる命令最適化の効果やプロセッサ・階層的メモリアウトなどの特性効果を含む
 - システム性能の評価に適する
- HWPCイベントの種類毎にカウンターグループを定義
- プログラム実行時に環境変数HWPC_CHOOSERで選択する
 - FLOPS
 - BANDWIDTH
 - VECTOR
 - CACHE
 - CYCLE
- 全イベントリスト別表(参考資料)

ハードウェアカウンタ(京計算ノード)

SPARC64TMVIIIfx



■ 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

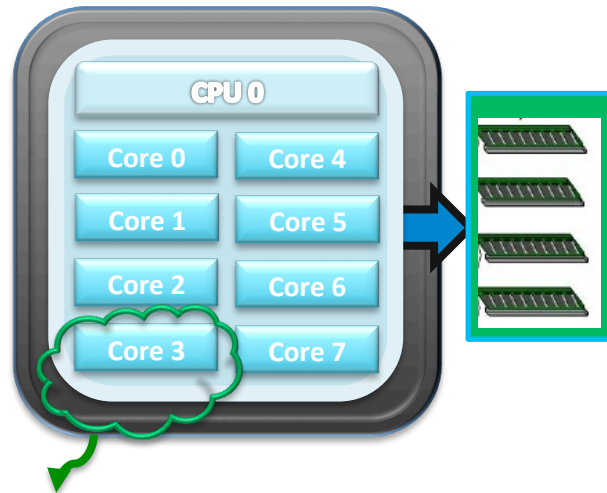
ハードウェアカウンタ(京計算ノード) preset とnative

```
$ papi_avail -e PAPI_FP_OPS
```

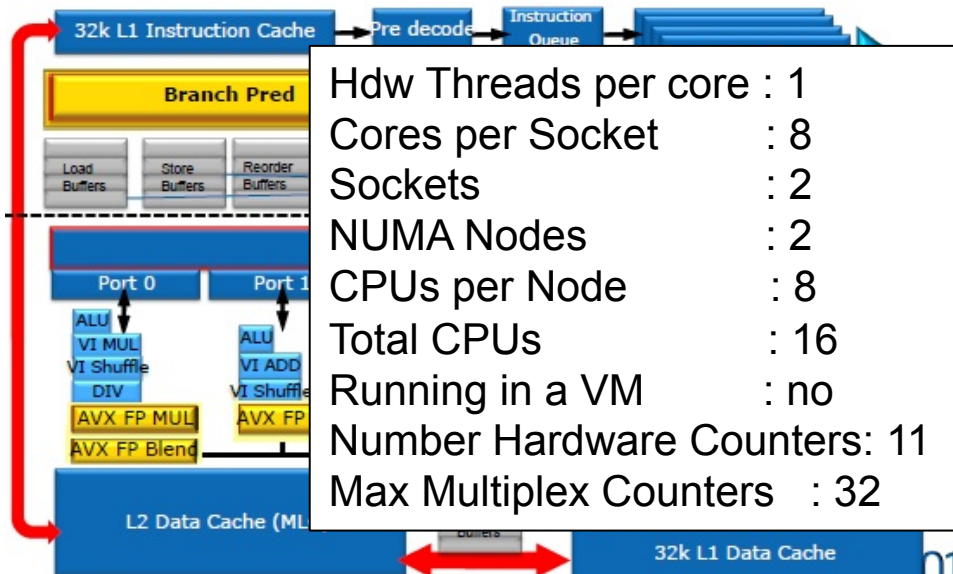
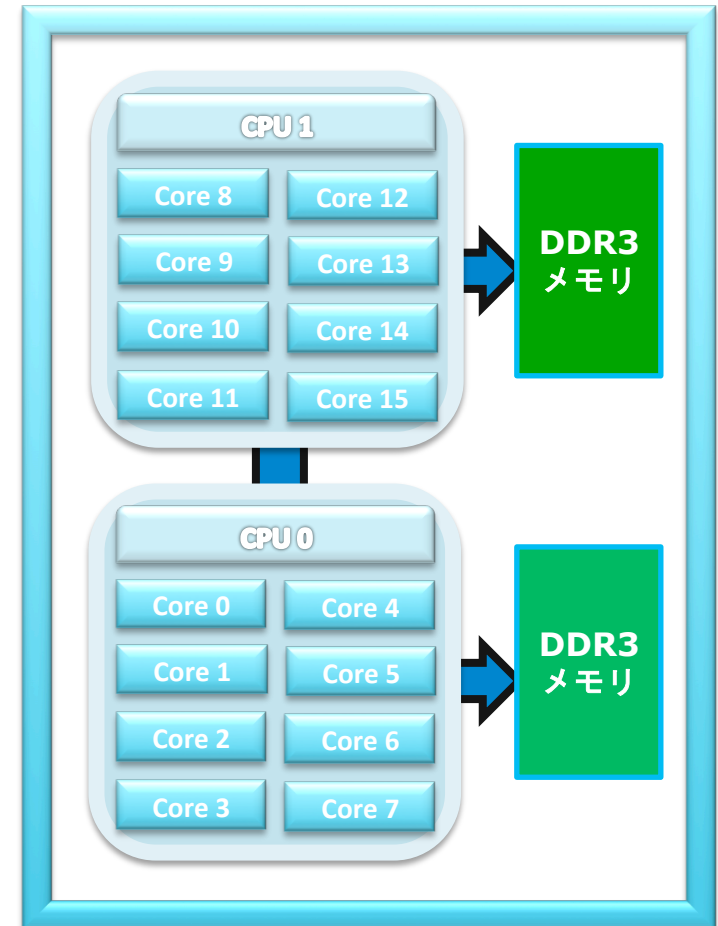
Event name:	PAPI_FP_OPS
Number of Native Events:	4
Short Description:	FP operations
Long Description:	Floating point operations
Derived Type:	DERIVED_POSTFIX
Native Code[0]: 0x40000010	FLOATING_INSTRUCTIONS
Native Event Description:	Counts the number of committed floating-point operation instructions.
Native Code[1]: 0x40000011	FMA_INSTRUCTIONS
Native Event Description:	Counts the number of committed floating-point Multiply-and-Add operation instructions.
Native Code[2]: 0x40000008	SIMD_FLOATING_INSTRUCTIONS
Native Event Description:	Counts the number of committed floating-point SIMD instructions of one operation in SIMD.
Native Code[3]: 0x40000009	SIMD_FMA_INSTRUCTIONS
Native Event Description:	Counts the number of committed floating-point SIMD instructions of two operation in SIMD.

ハードウェアカウンタ (Intel Xeon E5)

CPU : Xeon E5-2670



uncore: L3, QPI, PCIe, ...



Of 50 available events, 17 are derived.

出力レポート

- 1、基本レポート
 - 測定区間毎のプロセス平均プロファイル
 - ジョブあたりの総合性能
- 2、詳細プロファイル:MPIプロセス毎
 - 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。
- 3、詳細プロファイル:HWPCイベント統計
 - HWPCイベントグループを環境変数で指定
 - 各MPIプロセス毎のHWPCイベント統計量を出力
 - 同上
- 区間を登録順、または経過時間順にソート出力

出力するプロフィール情報の選択

- 出力レポートに表示される情報はモード・引数の組み合わせで決める
- 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setProperty() とstop()への引数により出力内容が決定、HWPC詳細レポートは出力されない。
 - (1) ::setProperty(区間名, type, exclusive)の第2引数typeは測定量のタイプを指定する。計算(CALC)タイプか通信(COMM)タイプかの選択を行なう、ユーザ申告モードで有効な引数。
 - (2) ::stop (区間名, fPT, iC)の第2引数fPTは測定量。計算(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。

setProperty() type引数	stop() fPT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

出力するプロファイル情報の選択

- 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 HWPC_CHOOSER	setProperty()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
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

基本レポート例 ユーザ申告モード

PMLib Basic Report -----

Timing Statistics Report from PMLib version 5.0.3

Linked PMLib supports: MPI, OpenMP, HWPC, OTF

Host name : vsp01

Date : 2016/06/19 : 15:28:19

Mrs. Kobe

Parallel Mode: Hybrid (4 processes x 4 threads)

The environment variable HWPC_CHOOSER is not provided. No HWPC report.

Total execution time = 9.795189e-01 [sec]

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

Exclusive sections statistics per process and total job.

Inclusive sections are marked with (*)

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

基本レポート例 HWPCによる自動算出モード

PMLib Basic Report -----

Timing Statistics Report from PMLib version 5.0.3

Linked PMLib supports: MPI, OpenMP, HWPC, OTF

Host name : vsp01

Date : 2016/06/19 : 15:26:50

Mrs. Kobe

Parallel Mode: Hybrid (4 processes x 4 threads)

The environment variable HWPC_CHOOSER=FLOPS is provided.

Total execution time = 9.848690e-01 [sec]

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

Exclusive sections statistics per process and total job.

Inclusive sections are marked with (*)

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

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

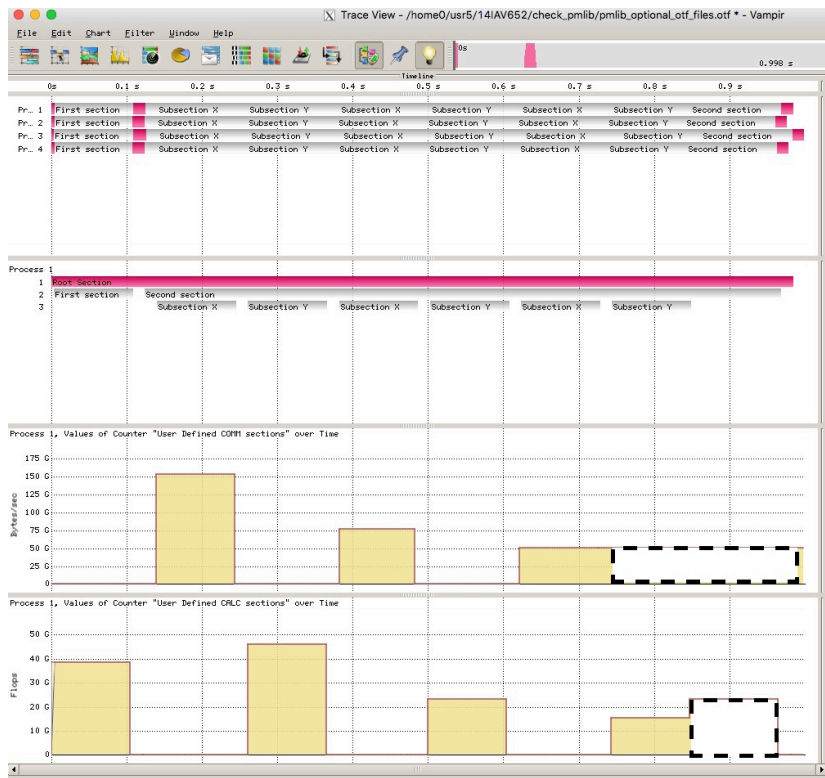
出力のオプション: ポスト処理用ファイル

- 計算性能トレース情報を時刻歴に可視化処理を行うための機能
- Open Trace Format API Specification 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://wwpub.zih.tu-dresden.de/~jurenz/otf/api/current/>
 - <http://www.paratools.com/otf/specification.pdf>
- 以下の環境変数を指定してPMlib からOTFファイルを出力
 - OTF_TRACING=[off | on | full]
 - OTF_FILENAME=出力ファイル名ヘッダ

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ェアで処理
 - Vampir

- TRaIL (可視化チームで開発中のオープンソースSW)



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・区間毎一回	(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
gather()	f_pm_gather()	測定結果情報をマスタープロセスに集約	測定終了後・一回	なし
print()	f_pm_print()	測定区間毎の基本統計結果表示	測定終了後・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	測定終了後・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 ()		MPI_Comm_split()で作成したcommunicatorからプロセスグループ自動作成し、printGroup ()をよびだすヘルパー関数	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

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。

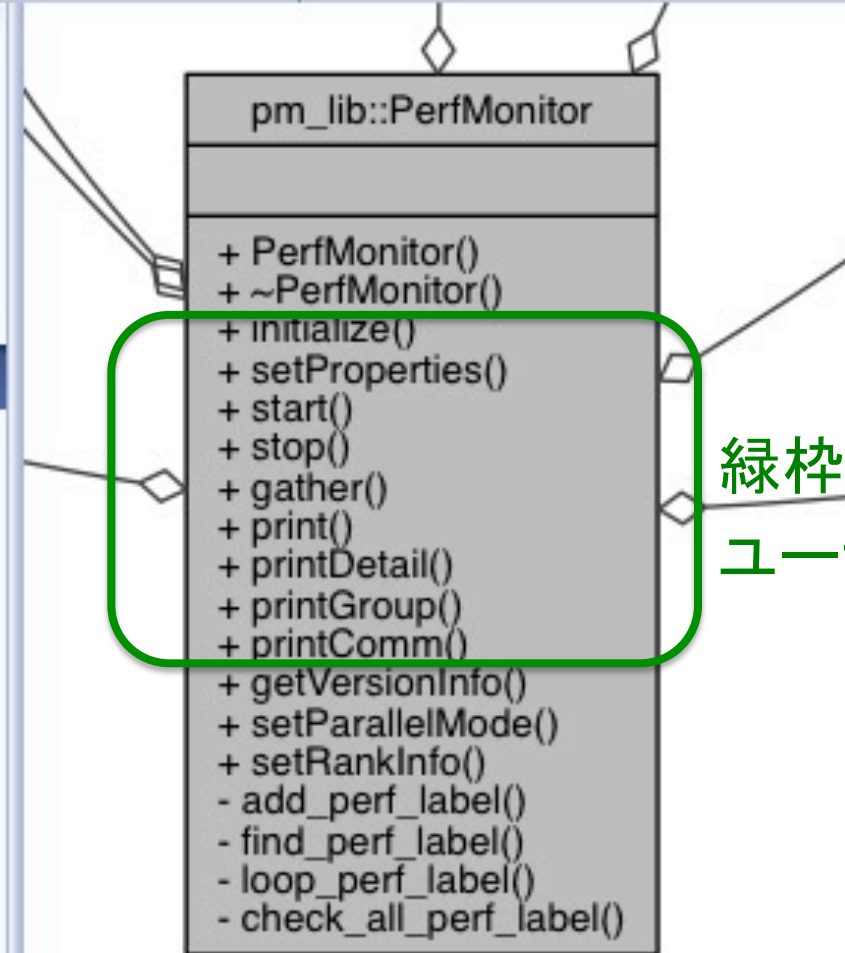
```
$ tar -zxf PMlib-master.tar.gz  
$ cd PMlib-master/doc  
$ doxygen  
$ open html/index.html
```

- PC上でdoxygenコマンドが利用できない場合は、「本日使用する資料のページ」から PMlib-5.0-API-html.tar.gz ファイルをダウンロードし、復元したindex.htmlファイルを各自のPC上のWebブラウザで表示すると良い

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

Main Page	Namespaces	Classes	Files
Class List	Class Index	Class Members	

▼ PMlib	↔
▶ Namespaces	
▼ Classes	
▼ Class List	
▼ pm_lib	
▶ PerfMonitor	
▶ PerfWatch	
▶ hwpc_group_chooser	
▶ pmlib_papi_chooser	
Class Index	
▶ Class Members	
▶ Files	



緑枠内が
ユーザ用API

- ▼ 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 インタフェース一般の注意

Note

PMlib Fortran インタフェー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)というインタフェー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 !! 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 インタフェースの初期化

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 $\text{flopPerTask} \times \text{iterationCount}$

(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)::setProperty() と(2)::stop()への引数により出力内容が決定される。
- (1) ::setProperty(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。

setProperty()	stop()	
type引数	fpt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	setProperty()の type引数	stop()の fpt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、fpt引数によるFlops	なし
NONE (無指定)	COMM	指定値	時間、fpt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づらい場合がある。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");
    subkernel();
    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