PMlib 講習会

理化学研究所 計算科学研究機構
可視化技術研究グループ
2014年5月27日

講習会の概要

- 講義編
  - PMlibとは
  - PMlibの位置づけ
  - PMlibの入手方法
  - PMlibの利用方法
- 関連情報編（別途資料A）
  - コンピュータシステムの性能
  - アプリケーションの性能
  - 性能モニターの範囲
- 実習編
  - 実習システムへのログイン
  - PMlibのインストール
  - 例題プログラムへの組み込み
  - 実行結果の解釈
  - 例題プログラムの性能向上の検討

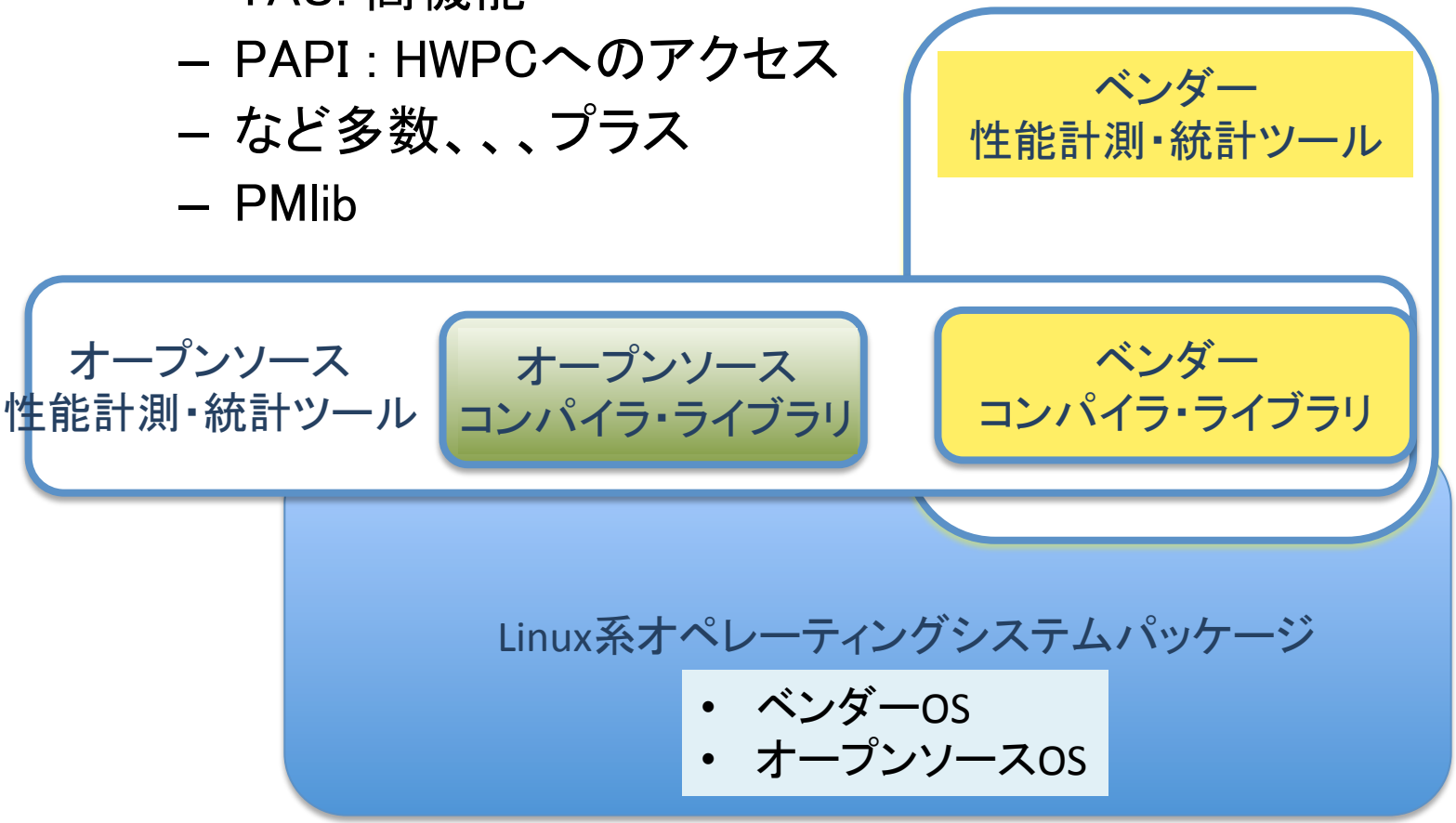
講習編

PMlib

- PMlibとは
  - 計算性能モニター用クラスライブラリ
  - アプリケーションの性能を改善するための補助ツール
  - オープンソースソフトウエア（理研 AICSが開発・提供）
  - インストール・利用が容易な「軽量」ツール
  - 計測結果情報は指定区間毎に出力
    - 出力タイプ1:全プロセスの平均した基本情報
    - 出力タイプ2:ハードウエアイベント毎の情報
    - 出力タイプ3:MPIランク（プロセス）毎の情報
  - 出力タイプ1、3はLinuxシステム全般で利用可能
  - 出力タイプ2は「京」コンピュータ、X86系などで利用可能

PMlibの位置づけ

- オープンソース性能統計ツール一般
  - 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
  - TAU: 高機能
  - PAPI : HWPCへのアクセス
  - など多数、、、プラス
  - PMlib



ベンダーツール

X86系
- Intel
- PGI
- Cray

Sparc系
- 富士通
- Oracle

Power系
- 日立
- IBM

その他の系
- GPU
- Phi
- SX

PMlibの位置づけ

- ベンダー性能計測・統計ツール
  - 〇豊富な機能、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統計ツール
  - 〇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
  - 機能・出力情報を絞ったコンパクトなツール
  - インストール・利用方法とも簡単→手軽に利用可能
  - ソースプログラム中にPMlib APIを記述して利用
  - 実際の稼働ベースでアプリケーションの実行性能が測定可能

PMlibの入手方法

- PMlibパッケージの入手方法
  - 下記から直接Download
  - https://github.com/avr-aics-riken/PMlib
- PMlibに関するドキュメント
  - パッケージに含まれるdoc/ディレクトリ以下にある
    - How_to_use_PMlib.pdf　:クラスライブラリの説明書
    - PMlib_getting_started.pdf :本資料
- PMlibのインストール方法・実例は実習編を参照

PMlib計算性能測定機能

- プログラム実行時に測定指定区間の実行時間と性能に関連する統計情報を出力する機能
- 各測定区間は以下のプロパティを持つ
  - キー番号：整数値
  - ラベル：統計情報出力時のラベル文字列
  - 測定対象タイプ：「計算時間」または「通信時間」
  - 排他測定フラグ：「排他測定」または「非排他測定」
- 性能統計情報
  - 計算量をユーザが明示的に申告
    - 測定区間の「計算量」を引数として自己申告。（stop メソッドのオプション）
    - 申告量の解釈は測定対象タイプがによる
      - 計算時間の場合は浮動小数点演算量：計算速度（FLOPS 値）を出力
      - 通信時間の場合は通信量バイト単位：通信速度（Byte/s 単位）を出力
  - 計算量の自動算出
    - ハードウエア性能カウンター(HWPC)のイベントを測定して出力
    - イベントグループを環境変数で動的に選択する

PMlibのプロファイルのタイプ

- 基本プロファイル例
  - 全プロセスの平均情報
  - プログラム終了時に各MPIプロセス（ランク）の情報をマスターランクに集計。統計処理して出力
- 詳細プロファイル(1：MPIプロセス毎)
  - MPIの各プロセス毎の情報を出力
- 詳細プロファイル(2：HWPCイベント統計)
  - 計測するHWPCイベントグループを環境変数で指定
  - プロセスがOpenMPスレッドを発生した場合各プロセスのにスレッド測定値を内部で合計する。マスタープロセス(MPI rank 0)の値を出力
- スレッド中からのPMlibよびだしには未対応

プロファイル：明示的な自己申告

- 計算量をユーザが明示的に申告する場合
  - 実行時の各セクションのタイミングと演算数を積算して記録
    - タイミング測定区間はラベル管理で，コーディング時に指定
    - 演算数は，各関数毎にマニュアルでカウント
      - FX10のPA情報から推定．例えば．．
      - +, −, x　　：　1 flop
      - ÷　　：　8 flops（単精度），13 flops（倍精度）
      - abs()　：　1 flops
  - ソースプログラムの計算実行式通りの実行性能を測定可能
- FX10の詳細プロファイルとの連携
  - 同じタイミングで、FXの区間指定が可能

プロファイル：計算量の自動算出

- 性能システムのCPUが内部に持つハードウエア性能カウンター(HWPC)のイベントを測定して出力
- HWPCのイベントリスト別表
- PMlib用にイベントの種類毎グループを定義
  - FLOPS
  - VECTOR
  - BANDWIDTH
  - CACHE
  - CYCLE
- プログラム実行時に環境変数で動的に選択する
  - マスタープロセス(MPI rank 0)の測定値を代表値として出力
  - もしOpenMPスレッド並列処理の場合はマスタープロセスが発生するスレッド群の合計値を出力

基本プロファイル例

```
-----------------------------------------------------
     Report of Timing Statistics PMlib version 2.1.2
     Operator  : Kenji_Ono
     Host name : vsp22
     Date      : 2014/05/26 : 03:49:43

     Parallel Mode                     :   Hybrid (4 processes x 8 threads)

     Total execution time            = 4.429648e+00 [sec]
     Total time of measured sections = 3.695136e+00 [sec]

     Statistics per MPI process [Node Average]
     Label                  |     call     |              accumulated time                  |          flop | messages[Bytes]
                            |              |   avr[sec]    avr[%]     sdv[sec]  avr/call[sec] |   avr         sdv         speed
     -----------------------+--------------+-----------------------------------------------+-----------------------------------
     Poisson_SOR2_SMA       :        26200   1.071254e+00   28.99   6.4536e-03   4.088757e-05    2.318e+10   0.000e+00    21.64 Gflops
     File_Output            :           11   8.003855e-01   21.66   1.3644e-02   7.276232e-02    1.311e+07   0.000e+00    16.38 Mflops
     Poisson_Src_Norm       :        13300   7.372693e-01   19.95   2.6937e-03   5.543378e-05    4.112e+10   0.000e+00    55.77 Gflops
     Sync_Pressure          :        26200   5.582871e-01   15.11   7.6391e-03   2.130867e-05    1.717e+09   0.000e+00     2.86 GB/sec
     A_R_Poisson_Src        :        13300   2.086478e-01    5.65   2.1901e-02   1.568781e-05    8.512e+05   0.000e+00     3.89 MB/sec
     Pseudo_Velocity        :          100   7.437694e-02    2.01   4.6558e-04   7.437694e-04    5.230e+09   0.000e+00    70.31 Gflops
     Projection_Velocity    :          262   7.227474e-02    1.96   6.3243e-04   2.758578e-04    1.820e+09   0.000e+00    25.18 Gflops
     Poisson_BC             :        26200   5.180532e-02    1.40   8.1025e-04   1.977302e-06    0.000e+00   0.000e+00     0.00 Mflops
     Poisson_Setup_for_Itr  :        13100   2.677810e-02    0.72   2.0531e-04   2.044130e-06    0.000e+00   0.000e+00     0.00 Mflops
     Sync_Velocity          :          100   1.325995e-02    0.36   3.9118e-04   1.325995e-04    1.573e+08   0.000e+00    11.05 GB/sec
     Sync_Pseudo_Velocity   :          100   1.039678e-02    0.28   7.3619e-04   1.039678e-04    3.932e+07   0.000e+00     3.52 GB/sec
     Variation_Space        :          100   9.503841e-03    0.26   5.3508e-05   9.503841e-05    4.260e+08   0.000e+00    44.82 Gflops
     Force_Calculation      :          100   6.575465e-03    0.18   7.8837e-03   6.575465e-05    0.000e+00   0.000e+00     0.00 Mflops
     Copy_Array             :          200   6.332040e-03    0.17   8.1976e-05   3.166020e-05    0.000e+00   0.000e+00     0.00 Mflops
     Projection_Velocity_BC :          262   6.013393e-03    0.16   8.4818e-05   2.295188e-05    0.000e+00   0.000e+00     0.00 Mflops
     Divergence_of_Pvec     :          100   5.138874e-03    0.14   5.1616e-05   5.138874e-05    2.425e+08   0.000e+00    47.19 Gflops
     A_R_Poisson_Norm       :          262   5.013704e-03    0.14   6.8061e-04   1.913628e-05    1.677e+04   0.000e+00     3.19 MB/sec
     Allocate_Arrays        :            4   5.009890e-03    0.14   2.1477e-04   1.252472e-03    0.000e+00   0.000e+00     0.00 Mflops
    ...
-----------------------+--------------+-----------------------------------------------+-----------------------------------
     Total                  |                3.695136e+00                                    7.225e+10                19.55 Gflops
     Performance            |                                                                                         78.21 Gflops
```

詳細プロファイル(1)

```
  Elapsed time variation over MPI ranks

*Initialization_Section
MPI_rank        call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :            1  1.623359e-01    4.39  6.041527e-04  1.623359e-01   0.000e+00     0.00 Mflops
#1    :            1  1.629400e-01    4.41  0.000000e+00  1.629400e-01   0.000e+00     0.00 Mflops
#2    :            1  1.543641e-01    4.18  8.575916e-03  1.543641e-01   0.000e+00     0.00 Mflops
#3    :            1  1.500421e-01    4.06  1.289797e-02  1.500421e-01   0.000e+00     0.00 Mflops

Allocate_Arrays
MPI_rank        call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :            4  5.146027e-03    0.14  9.012222e-05  1.286507e-03   0.000e+00     0.00 Mflops
#1    :            4  5.236149e-03    0.14  0.000000e+00  1.309037e-03   0.000e+00     0.00 Mflops
#2    :            4  4.867315e-03    0.13  3.688335e-04  1.216829e-03   0.000e+00     0.00 Mflops
#3    :            4  4.790068e-03    0.13  4.460812e-04  1.197517e-03   0.000e+00     0.00 Mflops

*Voxel_Prep_Section
MPI_rank        call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :            1  6.849098e-02    1.85  0.000000e+00  6.849098e-02   0.000e+00     0.00 Mflops
#1    :            1  6.354094e-02    1.72  4.950047e-03  6.354094e-02   0.000e+00     0.00 Mflops
#2    :            1  6.057596e-02    1.64  7.915020e-03  6.057596e-02   0.000e+00     0.00 Mflops
#3    :            1  2.413917e-02    0.65  4.435182e-02  2.413917e-02   0.000e+00     0.00 Mflops

Restart_Process
MPI_rank        call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :            1  4.053116e-06    0.00  9.536743e-07  4.053116e-06   0.000e+00     0.00 Mflops
#1    :            1  5.006790e-06    0.00  0.000000e+00  5.006790e-06   0.000e+00     0.00 Mflops
#2    :            1  3.099442e-06    0.00  1.907349e-06  3.099442e-06   0.000e+00     0.00 Mflops
#3    :            1  4.053116e-06    0.00  9.536743e-07  4.053116e-06   0.000e+00     0.00 Mflops
```

詳細プロファイル(2)

```
PMlib detected the CPU architecture:
   The available Hardware Performance Counter (HWPC) events depend on this CPU architecture.
       GenuineIntel
       Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
   HWPC event values of the master rank, sum of threads. count unit in Giga (x 10e9)
   -                       :       LD_INS      SR_INS    :HIT_LFB     :L1_HIT     :L2_HIT     :L3_HIT     OFFCORE  [Mem GB/s]
   Initialization_Section  :        0.702       0.104       0.000       0.702       0.000       0.000       0.000       0.069
   Allocate_Arrays         :        0.125       0.023       0.000       0.125       0.000       0.000       0.000       2.371
   Voxel_Prep_Section      :        0.126       0.021       0.000       0.126       0.000       0.000       0.000       0.023
   Restart_Process         :        0.000       0.000       0.000       0.000       0.000       0.000       0.000      14.726
   Time_Step_Loop_Section  :        0.005       0.001       0.000       0.005       0.000       0.000       0.000       0.001
   Search_Vmax             :        0.027       0.003       0.001       0.026       0.000       0.000       0.002      64.719
   A_R_Vmax                :        0.024       0.005       0.000       0.024       0.000       0.000       0.000       3.541
   Copy_Array              :        0.074       0.042       0.022       0.051       0.000       0.000       0.004      36.320
   assign_Const_to_Array   :        0.010       0.005       0.000       0.009       0.000       0.000       0.000       2.816
   Flow_Section            :        0.005       0.001       0.000       0.005       0.000       0.000       0.000       0.001
   NS__F_Step_Section      :        0.004       0.001       0.000       0.004       0.000       0.000       0.000       0.035
   NS__F_Step_Sct_1        :        0.005       0.001       0.000       0.005       0.000       0.000       0.000       0.571
   NS__F_Step_Sct_2        :        0.005       0.001       0.000       0.005       0.000       0.000       0.000       0.047
   Pseudo_Velocity         :        0.721       0.363       0.002       0.705       0.000       0.000       0.010       8.086
```

利用するプログラム例

```
#include <PerfMonitor.h>
using namespace pm_lib;
     //   "Serial"; "OpenMP"; "FlatMPI"; "Hybrid" ;
     char parallel_mode[] = "Hybrid" ;
PerfMonitor PM;
     int main (int argc, char *argv[])
     {
     enum timing_key {tm_sec1, tm_sec2, tm_sec3, tm_END_,};
     MPI_Init(&argc, &argv);
     MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
     MPI_Comm_size(MPI_COMM_WORLD, &npes);
     num_threads  = omp_get_max_threads();

PM.setRankInfo(my_id);
PM.initialize(tm_END_);
PM.setProperties(tm_sec1, "section1", PerfMonitor::CALC);
PM.setParallelMode(parallel_mode, num_threads, npes);
PM.start(tm_sec1);
     compute_some_kerel();
PM.stop (tm_sec1);
PM.gather();
PM.print(stdout, "QQQ", "RRR");
PM.printDetail(stdout);
     MPI_Finalize();
     return 0;
     }
```

利用方法の簡単な説明 1

- 計時区間のキーと総数 tm_END

```
enum timing_key {
  tm_init_sct,
    tm_init_alloc,

    tm_voxel_prep_sct,
      tm_voxel_load,
      tm_polygon_load,
      tm_cutinfo,

    tm_restart,

  tm_loop_sct,

  tm_vmax,
  tm_vmax_comm,

  ...

  tm_END
};
```

プログラム中で測定する区間に対して、
enumで順に整数を割り当てる。
tm_ENDで総数がわかる。

利用方法の簡単な説明 2

- ラベル（測定区間名）の配列宣言とラベル指定

```
Void set_timing_label(void)
{
 set_label(tm_init_sct,          "Initialization_Section",  PerfMonitor::CALC, false);
 set_label(tm_init_alloc,        "Allocate_Arrays",        PerfMonitor::CALC)
 set_label(tm_vmax,              "Search_Vmax",            PerfMonitor::CALC);
 set_label(tm_vmax_comm,         "A_R_Vmax",               PerfMonitor::COMM);
  ...
```

false 測定区間が重複していることを指示

キー

ラベル
文字数は任意

CALC 計算部であることを指定
COMM 通信部 〃

```
//@param[in] key キー番号
//@param[in] label ラベル
//@param[in] type  測定対象タイプ(COMM or CALC)
//@param[in] exclusive 排他測定フラグ(ディフォルトtrue)
Void set_label(unsigned key, char* label, PerfMonitor::Type type, bool exclusive)
{
  // Performance Monitorへの登録
  string tmp(label);
  PM.setProperties(key, tmp, type, exclusive);

  // FX用プロファイラの文字列登録
  strcpy(tm_label_ptr[key], label);
}
```

利用方法の簡単な説明 3

- 測定区間の記述（方法1：演算数やデータ量をユーザーが登録する）
  - TIMING_start/_stopメソッドで区間と数量を指示

+, -, x : 1 flop
÷ : 8 flops （FXの単精度）
　　13 flops（FXの倍精度）
abs(), max() : 1 flops

```
TIMING_start(tm_div_pvec);

flop_count = 0.0;
cbc_div_(src0, sz, gc, &coef, vc, (int*)bcv, v00, &flop_count);

TIMING_stop(tm_div_pvec, flop_count);
```

flop_count 引数を渡して登録する

```
subroutine cbc_div (div, sz, g, ..., flop)
implicit none
integer                                                  ::  i, j, k, ix, jx, kx, g
integer, dimension(3)                                    ::  sz
real                                                     ::  flop
real, dimension(1-g:sz(1)+g, 1-g:sz(2)+g, 1-g:sz(3)+g)   ::  div

ix = sz(1)
jx = sz(2)
kx = sz(3)
flop = flop + real(ix)*real(jx)*real(kx)*49.0
...
```

ループ中の浮動小数点の演算数をカウント
オーバーフロー防止のため、整数演算をrealにキャスト

利用方法の簡単な説明 3

- 測定区間の記述（方法2：演算数やデータ量をHWPCに自動算出させる）
  - TIMING_start/_stopメソッドで区間を指示

```
TIMING_start(tm_div_pvec);

cbc_div_(src0, sz, gc, &coef, vc, (int*)bcv, v00 );

TIMING_stop(tm_div_pvec);
```

```
subroutine cbc_div (div, sz, g, ...,)
implicit none
integer                                                  ::  i, j, k, ix, jx, kx, g
integer, dimension(3)                                    ::  sz
real                                                     ::  flop
real, dimension(1-g:sz(1)+g, 1-g:sz(2)+g, 1-g:sz(3)+g)   ::  div

ix = sz(1)
jx = sz(2)
kx = sz(3)
...
```

利用方法の簡単な説明 4

- 測定メソッドの切り替え、あるいは併用

```
 //@fn プロファイラのラベル取り出し
 //@param 格納番号
 inline const char* get_tm_label(unsigned key) {
    return (const char*)tm_label_ptr[key];
 }

 //@fn タイミング測定開始
 //@param 格納番号
 inline void TIMING_start(unsigned key) {
    // Intrinsic profiler
    TIMING__ PM.start(key);

#ifdef __FX_FAPP
    fapp_start( get_tm_label(key), 0, 0);
#endif
  }

 //@fn タイミング測定終了
 //@param 格納番号
 //@param[in] flopPerTask 「タスク」あたりの計算量/通信量(バイト) (ディフォルト0)
 //@param[in] iterationCount  実行「タスク」数 (ディフォルト1)
 inline void TIMING_stop(unsigned key, REAL_TYPE flopPerTask=0.0, unsigned iterationCount=1) {

#ifdef __FX_FAPP
    fapp_stop( get_tm_label(key), 0, 0);
#endif

    // Intrinsic profiler
    TIMING__ PM.stop(key, flopPerTask, iterationCount);
  }
```

内蔵プロファイラとFXのプロファイラをコンパイラオプション
-D__FX_FAPPで切り替え

利用方法の簡単な説明 5

- ヘッダのインクルード

```
#include "PerfMonitor.h"
```

```
// FX用のプロファイラ
#ifdef __FX_FAPP
#include "/fj_tool/fapp.h"
#endif
```

- クラスライブラリのインスタンス

```
PerfMonitor     PM;
```

- 初期化
  - 並列時のランク番号の割り当て（V-Sphere利用時）？
  - 初期化（ラベルの配列数 tm_END を渡す）

```
// タイミング測定の初期化
PM.initialize(tm_END);
PM.setRankInfo(pn.ID);
PM.setParallelMode(para_mode, C.num_thread, C.num_process);
set_timing_label();
```

利用方法の簡単な説明 2

- サンプリング後の統計処理

```
FILE* fp = NULL;

Hostonly_ {
  if ( !(fp=fopen("profiling.txt", "w")) ) {
    stamped_printf("\tSorry, can't open 'profiling.txt' file. Write failed.\n");
    assert(0);
  }
}

// 測定結果の集計(gathreメソッドは全ノードで呼ぶこと)
PM.gather();

// マスターノードでのみ結果出力(排他測定のみ)
Hostonly_ {
 PM.print(stdout);
 PM.print(fp);

  // 結果出力(非排他測定も)
 if ( C.Mode.Profiling == DETAIL) {
     PM.printDetail(stdout);
     PM.printDetail(fp);
   }
   if ( !fp ) fclose(fp);
 }
```

自己申告:四則以外のflop count

max

```
do k=1,kx
do j=1,jx
do i=1,ix
  vm = vm + max( vm, p(i,j,k) )
end do
end do
end do
```

PA情報のカウンタ値とループ数から求めると

```
max() = 1 flops
```

推定方法

1. FXのPA情報からflop countを得る.
2. ループカウントとflop countから, ループあたりのflop countを求める.
3. 加算分を差し引き, 関数のflop countを得る.

測定flop count（FX）

function	Total flop count	loop count	flop count/ loop	Estimated flop
add	1.84E+09	1.66E+09	1.1	1
subtract	1.84E+09	1.66E+09	1.1	1
mply_f	3.51E+09	1.66E+09	2.1	1
mply_d	3.51E+09	1.66E+09	2.1	1
div_f	1.51E+10	1.66E+09	9.1	8
div_d	2.34E+10	1.66E+09	14.1	13
abs_f	3.50E+09	1.66E+09	2.1	1
abs_d	3.50E+09	1.66E+09	2.1	1
min_f	3.22E+09	1.66E+09	1.9	1
min_d	3.32E+09	1.66E+09	2.0	1
max_f	3.32E+09	1.66E+09	2.0	1
max_d	3.32E+09	1.66E+09	2.0	1
max3_f	4.98E+09	1.66E+09	3.0	2
sign	1.86E+09	1.66E+09	1.1	0
sqrt_f	1.85E+10	1.66E+09	11.1	10
sqrt_d	3.50E+10	1.66E+09	21.1	20
sin_f	5.00E+10	1.66E+09	30.1	29
sin_d	5.33E+10	1.66E+09	32.1	31
cos_f	5.00E+10	1.66E+09	30.1	29
cos_d	5.00E+10	1.66E+09	30.1	29

単精度：_f
倍精度：_d
max3 : max(a, b, c)

exp_f	3.86E+10	1.66E+09	23.2	22
exp_d	4.55E+10	1.66E+09	27.4	26
log_f	3.71E+10	1.66E+09	22.3	21
log_d	4.44E+10	1.66E+09	26.7	25
log10_f	4.18E+10	1.66E+09	25.2	24
log10_d	5.28E+10	1.66E+09	31.8	30
modulo_f	1.75E+10	1.66E+09	10.5	9
modulo_d	1.74E+10	1.66E+09	10.5	9
aint	1.17E+10	1.66E+09	7.0	6
anint	1.83E+10	1.66E+09	11.0	10
ceiling	4.98E+09	1.66E+09	3.0	2
i2dble	3.46E+09	1.66E+09	2.1	1
f2dble	3.50E+09	1.66E+09	2.1	1
floor	4.98E+09	1.66E+09	3.0	2
int	1.66E+09	1.66E+09	1.0	0
nint	1.16E+10	1.66E+09	7.0	6
i2real	3.46E+09	1.66E+09	2.1	1
d2real	3.50E+09	1.66E+09	2.1	1

測定flop count（FX）サマリー

単精度と倍精度で同じ
- 加減乗算：1
- abs, min, max：1
- sin, cos：29

単精度と倍精度で異なる
- 除算：8/13
- sqrt：10/20
- exp：22/26
- log：21/25
- log10：24/30

変換
- aint：6　小数部切り捨て
- nint：6　引数に近い整数値
- anint：10　小数部の四捨五入
- ceiling：2　引数以上で最小の整数値
- floor：2　引数以下で最大の整数値

Cast
- ＊ -> real, ＊ -> double：1
- ＊ -> int：0

符号
- sign：0

Flop countの注意点

- ループ中の定数演算は外に出す
  - コンパイラが自動的に判断し，ソースを変更するので無駄な計算部分のflop countは少なくなる
  - FXのプロファイラよりPMクラスのMFLOPSが大きければ，無駄な計算をしている可能性が高い

- ループ中のif文
  - if文内の演算数がある程度多ければ(10 flop以上)カウンタを使う
    - FXでは浮動小数点で加算した方がよい(整数レジスタ利用の弊害で最適化がされない場合がある)
  - If文内の演算数が少なければ，カウントしない．あるいは，適当に近似

HWPCによる計算量の自動算出

- 性能システムのCPUが内部に持つHWPC
  - CPU種類毎に物理的なカウンタの数が異なる
  - CPU種類毎に取得可能なイベントの種類が異なる
    - papi_avail
- 計測されるイベントの回数
  - プログラムの実行に伴う全ての処理が対象
    - ソースプログラムで書かれている計算式
    - プログラムの実行・制御に必要な追加処理
    - コンパイラによる最適化の副影響
    - 投機的な実行による無駄な計算
  - PMlibの区間設定で興味ある部分に焦点をあてて解釈するとわかりやすい

ハードウエアカウンタについて

- 京・FX10 preset event

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

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について

- Intel Xeon E5 preset event

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

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_L2_DCM 0x80000002 Yes Level 2 data cache misses
PAPI_L2_ICM 0x80000003 No Level 2 instruction cache misses
PAPI_L1_TCM 0x80000006 Yes Level 1 cache misses
PAPI_L2_TCM 0x80000007 No Level 2 cache misses
PAPI_L3_TCM 0x80000008 No Level 3 cache misses
PAPI_TLB_DM 0x80000014 Yes Data translation lookaside buffer misses
PAPI_TLB_IM 0x80000015 No Instruction TLBmisses
PAPI_L1_LDM 0x80000017 No Level 1 load misses
PAPI_L1_STM 0x80000018 No Level 1 store misses
PAPI_L2_STM 0x8000001a No Level 2 store misses
PAPI_STL_ICY 0x80000025 No Cycles with no instruction issue
PAPI_BR_UCN 0x8000002a Yes Unconditional branch instructions
PAPI_BR_CN 0x8000002b No Conditional branch instructions
PAPI_BR_TKN 0x8000002c Yes Conditional branch taken
PAPI_BR_NTK 0x8000002d No Conditional branch not taken
PAPI_BR_MSP 0x8000002e No Conditional branch mispredicted
PAPI_BR_PRC 0x8000002f Yes Conditional branch correctly predicted
PAPI_TOT_INS 0x80000032 No Instructions completed
PAPI_FP_INS 0x80000034 Yes Floating point instructions
PAPI_LD_INS 0x80000035 No Load instructions
PAPI_SR_INS 0x80000036 No Store instructions
PAPI_BR_INS 0x80000037 No Branch instructions
PAPI_TOT_CYC 0x8000003b No Total cycles
PAPI_L2_DCH 0x8000003f Yes Level 2 data cache hits
PAPI_L2_DCA 0x80000041 No Level 2 data cache accesses
PAPI_L3_DCA 0x80000042 Yes Level 3 data cache accesses
PAPI_L2_DCR 0x80000044 No Level 2 data cache reads
PAPI_L3_DCR 0x80000045 No Level 3 data cache reads
PAPI_L2_DCW 0x80000047 No Level 2 data cache writes
PAPI_L3_DCW 0x80000048 No Level 3 data cache writes
PAPI_L2_ICH 0x8000004a No Level 2 instruction cache hits
PAPI_L2_ICA 0x8000004d No Level 2 instruction cache accesses
PAPI_L3_ICA 0x8000004e No Level 3 instruction cache accesses
PAPI_L2_ICR 0x80000050 No Level 2 instruction cache reads
PAPI_L3_ICR 0x80000051 No Level 3 instruction cache reads
PAPI_L2_TCA 0x80000059 Yes Level 2 total cache accesses
PAPI_L3_TCA 0x8000005a No Level 3 total cache accesses
PAPI_L2_TCR 0x8000005c Yes Level 2 total cache reads
PAPI_L3_TCR 0x8000005d Yes Level 3 total cache reads
PAPI_L2_TCW 0x8000005f No Level 2 total cache writes
PAPI_L3_TCW 0x80000060 No Level 3 total cache writes
PAPI_FDV_INS 0x80000063 No Floating point divide instructions
PAPI_FP_OPS 0x80000066 Yes Floating point operations
PAPI_SP_OPS 0x80000067 Yes Floating point operations; optimized to count scaled single precision vector operations
PAPI_DP_OPS 0x80000068 Yes Floating point operations; optimized to count scaled double precision vector operations
PAPI_VEC_SP 0x80000069 Yes Single precision vector/SIMD instructions
PAPI_VEC_DP 0x8000006a Yes Double precision vector/SIMD instructions
PAPI_REF_CYC 0x8000006b No Reference clock cycles

ハードウエアカウンタ Xeon E5 preset とnative

Intel Xeon E5ではPAPI_FP_OPSとPAPI_FP_INSは同じ内容を表示

```
Event name:                    PAPI_FP_OPS
Event Code:                    0x80000066
Number of Native Events:       2
Short Description:            |FP instructions|
Long Description:             |Floating point instructions|
Developer's Notes:            ||
Derived Type:                 |DERIVED_ADD|
Postfix Processing String:    ||
 Native Code[0]: 0x4000001c |FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE|
 Native Event Description: |Counts number of floating point events, masks:Number of SSE double precision FP scalar uops executed|

 Native Code[1]: 0x4000001d |FP_COMP_OPS_EXE:SSE_FP_SCALAR_SINGLE|
 Native Event Description: |Counts number of floating point events, masks:Number of SSE single precision FP scalar uops executed|


$ papi_avail -e  PAPI_DP_OPS
Event name:                    PAPI_DP_OPS
Event Code:                    0x80000068
Number of Native Events:       3
Short Description:            |DP operations|
Long Description:             |Floating point operations; optimized to count scaled double precision vector operations|

 Native Code[0]: 0x4000001c |FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE|
 Native Event Description: |Counts number of floating point events, masks:Number of SSE double precision FP scalar uops executed|

 Native Code[1]: 0x40000020 |FP_COMP_OPS_EXE:SSE_FP_PACKED_DOUBLE|
 Native Event Description: |Counts number of floating point events, masks:Number of SSE double precision FP packed uops executed|

 Native Code[2]: 0x40000021 |SIMD_FP_256:PACKED_DOUBLE|
 Native Event Description: |Counts 256-bit packed floating point instructions, masks:Counts 256-bit packed double-precision|


$ papi_avail -e  PAPI_VEC_DP
Event name:                    PAPI_VEC_DP
Event Code:                    0x8000006a
Number of Native Events:       2
Short Description:            |DP Vector/SIMD instr|
Long Description:             |Double precision vector/SIMD instructions|

 Native Code[0]: 0x40000020 |FP_COMP_OPS_EXE:SSE_FP_PACKED_DOUBLE|
 Native Code[1]: 0x40000021 |SIMD_FP_256:PACKED_DOUBLE|
```

ハードウエアカウンタ SPARC64 VIIIfx 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.|
```

Pmlibの動作確認がとれているシステム

- 京/FX10
  - ログインノードでのクロスコンパイル環境
  - 計算ノードでのネイティブコンパイル環境
  - 富士通コンパイラ＋MPI
- Intel Xeon E5 クラスタ
  - Intelコンパイラ＋IntelMPI
  - GNUコンパイラ+OpenMPI/gnu
  - PGIコンパイラ+OpenMPI/pgi
- 必要なソフトウエア環境
  - C, C++ compiler
  - PAPIを組み込む場合はLinux kernel 2.6.32+
    - (Kernel バージョンが古い場合はPerfmod組み込み必要)

PMlib実習編

実習編

- 実習システムへのログイン
- PMlibのインストール
- 例題プログラムへの組み込み
- 実行結果の解釈
- 例題プログラムの性能向上の検討

PMlibの入手方法

- 公式なPMlibパッケージの入手方法
  - 下記から入手(Download zipボタン)
  - https://github.com/avr-aics-riken/PMlib

- 本日の実習では例題プログラムを追加したパッケージを使用します。
  - 京コンピュータログインノード2,3,4,5,6
  - /tmp/mikami/PMlib-master.2.1.3.rev2.tar.gz

PMlibのインストール　京コンピュータ(1)

- PMlibのインストール作業はログインノードでも計算ノードでも可能。本日はログインノードでインストール実施
- PMlibの利用は計算ノードアプリケーションが行う
- 任意のディレクトリでパッケージを展開。インストール先のディレクトリを　--prefix で指定しconfigureの実施。自動作成されるMakefileを用いて、makeの実施。

```
$ tar -zxf PMlib-master.2.1.3.vsh.tar.gz
$ ls -l
$ drwxr-xr-x 8 a03155 ra000004     4096  5月 26 04:30 PMlib-master
$ cd PMlib-master
$ ./configure CXX=mpiFCCpx CC=mpifccpx FC=mpifrtpx \
    CFLAGS='-Kopenmp,fast  -Ntl_notrt' \
    CXXFLAGS='-Kopenmp,fast -DUSE_PAPI  -Ntl_notrt' \
    --host=sparc64-unknown-linux-gnu   \
    --with-papi=yes  --with-example=yes \
    --prefix=${HOME}/pmlib/install_dir
$ make
$ make install
```

PMlibのインストール　京コンピュータ(2)

- 京でのインストール時間は数分で終了。正常にインストールされると以下のファイルができている。

```
$ ls -CF install_dir
bin/  doc/  include/  lib/  share/

$ ls -go install_dir/bin install_dir/include install_dir/lib

install_dir/bin:
total 4
-rwxr-xr-x 1 1563 May 26 19:18 pm-config

install_dir/include:
total 28
-rw-r--r-- 1 5798 May 26 19:18 PerfMonitor.h
-rw-r--r-- 1 6099 May 26 19:18 PerfWatch.h
-rw-r--r-- 1 1490 May 26 19:18 mpi_stubs.h
-rw-r--r-- 1  627 May 26 19:18 pmVersion.h
-rw-r--r-- 1 2079 May 26 19:18 pmlib_papi.h

install_dir/lib:
total 4136
-rw-r--r-- 1 4219910 May 26 19:18 libPM.a
-rw-r--r-- 1   11938 May 26 19:18 libpapi_ext.a
```

```
$ ls -go ./example/
total 2764
-rw-r--r-- 1   24025 May 26 19:17 Makefile
-rw-r--r-- 1    1649 May 23 00:14 Makefile.am
-rw-r--r-- 1   25696 May 23 00:14 Makefile.in
-rw-r--r-- 1    1130 May 23 00:14 Makefile_hand.fx10.login
-rw-r--r-- 1    1083 May 23 00:14 Makefile_hand.intel
-rwxr-xr-x 1 1062806 May 26 22:33 check_new_api
-rw-r--r-- 1    6096 May 23 00:14 check_new_api.c
-rwxr-xr-x 1 4409884 May 26 22:33 pmlib_test
-rw-r--r-- 1    2181 May 23 00:14 pmlib_test.cpp
-rw-r--r-- 1    1156 May 23 00:14 sub_kernel.c
```

PMlibを用いる　京コンピュータ(1)

- Example/ 以下のサンプルプログラムでPMlibを利用してみる

```
#!/bin/bash
set -x
date; hostname; /opt/FJSVXosPA/bin/xospastop
PMLIB=${HOME}/pmlib/PMlib-master
PMLIB_INCLUDE=-I${PMLIB}/include
PMLIB_LIB=${PMLIB}/src/libPM.a
PAPI_ROOT=/usr
PAPI_LIB="$PAPI_ROOT/lib64/libpapi.a $PAPI_ROOT/lib64/libpfm.a"
PAPI_EXT="$PMLIB/src_papi_ext/libpapi_ext.a”
CXXFLAGS="-Kfast,parallel,openmp ${PMLIB_INCLUDE} ${PAPI_INCLUDE}”
CCFLAGS="-std=c99 -Xg -Kfast,parallel,openmp ${PMLIB_INCLUDE} ${PAPI_INCLUDE}”
LDFLAGS="${PMLIB_LIB} ${PAPI_LIB} ${PAPI_EXT}”
SRC_DIR=${HOME}/pmlib/PMlib-master/example
WKDIR=/data/ra000004/a03155/tmp/check_pmlib ; mkdir -p $WKDIR
cd $WKDIR; if [ $? != 0 ] ; then echo '@@@ Directory error @@@'; exit; fi
cp $SRC_DIR/pmlib_main.cpp main.cpp
cp $SRC_DIR/sub_kernel.c sub.c
mpiFCC -c ${CXXFLAGS}  main.cpp
mpifcc -c ${CCFLAGS}  sub.c
mpiFCC    ${CXXFLAGS}  main.o sub.o ${LDFLAGS}
export OMP_NUM_THREADS=4
mpirun -np 2 ./a.out
```

Pmlibを用いる　京実行結果例 (1)

- 基本プロファイル＋MPIプロセス毎プロファイル

```
Report of Timing Statistics PMlib version 2.1.3
Operator  : RRR
Host name : QQQ
Date      : 2014/05/26 : 23:25:43
Parallel Mode                 :   Hybrid (2 processes x 4 threads)
Total execution time          = 7.231672e-01 [sec]
Total time of measured sections = 7.296531e-01 [sec]

Statistics per MPI process [Node Average]
Label    |     call     |            accumulated time                 |         flop | messages[Bytes]
         |              |   avr[sec]    avr[%]     sdv[sec]  avr/call[sec] |   avr         sdv         speed
---------+--------------+-------------------------------------------------+-----------------------------------
section1 :            2   4.918501e-01   67.41   2.4496e-02  2.459251e-01    0.000e+00   0.000e+00     0.00 Mflops
section2 :            1   2.378030e-01   32.59   1.9418e-03  2.378030e-01    4.000e+09   0.000e+00    16.82 Gflops
---------+--------------+-------------------------------------------------+-----------------------------------
Total    |              7.296531e-01                                  4.000e+09               5.48 Gflops
Performance|                                                                                  10.96 Gflops

-----------------------------------------------------
Elapsed time variation over MPI ranks

section1
MPI_rank       call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :          2  4.745290e-01   65.03  3.464222e-02  2.372645e-01   0.000e+00     0.00 Mflops
#1    :          2  5.091712e-01   69.78  0.000000e+00  2.545856e-01   0.000e+00     0.00 Mflops
section2
MPI_rank       call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :          1  2.391760e-01   32.78  0.000000e+00  2.391760e-01   4.000e+09    16.72 Gflops
#1    :          1  2.364299e-01   32.40  2.746105e-03  2.364299e-01   4.000e+09    16.92 Gflops
```

Pmlibを用いる　京実行結果例 (2)

- 環境変数を追加　export HWPC_CHOOSER=FLOPS

```
Statistics per MPI process [Node Average]
Label    |     call     |            accumulated time                |         flop | messages[Bytes]
         |              |  avr[sec]    avr[%]     sdv[sec]  avr/call[sec] |   avr         sdv         speed
---------+--------------+-------------------------------------------------+-------------------------------
section1 :            2   4.843562e-01   67.16   9.8485e-03  2.421781e-01    8.123e+09   2.828e+00    16.77 Gflops
section2 :            1   2.368304e-01   32.84   2.1871e-03  2.368304e-01    4.033e+09   7.071e-01    17.03 Gflops
---------+--------------+-------------------------------------------------+-------------------------------
Total    |              7.211865e-01                                1.216e+10              16.86 Gflops
Performance|                                                                                33.71 Gflops

-------------------------------------------------
PMlib detected the CPU architecture:
The available Hardware Performance Counter (HWPC) events depend on this CPU architecture.
          Sun
          Fujitsu SPARC64 VIIIfx
HWPC event values of the master rank, sum of threads. count unit in Giga (x 10e9)
-       :      FP_OPS    [GFlops]
section1 :       8.123      17.015
section2 :       4.033      16.919

-------------------------------------------------
Elapsed time variation over MPI ranks
section1
MPI_rank       call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :          2  4.773922e-01   66.20  1.392794e-02  2.386961e-01   8.123e+09    17.02 Gflops
#1    :          2  4.913201e-01   68.13  0.000000e+00  2.456601e-01   8.123e+09    16.53 Gflops
section2
MPI_rank       call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :          1  2.383769e-01   33.05  0.000000e+00  2.383769e-01   4.033e+09    16.92 Gflops
#1    :          1  2.352839e-01   32.62  3.093004e-03  2.352839e-01   4.033e+09    17.14 Gflops
```

Pmlibを用いる　京実行結果例 (3)

- 環境変数を追加　export HWPC_CHOOSER=FLOPS,VECTOR

PMlib detected the CPU architecture:
The available Hardware Performance Counter (HWPC) events depend on this CPU architecture.
Sun
Fujitsu SPARC64 VIIIfx
HWPC event values of the master rank, sum of threads. count unit in Giga (x 10e9)

- :	FP_OPS	[GFlops]	VEC_INS	FMA_INS
section1 :	8.123	17.010	3.180	0.881
section2 :	4.033	16.758	1.581	0.435

HWPC events legend: count unit in Giga (x 10e9)
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

Pmlibを用いる　京実行結果例 (4)

- 環境変数を変更　export HWPC_CHOOSER=BANDWIDTH

```
Statistics per MPI process [Node Average]
Label    |     call     |            accumulated time                |         flop | messages[Bytes]
         |              |   avr[sec]    avr[%]     sdv[sec]  avr/call[sec] |   avr         sdv         speed
---------+--------------+-------------------------------------------------+-------------------------------------
section1 :            2   4.823370e-01   66.93   3.7025e-03  2.411685e-01    1.007e+10   6.975e+08    19.45 GB/sec
section2 :            1   2.383659e-01   33.07   2.4961e-03  2.383659e-01    5.227e+09   6.785e+08    20.42 GB/sec
---------+--------------+-------------------------------------------------+-------------------------------------
Total    |               7.207029e-01                                 1.530e+10               19.77 GB/sec
Performance|                                                                                    39.54 GB/sec

-----------------------------------------------------
PMlib detected the CPU architecture:
The available Hardware Performance Counter (HWPC) events depend on this CPU architecture.
          Sun
          Fujitsu SPARC64 VIIIfx
HWPC event values of the master rank, sum of threads. count unit in Giga (x 10e9)
-        :      L2_TCM    L2_WB_DM    L2_WB_PF  [Mem GB/s]
section1 :       0.082       0.000       0.000      22.025
section2 :       0.045       0.000       0.000      23.763

          HWPC events legend: count unit in Giga (x 10e9)
                    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
```

プロファイルの検討と最適化への反映

- サンプルプログラムでの検討
  - STREAMベンチマークプログラムのC++ OpenMP版

```
$ cd src_others
$ ls –go
-rw-r--r-- 1     931 May 27 13:06 Makefile
-rw-r--r-- 1    1943 May 27 14:51 pmlib_stream.cpp
-rw-r--r-- 1    4073 May 27 15:37 stream.cpp
-rwxr-xr-x 1     740 May 27 15:25 x.pmlib-K.sh
$ make
$ ls -go
        …
-rwxr-xr-x 1 5489281 May 28 10:36 pmlib_stream.ex

$ # Edit x.pmlib-K
export HWPC_CHOOSER=BANDWIDTH
$ pjsub x.pmlib-K.sh
```

- STREAMの標準出力とPmlib出力の比較・解説

PMlibを用いる　Intel Xeon(1)

- Example/ 以下のサンプルプログラムでPMlibを利用してみる
  - Module環境が設定されてる場合の例

```
#!/bin/bash
module load intel impi papi pmlib/intel
CFLAGS="-O3 -openmp -vec-report -openmp-report”

mpicxx -c ${CFLAGS} ${INCLUDES} main.cpp
mpicc  -c ${CFLAGS} ${INCLUDES} sub.c
mpicxx    ${CFLAGS} ${INCLUDES}  main.o sub.o ${LDFLAGS}

export HWPC_CHOOSER=FLOPS
export OMP_NUM_THREADS=4
mpirun -np 2 ./a.out
```

Module環境が設定されていない場合は別途変数を指定しておく

```
INCLUDES=-I/usr/local/pmlib/pmlib-2.1.2-intel/include -I/usr/local/papi/papi-5.2.0/intel/include
LDFLAGS=-L/usr/local/pmlib/pmlib-2.1.2-intel/lib -lPM -lpapi_ext -L/usr/local/papi/papi-5.2.0/intel/lib -lpapi -lpfm
```