PerfMonitorクラスを用いたプロファイリング情報の取得

keno@riken.jp

2012-04-28

PerfMonitorクラス

- 内蔵プロファイラ
  - 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
  - プログラム終了時に，各MPIランクの情報をマスターランクに集めて，統計処理して出力
  - 簡易モードと詳細モード
    - 詳細モードでは同期待ち情報など
  - どの計算機でも，実際の実行性能をサンプリング可能
    - ユーザサイトの情報が，特別なオプションなしにそのまま得られる
    - 演算に与える影響は，計時記録のみなので無視できる程度
- 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

基本プロファイル例

```
Report of Timing Statistics

Parallel Mode                    :   Hybrid (2 processes x 4 threads)

Total execution time            = 4.212366e+00 [sec]
Total time of measured sections = 4.160442e+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
------------------------+--------------+------------------------------------------------+-------------------------------------
Projection_Velocity     :         4231   2.120372e+00   50.97   5.2882e-03  5.011516e-04    2.995e+10   0.000e+00    13.15 Gflops
Poisson_SOR2_SMA        :         8462   8.246812e-01   19.82   2.9627e-02  9.745701e-05    9.982e+09   0.000e+00    11.27 Gflops
Pseudo_Velocity         :           50   4.601424e-01   11.06   7.5712e-03  9.202849e-03    2.615e+09   0.000e+00     5.29 Gflops
Sync_Pressure           :         8462   2.605188e-01    6.26   7.6573e-03  3.078690e-05    2.773e+08   0.000e+00  1015.04 MB/sec
Poisson_Norm_Div_max    :         4231   1.144669e-01    2.75   1.8912e-03  2.705434e-05    1.386e+09   0.000e+00    11.28 Gflops
Projection_Velocity_BC  :         4231   1.107653e-01    2.66   1.0994e-02  2.617947e-05    1.563e+08   0.000e+00     1.31 Gflops
A_R_Poisson_Residual    :         8462   6.619477e-02    1.59   2.7398e-02  7.822592e-06    1.354e+05   0.000e+00     1.95 MB/sec
assign_Const_to_Array   :         4281   5.136287e-02    1.23   1.3673e-03  1.199787e-05    0.000e+00   0.000e+00     0.00 Mflops
A_R_Poisson_Norm        :         4231   4.450220e-02    1.07   1.3698e-02  1.051813e-05    1.354e+05   0.000e+00     2.90 MB/sec
Copy_Array              :          100   1.937538e-02    0.47   2.9908e-04  1.937538e-04    0.000e+00   0.000e+00     0.00 Mflops
Divergence_of_Pvec      :           50   1.908743e-02    0.46   5.2641e-05  3.817487e-04    1.835e+08   0.000e+00     8.95 Gflops
Sync_Pseudo_Velocity    :           50   1.709867e-02    0.41   7.4337e-03  3.419733e-04    9.830e+06   0.000e+00   548.29 MB/sec
Variation_Space         :           50   1.561308e-02    0.38   3.7020e-04  3.122616e-04    8.192e+07   0.000e+00     4.89 Gflops
Search_Vmax             :           50   8.687973e-03    0.21   2.9272e-03  1.737595e-04    2.949e+07   0.000e+00     3.16 Gflops
Pvec_Euler_Explicit     :           50   5.660295e-03    0.14   1.1770e-04  1.132059e-04    2.621e+07   0.000e+00     4.31 Gflops
Sync_Velocity           :           50   5.069375e-03    0.12   3.4467e-05  1.013875e-04    3.932e+07   0.000e+00     7.22 GB/sec
Allocate_Arrays         :            2   4.770339e-03    0.11   5.5245e-04  2.385169e-03    0.000e+00   0.000e+00     0.00 Mflops
A_R_Vmax                :           50   3.973961e-03    0.10   3.6906e-03  7.947922e-05    1.600e+03   0.000e+00   393.18 KB/sec
Pseudo_Vel_Flux_BC      :           50   2.162278e-03    0.05   1.7480e-04  4.324555e-05    3.692e+06   0.000e+00     1.59 Gflops
Poisson_BC              :         8462   1.696825e-03    0.04   9.9093e-05  2.005229e-07    0.000e+00   0.000e+00     0.00 Mflops
Poisson_Src_VBC         :           50   1.295030e-03    0.03   1.2410e-04  2.590060e-05    1.847e+06   0.000e+00     1.33 Gflops
Sync_Variation          :           50   7.169843e-04    0.02   3.9544e-04  1.433969e-05    9.600e+03   0.000e+00    12.77 MB/sec
Velocity_BC             :         4231   6.790757e-04    0.02   1.1076e-04  1.605001e-07    0.000e+00   0.000e+00     0.00 Mflops
Poisson_Setup_for_Itr   :         4231   6.543994e-04    0.02   7.5614e-05  1.546678e-07    0.000e+00   0.000e+00     0.00 Mflops
History_Stdout          :           50   3.979206e-04    0.01   7.8885e-04  7.958412e-06    0.000e+00   0.000e+00     0.00 Mflops
History_Base            :           50   2.369881e-04    0.01   4.5888e-04  4.739761e-06    0.000e+00   0.000e+00     0.00 Mflops
History_Domain_Flux     :           50   1.420975e-04    0.00   2.7545e-04  2.841949e-06    0.000e+00   0.000e+00     0.00 Mflops
Domain_Monitor          :           50   6.467104e-05    0.00   6.4399e-06  1.293421e-06    6.000e+02   0.000e+00     8.85 Mflops
Velocity_BC_Update      :           50   2.884865e-05    0.00   1.1183e-06  5.769730e-07    0.000e+00   0.000e+00     0.00 Mflops
Pseudo_Velocity_BC      :           50   2.312660e-05    0.00   4.4604e-06  4.625320e-07    0.000e+00   0.000e+00     0.00 Mflops
------------------------+--------------+------------------------------------------------+-------------------------------------
Total                   |              4.160442e+00                                      4.441e+10                 9.94 Gflops
   Performance          |                                                                                       39.77 Gflops
```

詳細プロファイル

- MPIの各プロセス毎の平均情報を出力

```
Pvec. Euler Explicit
                call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :           20  3.755836e-01    0.37  2.026558e-04  1.877918e-02   3.775e+08   958.51 Mflops
#1    :           20  3.757863e-01    0.37  0.000000e+00  1.878932e-02   3.775e+08   957.99 Mflops
#2    :           20  3.755608e-01    0.37  2.255440e-04  1.877804e-02   3.775e+08   958.57 Mflops
#3    :           20  3.756111e-01    0.37  1.752377e-04  1.878055e-02   3.775e+08   958.44 Mflops

Sync. Pseudo Velocity
                call       accm[s] accm[%]    waiting[s]  accm/call[s]    flop|msg     speed
#0    :           20  1.492591e-01    0.15  3.340559e-01  7.462955e-03   3.146e+07   200.99 MB/sec
#1    :           20  1.513844e-01    0.15  3.319306e-01  7.569218e-03   3.146e+07   198.17 MB/sec
#2    :           20  4.833150e-01    0.48  0.000000e+00  2.416575e-02   3.146e+07    62.07 MB/sec
#3    :           20  4.830239e-01    0.48  2.911091e-04  2.415119e-02   3.146e+07    62.11 MB/sec
```

利用方法の簡単な説明 1

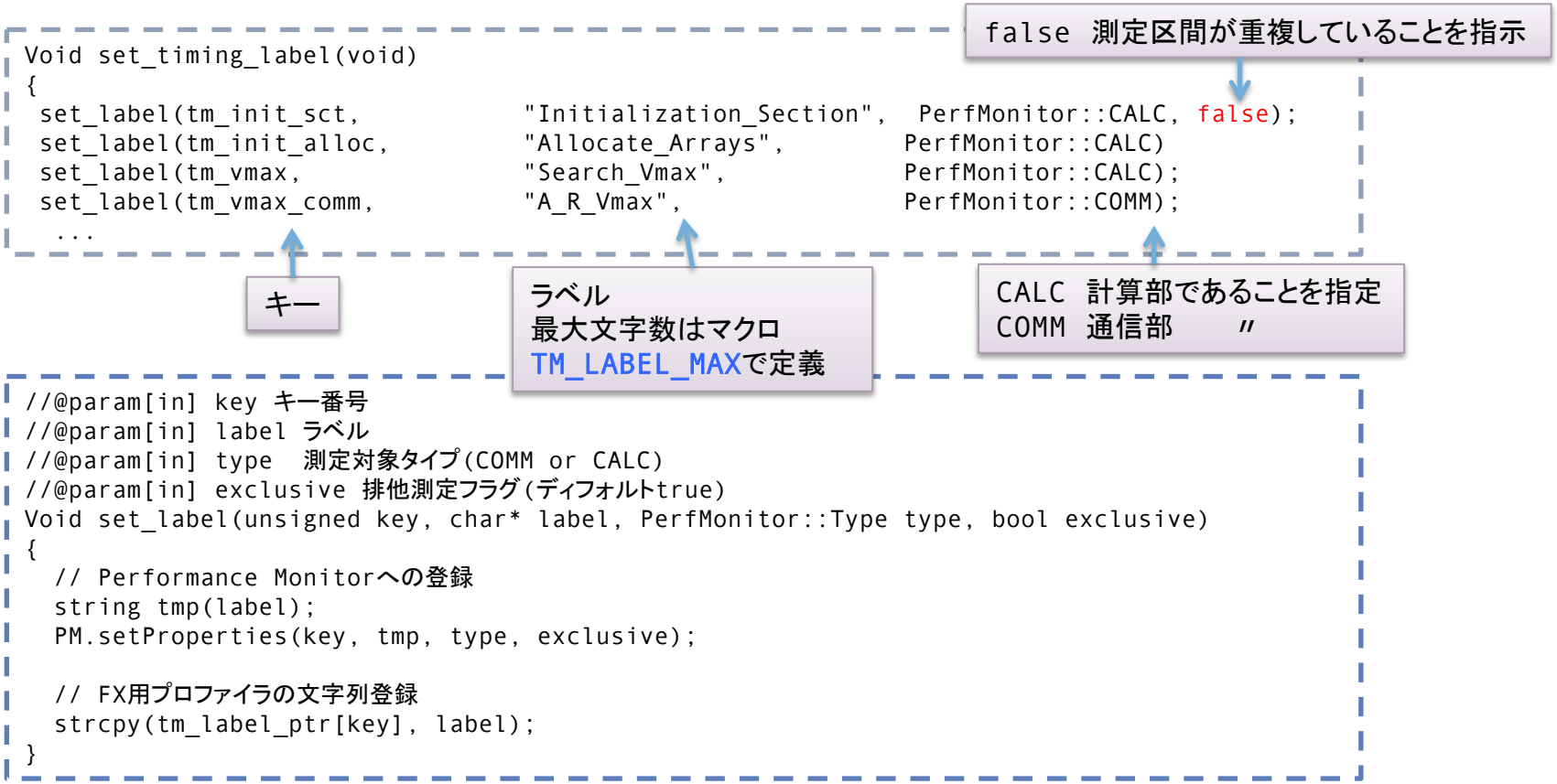
- 計時区間のキーと総数 tm_END

```
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

- タイミングラベルの配列宣言とラベル指定



利用方法の簡単な説明 3

- 測定区間の記述
  - TIMING_start/_stopメソッドで区間を指示

+, -, 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 4

- 測定メソッド

```
 //@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
```

- ヘルパーでラベル配列確保

```
// プロファイラ用のラベル宣言
  char tm_label_ptr[tm_END][TM_LABEL_MAX];
```

マクロで定義

ラベルはstringを使うとエラーとなる
（FXのバグ？）ので、静的な配列を使う。

- 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);
 }
```

flop countの測定

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

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

精度改善版の比較

	FX Profiler		PM class	
Range	**Time(S)**	**MFLOPS**	**MFLOPS**	**Diff [%]**
Projection Velocity	56.05739	1,743.3	1,500.0	-14.0
Pseudo Velocity	17.33970	1,915.9	1,800.0	-6.1
Poisson SOR2 (SMA)	15.39081	1,908.5	1,820.0	-4.6
Poisson Norm Div. max	2.72127	1,563.0	1,430.0	-8.5
Divergence of Pvec.	1.85583	1,401.2	1,180.0	-15.8
Projection Velocity BC	0.77164	153.1	144.7	-5.5
Averaging Space	0.39951	2,654.2	2,250.0	-15.2
Pvec. Euler Explicit	0.38077	881.2	839.2	-4.8
Pseudo Vel. Flux BC	0.07656	333.9	292.4	-12.4
Search Vmax	0.06411	6,225.1	5,450.0	-12.5
Poisson Src. VBC	0.04008	147.4	139.3	-5.5

PM classは平均10%程度少なめのMFLOPS値
→　陽には現れない浮動小数点計算
　　投機実行の分

Flop countの数え方に気をつける必要あり

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