スレッド性能測定機能を備えた性能モニタライブラリ PMlib

PMlib that provides a function of thread performance monitoring

○ 小野 謙二，九大 RIIT, 〒 819-0395 福岡市西区元岡 744, E-mail：keno@cc.kyushu-u.ac.jp
三上 和徳，理研 R-CCS, 〒 650-0047 神戸市中央区港島南町 7-1-26, E-mail：kazunori.mikami@riken.jp
Kenji ONO, Research Institute for Information Technology, Kyushu University
Kazunori Mikami, R-CCS RIKEN

PMlib is an open-source software to measure the behavior of the applications and is applicable to a wide range of architectures. This paper introduces a measurement function of threading performance of the PMlib at run-time of scientific applications. The usage of PMlib is very simple; the only things to do is to insert several APIs such as instance, initialization, measurement and reporting, to a source code.

1. はじめに

高性能なプログラムを記述する上では、ソフトウェアアルゴリズムの様々な並列性を抽出し、それらをハードウェアの各レベルで並列動作するデバイス（CPU, Core, SIMD 演算器など）にマッピングし、プログラム実装する必要がある。特に、データを継ぎ目なく供給し、SIMD 演算器にを常に稼働させることは高性能化の鍵である。この点は、ループ内の演算数に対するデータ移動量の比、B/F (Bye/Flop) が大きくなる場合に、特に問題となる。しかしながら、近年のアーキテクチャは演算性能とデータ転送性能の乖離が大きくなるメモリウォールの問題を抱えており、データ供給に特に配慮する必要があるが、データ供給は陽にプログラムできないことが高性能プログラムの記述を難しくしている要因である。データ供給を効率的に行うためには、（NUMA を意識した）キャッシュの利用、SIMD 専用のデータロード命令など、データ構造とメモリ上の配置に注意することが重要であるが、陽的にプログラムできる部分ではないことが、性能チューニングを難しくしている要因である。

プログラムチューニングを行うためには、演算器の稼働状況やデータの移動状況など実行時の計算機の振る舞いを知ることが必要である。これまでにも多くの性能測定ツールが開発されてきたが、それぞれ特徴がある。必要な情報を得るため、多くの場合、複数のツールが利用される。これらのツールは、サンプリングまたはトレーシングに分類され、オーバーヘッドはトレースの方が大きい。トレーシングには Scalasca [2] [6]、Extrae [7]、サンプリングには PAPI [3]、Linux perf tools、Intel VTune [4]、PGI Profiler [5] などがある。中でも、コアに内蔵されているハードウェア・パフォーマンス・カウンタ（HWPC）にアクセスし、直接演算ユニットなどの稼働情報を取得できる PAPI [3] の利用により、実行時の精度が高い情報が得られる。

これらのツールは、プログラム開発時に実際の動作を想定してプロファイルをとり、その結果からプログラムのチューニングを行うことに利用されることが多い。しかしながら、実行時のパラメータによって動作が大きく異なるプログラムがある。例えば、領域分割型の MPI 並列流体解析プログラムで粒子追跡を行う場合、粒子の分布に応じてロードバランスが崩れ、並列性能が低くなることなどがある。このような場合、実問題を計算しているときの状況を知る必要がある。多くのチューニングツールは、静的な状況を分析することに向いており、プロダクションランの時の実行性能は主にトレース機能 [8] を用いて分籍されることが多い。

PMlib [1] [9] は、プロダクションランの時の実行性能を低コストで簡便に取得することを目的として開発された性能モニターツールであり、プログラム実行へのインパクトが少ない特徴をもつ。本論文では、この PMlib にスレッド並列時の性能測定機能を追加し、マルチコアの振る舞いが分析可能となったので、その機能を紹介する。

2. PMlib

2.1 概要

PMlib は多くのプラットフォームに対応した性能測定機能を提供するオープンソースライブラリである。PAPI を用いて HWPC へアクセスし、カウンタ値を取得し、その情報から FLOPS やバンド幅、キャッシュヒット率などの情報を計算する。利用方法は簡単であり、ライブラリの初期化、プロパティ設定の後、測定区間の前後に開始と終了を指示する API を呼び出すだけである。測定が終了した後は、レポート出力を指示する。レポートには、基本的な情報を提示するモード (List 1)、プロセス毎の情報を提示するモード (List 2)、スレッド毎の情報を提示するモード (List 3) の 3 つのレベルがある。

2.2 API

PMlib は C++と Fortran ソースコード用の API を提供する。Table 1 は C++用の API である。

Tab. 1: List of basic APIs

C++	function
initialize	initial setup
setProperties	set sections property
start	start of section
stop	end of section
print	basic report
printDetail	per process report
printThreads	per thread information

2.3 測定モード

PMlib は幾つかの測定モードをもつ。

FLOPS|BANDWIDTH|VECTOR|CACHE|CYCLE|user

これらの測定モードは、どれか一つだけを指定できる。HWPC の数が限られているので一度に全てのイベントは取得できないため、目的に応じたイベントだけを取得するようにしていることによる。測定モードは実行シェルで環境変数 HWPC_CHOOSER を指定し、プログラムへ反映する。指定できる環境変数の値は下記になる。

(1) FLOPS
1 秒あたりの実行浮動小数点演算数を取得する。

(2) BANDWIDTH
メモリやキャッシュのバンド幅情報を取得する。

(3) VECTOR
SIMD 命令数（AVX2, AVX-512）や浮動小数点演算数のうち、ベクトル命令数の割合などを取得する。

(4) CACHE
キャッシュヒット率、ミス率などの情報を取得する。

(5) CYCLE
サイクル情報や、サイクルあたりの命令発行数をを取得する。

2.4 インスツルメント

実際のソースコードでの API の使い方を下記に示す。`mykernel()` が測定対象である。initialize() と setProperties() が初期設定、start(), stop() で測定区間を時間を計測する。print(), printDetail() でレポート出力を指定する。

```
#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;
}
```

2.5 スレッド性能測定機能

プロセス毎の OpenMP スレッド別統計情報について説明する。OpenMP 並列領域の外側で start()/stop() 測定を行う場合、PMlib の start()/stop() 間で OpenMP 並列計算を実行する。HPC アプリケーションでよく現れる、parallel do 型/for 型並列処理をサポートする。

一方、OpenMP 並列領域の内部で start()/stop() 測定を行う場合、OpenMP の各スレッドが非同期的に計算を行うような粗粒度のスレッド処理に対応する。

3. まとめ

性能測定ライブラリ PMlib は、利用法が簡単であり、ポータブルな、オープンオースライブラリである。PMlib の基本機能とスレッド毎の統計情報の取得について説明した。スレッド性能の取得機能により、スレッドの待ちやスレッド毎の性能を観察することができ、より詳細な実行時の情報を知ることができるようになった。今後、この機能を用いてチューニングでの活用などを行っていく。

参考文献

(1) PMlib, http://avr-aics-riken.github.io/PMlib/

(2) Scalasca, Jülich Supercomputing Centre available from http://www.scalasca.org/

(3) PAPI, Performance API, University of Tennessee, available from http://icl.cs.utk.edu/papi/software/index.html

(4) Intel Corporation, Intel VTune ® Amplifier, https://software.intel.com/en-us/intel-vtune-amplifier-xe/

(5) NVIDIA Corporation, PGI profiler, https://www.pgroup.com/products/

(6) Geimer, Markus; et al., The Scalasca Performance Toolset Architecture, Concurr. Comput. : Pract. Exper., Vol.22, No.6, pp.702–719, 2010.

(7) Barcelona Supercomputing Center, BSC Performance Tools, available from http://www.scalasca.org/

(8) Knüpfer, Andreas and Brendel, Ronny and Brunst, Holger and Mix, Hartmut and Nagel, Wolfgang E., Introducing the Open Trace Format (OTF), Proceedings of the 6th International Conference on Computational Science - Volume Part II, ICCS'06, pp.526–533, Reading, UK, doi:10.1007/11758525_71, 2006.

(9) Mikami, K., Ono, K., and Nonaka, J., Performance evaluation and visualization of scientific applications using PMlib, CANDAR 2018, to be appeared.

List 1. An example of basic report.

```
# 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

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
```

List 2. An example of detail report.

```
# PMlib hardware performance counter (HWPC) Report -------------------------
Label   Section-A
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.600e+09  1.300e+02  2.177e+09
Rank     1 :  1.600e+09  1.230e+02  2.178e+09
Label   Section-B
Header  ID :     SP_OPS     DP_OPS    [Flops]
Rank     0 :  1.600e+09  1.160e+02  1.088e+09
Rank     1 :  1.600e+09  1.130e+02  1.089e+09
```

List 3. An example of thread report. In this case, 4 threads are used. The section A is executed 2 time for all processes. In the section B, thread 1 and 3 are executed 4 times while thread 0 and 2 are idle.

```
# PMlib Thread Report for MPI rank 0  ----------------------
Label  Section-A
Thread  call  time[s]  ti/tav[%]     SP_OPS     DP_OPS    [Flops]
   0       2  7.347e-01  100.0  4.000e+08  3.100e+01  5.445e+08
   1       2  7.347e-01  100.0  4.000e+08  3.100e+01  5.445e+08
   2       2  7.348e-01  100.0  4.000e+08  3.500e+01  5.444e+08
   3       2  7.351e-01  100.0  4.000e+08  3.300e+01  5.442e+08
Label  Section-B
Thread  call  time[s]  ti/tav[%]     SP_OPS     DP_OPS    [Flops]
   0       0  0.000e+00    0.0  0.000e+00  0.000e+00  0.000e+00
   1       4  1.469e+00  100.0  8.001e+08  5.700e+01  5.445e+08
   2       0  0.000e+00    0.0  0.000e+00  0.000e+00  0.000e+00
   3       4  1.470e+00  100.0  8.001e+08  5.900e+01  5.442e+08
```