Vol.2018-HPC-163 No.1
2018/2/28

PMlibを用いた計算性能測定と性能可視化手法

三上 和徳[1,a)]　小野 謙二[1,†1,b)]

概要：オープンソースライブラリ PMlib を用いたアプリケーションの性能評価手法について報告する．HPC システムの仕様上の最大計算性能とアプリケーション実行時に達成される実行性能との間には，様々な要因により乖離が生じる．この違いを議論する場合には，演算器の並列性やメモリ・キャッシュ各階層間のデータ移動性能などのハードウエアの動作特性を中心に評価する視点と，アプリケーションがソースプログラムレベルで要求する数値計算上の計算量とコンピュータシステムが実際に実行する命令に基づいた計算量との違いを評価する視点がともに有効である．PMlib は数値計算上の計算量を明示的に測定する機能と，HWPC が記録する計算量を測定する機能とを有し，両者の違いを定量的に評価することを可能とする．また HWPC 測定時には内部で PAPI 低レベル API を利用し，一般に選択と解釈が容易ではない各種ハードウエアイベント統計情報をカテゴリ分けしてアプリ利用者が評価しやすい情報として選択出力する．これらの異なる基準での計算量あるいは計算命令の実行状態を統計的に評価することによって，アプリケーションの HPC システムにおける実行性能発現を理解する一助とすることが可能である．
本報告では PMlib を用いたアプリの性能評価手法を説明し，Intel Xeon サーバおよび富士通 FX100 上での評価事例を紹介する．

1. はじめに

研究の目的

計算科学アプリケーションの開発および利用の各局面において，利用する HPC システム上で性能評価作業を実施することが頻繁に行われる．これは主に，アプリケーションの性能特性を把握した後，ソフトウエア上の最適化を実施して高速処理を実現するための可能性を探るために行われる．このような性能評価作業が頻繁に実施される背景として，HPC システムの仕様上の最大性能とアプリケーションが実際に達成する実行性能との差が，アプリの種類によっては非常に大きいことがある．

最大性能と実行性能との違いに関して演算器の並列性やメモリおよびキャッシュ階層構成における局所性などのハードウエアの動作特性と関連づけられて多くの研究が行われている．

アプリケーションがソースプログラムレベルで要求する数値計算上の計算量と，コンピュータシステムが実際に実行する命令を　HWPC（ハードウエア性能カウンタ）で測定した計算量との間ではしばしば乖離がある．この事は性能評価において重要な問題点として考慮されなければならない．

PMlib は数値計算上の計算量を明示的に測定する機能と，HWPC が記録する計算量を測定する機能とを有し，両者の違いを定量的に評価することを可能とする．HWPC 測定には内部で PAPI 低レベル API を利用し，一般に選択と解釈が容易ではない各種ハードウエアイベント統計情報をカテゴリ分けしてアプリ利用者が評価しやすい情報として選択出力する．

出力情報としてはアプリ実行中に蓄積された統計情報を時間平均化した標準レポートに加え，経過時間軸に沿った動的な情報の出力も可能であり，計算性能の時系列挙動を可視化する Web ブラウザパッケージとの連携利用が可能な構成となっている．

本報告ではまず計算科学的観点での計算量とシステム評価的観点での計算量との違いについていくつかの評価事例を示す．

次に，計算量から計算性能を求める段階で，その性能を律速するハードウエアの動作特性を読み取る PMlib の機能について事例を示す．

2. 計算量と性能評価

2.1 計算科学的観点での性能

計算量という用語を用いる場合，計算科学アプリケーションの開発者は，Fortran 言語や C++言語などで記述さ

[1] 理化学研究所 計算科学研究機構
[†1] 現在，九州大学 情報基盤研究開発センター
[a)] kazunori.mikami@riken.jp
[b)] keno@cc.kyushu-u.ac.jp

れたソースプログラム上で計上される数値計算量すなわち演算回数の合計値としての計算量 OP は以下の式で表わされる．

$$OP_{source} = \#add + \#sub + \#mult + \#div + \#max + \#sqrt + ...$$

ソースプログラムで記述される四則演算や基本的な数学関数ごとそれぞれに計算の「重たさ」が異なるため，アプリケーションの計算負荷としてはその「重たさ」を係数 c_{op} として考慮した以下の式で計算量を表現することができる．

$$OP_{weight} = c_{add} \times \#add + c_{sub} \times \#sub + c_{mult} \times \#mult + c_{div} \times \#div + c_{max} \times \#max + c_{sqrt} \times \#sqrt + ...$$

計算性能はこの計算量を計算経過時間で割った値として定義される．

$$Perf = OP/Time$$

HPC アプリケーションにおいて所与の計算を効果的に達成するために計算時間の短縮をめざす場合，必要な計算量を削減するアルゴリズムを開発・採用するというアプローチが多いが，そのような場合の計算量とは主に上記した計算科学的観点での評価が意図される．このような計算科学的観点での計算性能を以降数値計算性能と呼ぶことにする．

HPC システムの性能評価に長く利用されてきた Linpack（HPCC）が出力する Gflops 値も加算・乗算の「重たさ」を等しく 1 として上記結合式を評価した計算量から，計算性能を次式で求めている．

$$Gflops = N^2 \times (2/3 * N + 3/2) \times 1.0^{-9}/Time$$

2.2 システム評価的観点での性能

アプリケーションを HPC システム上で実行する段階では，ソースプログラムを言語コンパイラを通して生成した実行プログラムの機械語命令列としてスケジューリング処理することになる．ソースプログラムでは単純な計算式であっても，実際に実行されるのは複数の命令列であり，命令の種類・量・命令に対応する計算量（演算数）は，実行する HPC システムのアーキテクチャおよびコンパイラなど言語処理系ソフトウエアの最適化水準に依存する部分が大きい．

HPC システムが実際に処理するこの計算量を経過時間で割った値がシステム評価的観点での計算性能として定義される．

この計算量を測定するためには，HPC システムに通常装備されている HWPC(Hardware Performance Counter) に記録されるイベントカウント数を利用することが可能である．適切な API によって HWPC に記録された情報へアクセスし，実行された命令数を測定して計算量を求める．

例えば後出の Intel Xeon Skylake Gold プロッセサの浮動小数点演算に属する計算量は HPWC で計上されたイベント数を基にして以下の式により求められる．

$fpsp1$: "$FP\_ARITH : SCALAR\_SINGLE$";
$fpsp4$: "$FP\_ARITH : 128B\_PACKED\_SINGLE$";
$fpsp8$: "$FP\_ARITH : 256B\_PACKED\_SINGLE$";
$fpsp16$: "$FP\_ARITH : 512B\_PACKED\_SINGLE$";
$fpdp1$: "$FP\_ARITH : SCALAR\_DOUBLE$";
$fpdp2$: "$FP\_ARITH : 128B\_PACKED\_DOUBLE$";
$fpdp4$: "$FP\_ARITH : 256B\_PACKED\_DOUBLE$";
$fpdp8$: "$FP\_ARITH : 512B\_PACKED\_DOUBLE$";

$$OPhwpc = fpsp1 + fpdp1 + 4.0 * fpsp4 + 8.0 * fpsp8 + 16.0 * fpsp16 + 2.0 * fpdp2 + 4.0 * fpdp4 + 8.0 * fpdp8;$$

富士通 FX100 のプロッセサであれば，

$fpdp1$: "$1FLOPS\_INSTRUCTIONS$";
$fpdp2$: "$2FLOPS\_INSTRUCTIONS$";
$fpdp4$: "$4FLOPS\_INSTRUCTIONS$";
$fpdp8$: "$8FLOPS\_INSTRUCTIONS$";
$fpdp16$: "$16FLOPS\_INSTRUCTIONS$";

$$OPhwpc = fpdp1 + 2.0 * fpdp2 + 4.0 * fpdp4 + 8.0 * fpdp8 + 16.0 * fpdp16;$$

このように測定される計算量を以下，HWPC 計算量とよび，HWPC 計算量を基準とした計算性能を HWPC 計算性能と呼ぶことにする．いわゆるシステム性能に主眼がおかれた文脈で使用されることが多い．

HWPC 測定性能が高く表示されるにもかかわらず，数値計算上の性能は低いという状況は，本節で述べた観点が違う事による計算量の定義の違い，およびハードウエアの有効利用の状況の双方の影響を受けた結果であるといえる．

3. PMlib

3.1 PMlib について

PMlib はアプリケーションの計算性能モニター用のクラスライブラリであり，オープンソースソフトウエアとして公開されている．[7]

PMlib は前述した 2 種類の計算性能（数値計算性能と HWPC 計算性能）を測定する機能を持つ．アプリのソース中に測定区間を指定して，測定区間毎の統計情報をアプリ終了時・あるいは計算途中の指定位置で出力する．

測定区間は区間の名称，測定計算量の種類（演算量｜データ移動量），排他性（排他的｜非排他的），蓄積計算量からなる少数の属性を持つ．アプリ終了時の出力レポートに測定区間単位，プロセス単位，アプリ単位での計算性能

が表示される．出力する情報の内容はアプリ実行時のオプションとして環境変数を指定する事により制御される．

3.2 PMlib API の利用

PMlib は C++および Fortran プログラムから呼び出して利用することができる．以下は PMlib API を利用する Fortran ソースプログラムの例である．

```
1  program main
2  call f_pm_initialize (nWatch)
3  call f_pm_setproperties ("Section1" icalc,
       iexcl)
4  call f_pm_start ("Section1")
5  call MyComputation (fops)          !cx measured
       computation
6  call f_pm_stop ("Section1", fops, ncall)
7  call f_pm_print ("", isort)
8  call f_pm_printdetail ("", ilegend, isort)
9  end
```

3.3 測定する計算量の指定

数値計算上の計算性能を測定したい場合は，PMlib API の引数として該当区間の計算量を評価式の形で明示的に指定して積算する．ソースプログラムに記述された計算式の全計算回数を数え上げる事に等しい．その際に計算の種類ごとの「重たさ」の設定は任意である．

HWPC 測定値を基準とした（システム評価的観点での）計算性能を測定したい場合は，該当区間の HWPC イベント統計情報を PMlib に読み取らせ計算量を積算させるため，API へ引数として渡される値は用いられない．HWPC の読み取りには内部で PAPI 低レベル API を利用し，アプリ利用者が評価しやすい情報として自動的に選択出力する．

HWPC イベントをアプリから直接 PAPI API でアクセス利用するのは一般に容易ではない．プリセットされた基本的な HWPC イベントだけではなく，より詳細な HWPC イベントを選択抽出するために，native イベント群とそのマスクを調査する必要がある．例えば後出の Skylake Gold プロッセサを搭載したサーバで papi_native_avail コマンドの出力を確認すると 5000 行以上の情報が出力され，その出力からプロセッサアーキテクチャに依存する注目する計算性能（計算量）を構成するにイベント名とイベントコードを調査し，少数の物理的なカウンターに選択登録するするだけでも相当な負担である．

PMlib を組み込んだアプリケーションでは環境変数 HWPC_CHOOSER を右のカテゴリ値のどれかに設定すると，上記のような調査・妥当な HWPC イベントの選択登録などを内部で実施し，計算量の統計を収集することができる．
カテゴリ値：FLOPS |BANDWIDTH|VECTOR|CACHE

PMlib ではプロセス単位で性能情報を取得する設計となっていて，プロセスが OpenMP スレッドを発生する場合，各スレッドの性能情報は帰属するプロセスに集計される．

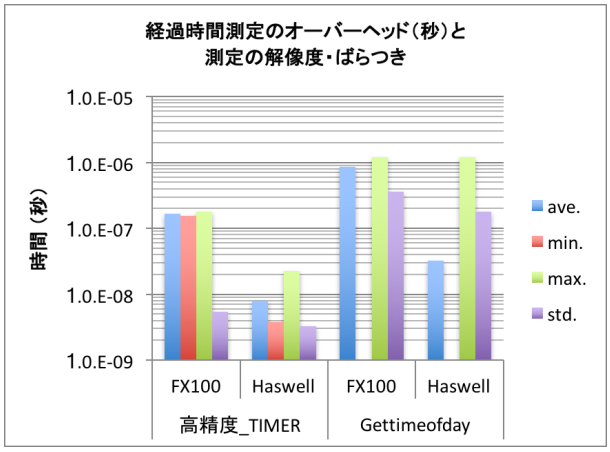

図 1 precise-timer

PMlib は一時的なベンチマークツールというよりは，アプリに常時組み込んでプロダクション利用することを想定したライブラリである．

3.4 PMlib の内部タイマー

PMlib は移殖が容易な汎用のパッケージであり，デフォルトでは Linux の標準タイマーである gettimeofday 関数を利用するが，HPC システムがオーバーヘッドの少ない高解像度タイマーを有している場合や，タイムスタンプカウンタを直接アクセスできる場合は，それを利用するバージョンをビルドできるパッケージングを目指している．前節で示した HPC システムでそのようなオプションを利用できる．図 1 は標準タイマーを使用した場合と高性能タイマーを使用した場合とのタイマー自身の測定解像度の比較である．

3.5 PMlib の出力情報

PMlib の出力として，標準出力や指定ファイルへのテキストレポート，および汎用のトレースフォーマットである OTF(Open Trace Format) ファイルを出力する機能を持っている．

3.6 関連する性能評価ツールと関連研究

HPC システム向けに様々な性能評価のためのツールが公開あるいは販売されている．

オープンソース性能評価ツール類としては

- Scalasca [2]：トレース生成，Score-P 共通インフラ
- Extrae [1]：トレース生成
- PAPI [5]：HWPC へのアクセス
- Linux perf tools：HWPC へのアクセス

ベンダーが提供する性能評価ツール類としては X86 系では

- Intel VTune [3], PGI Profiler [4]

また各 HPC システムは通常専用の性能評価ツールを装備している．

- FX100 システムでは富士通プロファイラなど

それぞれに特徴があり，ベンダー製品のツールはパッケージが統合化されて完成度が高い反面，利用できるシステムが制限され，相当の習熟期間が必要となる．オープンソースのツールは様々なシステムへの移植が可能であるが，各ツール毎に特徴・機能が明確で複数のツールを組み合わせて使う場合が多い．

特筆すべき点は，これら既存のツールはほぼ全てが評価の基準を，システム評価的観点での性能すなわち HWPC 測定性能に拠っていることである．

4. PMlib を用いた性能測定と分析例

基本的な演算について PMlib を用いた分析例を示す。

4.1 性能測定プラットフォーム

測定には以下の HPC システムを用いた．

- Intel Ivybridge CPU 搭載サーバ
- Intel Skylake CPU 搭載サーバ
- 富士通 prime HPC FX100

これらプラットフォームの主に CPU 部分についての性能諸元を表 1 に示す．

4.2 基本的な演算性能の評価

基本的な演算として，四則演算と平方根の配列計算カーネルを評価する．fortran ソースプログラムの抜粋を以下に示す．

```
1  subroutine sub_add(a,b,c,n)
2  real a(n), b(n), c(n)
3  do i=1,n
4  c(i)=a(i)+b(i)
5  end do
6  return
7  subroutine sub_fma(a,b,c,n)
8  real a(n), b(n), c(n)
9  do i=1,n
10 c(i)=a(i)+b(i)*d
11 end do
12 return
13 subroutine sub_divide(a,b,c,n)
14 real a(n), b(n), c(n)
15 do i=1,n
16 c(i)=b(i)/a(i)
17 end do
18 return
19 subroutine sub_sqrt(a,b,c,n)
20 real a(n), b(n), c(n)
21 do i=1,n
22 c(i)=sqrt(a(i))
23 end do
24 return
```

表 1 サーバの構成と最大性能（主に CPU 部分）

Symbol	FX100	SKY	IVY
Platform	FX100	Skylake	Ivybridge
CPU	SPARC64 XIfx	Gold 6148	E5-4620v2
core GHz	1.975	2.4	2.6
core Gflops	31.6	～30	
#core/cpu*	16	20	8
L1$ size(D,I)	64KB, 64KB	32KB, 32KB	
L1D$ BW GB/s	140/R+70/W	154/R + 77/W	
$ Linesize	256B	64B	64B
L2$ size	-	1MB	256KB
L2$ BW GB/s/c	-	154 (70)	
LL$ size	12MB	28MB	20 MB
LL$ BW GB/s/c	70/R+35/W	77 (43)	
Memory	HMC(8x16Ls)	DDR4-2666	DDR3-1600
Mem GB/s/cpu*	120/R+120/W	128	
#cores/cpu*	16	20	

注 cpu*は CPU または CMG を意図する

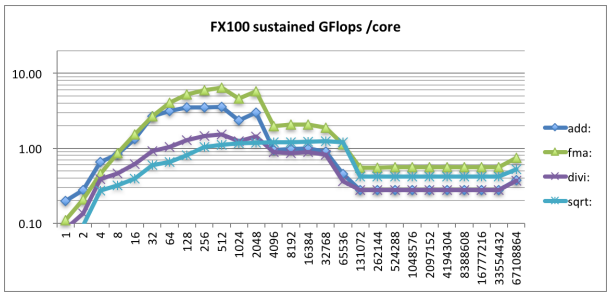

図 2 fx100-gflops-long-R8

どの計算もストライド 1 の依存性のないループで行われるため，効率の良いパイプライン化・SIMD 化・ベクトル化処理が適用されると期待されるものである．

上記ソースの各サブルーチンをメインプログラムから多数回呼び出して，その平均値を得てから，サブルーチン呼び出しオーバーヘッドの時間を差し引いて，計算に必要な時間を求める．do ループのオーバーヘッドは差し引かない．占有状態で負荷変動が少ない小規模構成の環境で測定を行う場合は，上記のような単純な方法で測定して問題ないが，利用負荷が高い共用サーバなどの上で同様の結果を得るためには，一般に OS，共有ファイルシステム，トポロジを共有するネットワーク，などの様々な要因から時間変動があり，粒度の小さな計算においてそのようなノイズを除去することは容易ではない．測定におけるノイズ成分を利用者側で除去する方法としては，上記の多数回呼び出しループの外側にさらに 10 回程度反復する最外側ループを設け，最外側 10 回測定値の内，標準偏差 1 σ の範囲だけを採用する方法などが効果的である．

この演算カーネルの測定をループ長 $n = 1, 2, 4, .., 2^{26}$ に対して FX100 で実行した結果を図 2 に示す．

表示しているのは数値計算上の性能である．n の増加に

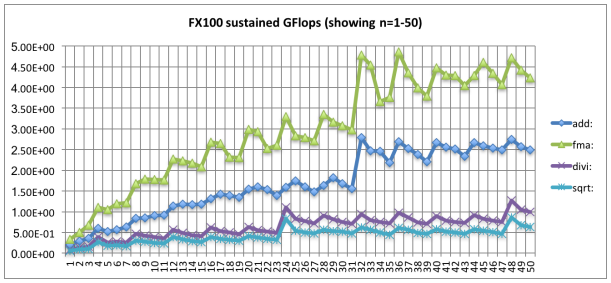

図 3 fx100-gflops-short-R8

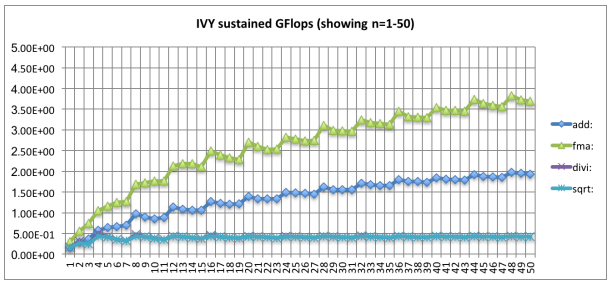

図 4 ivy-gflops-short-R8

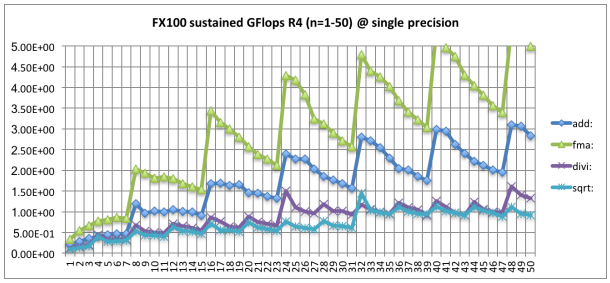

図 5 fx100-gflops-short-R4

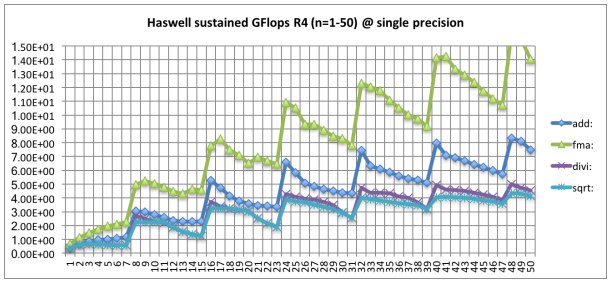

図 6 HAS-gflops-short-R4

連れてなだらかに性能が向上し，L1 キャッシュ容量に収まる範囲で最も高い性能に達し，その後 L2 キャッシュ容量の範囲内で，ついでメモリからのバンド幅で律速されるラインに落ち着くという全体傾向は，従来の多数の関連研究結果と重なるものであり，ループ長が長い部分においては roofline envelop の L1$ / L2$ / LLC / メモリ各階層のバンド幅で段階的に律速されている．ループ長が短い部分においてはループ処理自体のオーバーヘッドが演算量に比して重く，アクセスレイテンシで律速されている．[6]

同じカーネルに対してループ長が短い部分で n を連続的に変化させて測定した結果を FX100 について図 3 に，IVY について図 4 に各々示す．n を 1 から 128 の範囲で連続的に変化させて測定した結果と 1 から 50 の範囲をクローズアップした結果を示している．

Perf が単調に増加するのではなく，n が一定値の倍数において規則的に向上する変動がみられるが，これは連続データに対して SIMD 化命令を適用した計算でよくみられる状況である．この一定値は SIMD 幅（SIMD ビット数/データビット数）に対応するもので，例えば FX100 において倍精度（64 ビット）計算を行う場合は 256/64 = 4 であるため，計算性能は 4n,4n+1,4n+2,4n+3 と低下した後，4(n+1) で再び向上する．

図 3 は SIMD 機構をもつシステム上でのプログラミングにおいては，プログラムのループ長を該当アーキの SIMD 幅の倍数となるようなコーディングを意識することが，短いループ長の計算性能の向上においては特に重要であることを示している．

単精度（32 ビット）計算を行う場合はこの一定値は 256/32 = 8 となり，さらに顕著な性能影響効果がみられる．図 5 は FX100，図 6 は Haswell CPU での単精度計算の性能状況を示している．

PMlib を用いたプログラムの統計情報の出力および可視化によって、このような性能特性の把握が可能となり、計算の高速化への指針の一助が得られる。

5. 謝辞

本論文の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」で実施された内容に基づくものである．

参考文献

[1] Barcelona Supercomputing Center. Bsc performance tools. available from https://tools.bsc.es/.
[2] Jlich Supercomputing Centre. Scalasca. available from http://www.scalasca.org/.
[3] Intel Corporation. Intel vtune ® amplifier. https://software.intel.com/en-us/intel-vtune-amplifier-xe.
[4] NVIDIA Corporation. Pgi profiler. https://www.pgroup.com/products/.
[5] University of Tennessee. Papi, performance api. available from http://icl.cs.utk.edu/papi/software/index.html.
[6] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, Vol. 52, No. 4, pp. 65–76, April 2009.
[7] 小野謙二, 他. Pmlib 開発リポジトリ. available from http://avr-aics-riken.github.io/PMlib/.