Vol.2013-HPC-140 No.4
2013/7/31

システム評価のためのアプリケーション性能リポジトリの構築と性能モデルの評価

野村 哲弘[1,a)]　三浦 信一[1]　遠藤 敏夫[1]　松岡 聡[1]　鈴木 惣一朗[2]　丸山 直也[2]

概要：次世代スーパーコンピュータの開発に向けて実アプリケーション本位でのシステムデザインを行うためには，アーキテクチャと実アプリケーション実行性能を結びつける標準ベンチマークやパフォーマンスモデルが必要となるが，そのようなものはいまだ実現されていない．本報告では，将来 HPCI システムのあり方の調査研究「アプリケーション分野」におけるシステム評価用ベンチマークと性能モデルの作成について紹介する．本研究では，実アプリケーションコードの提供を受けてコードを単純化した「ミニアプリ」を作成し，ベンチマーク・性能モデルの基礎とする．また，複数の HPCI システム上で動作する性能評価ツールを用いてベンチマーク結果を収集し，性能モデルを構築するための基礎データとするとともに，性能モデルの構築方法・表現形式について考察する．

1. 背景

エクサスケールのスーパーコンピュータを実現するためには，電力をはじめとする多数のアーキテクチャ設計上の制約があることが示されている．[1], [2] そのような制約下において，今日スーパーコンピュータの性能比較の標準となった TOP500 ランキングで用いられている Linpack Benchmark でのスコアを最適化すべくスーパーコンピュータを構築することで，Linpack ベンチマークと実アプリケーションの実行性能が乖離しているという指摘がなされるようになってきた．そのため，アーキテクチャの提案者はそれぞれ，ターゲットとなるアプリケーションを定めてそのアプリケーションの実行性能を示すことで，自らのアーキテクチャの優位性を示そうとしている．一方，今日スーパーコンピュータで実行されているアプリケーションの分野および性能特性は多岐にわたり，すべてのアプリケーションの実行性能を Linpack のピーク性能値のような単一の指標に代表させて表現することは難しく，前述のアプリケーション性能による比較も，アーキテクチャ提案者が有利となるようなアプリケーションしか用いられていないためにアーキテクチャ間の比較が困難となる．このように，実アプリケーションの実行性能でアーキテクチャを比較しようとしても，共通する評価指標がないために意味のある比較がおこなえていないというのが現状である．

本研究では，そのような状況を打破すべく，アーキテクチャ間で共通して使えるベンチマークセットとしてミニアプリベンチマークを提案する．また，ベンチマーク時の測定項目を共通化したベンチマークスキーマを設計し，ベンチマーク結果を集約するベンチマーキングレポジトリの構築と，ベンチマークの性能モデルの作成，および実測結果との比較手法についても考察する．

2. ミニアプリベンチマークの構築

まず，今日のスーパーコンピュータで利用されている実アプリケーションをもとにしたベンチマークセットを構築する．アプリケーションの母集団としては，計算科学研究ロードマップ白書 [3] に記載されている分野の計算科学者に協力を仰ぎ，アプリケーションの収集を行った．表 **1** に一覧を示す．これらのアプリケーションからアルゴリズムの重複するものを除いたうえで，それぞれを以下の要素からなるミニアプリケーションベンチマークとして再構成する．

- 解説文書
  - アプリケーションの数学的背景
  - 実装されている機能
- 入力データセット
- 結果検証手法
- 参照実装

参照実装を作るために，我々はまずアプリケーションの

[1] 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
[2] 理化学研究所計算科学研究機構
RIKEN AICS
[a)] nomura.a.ac@m.titech.ac.jp

表 1　オリジナルアプリケーション一覧

名称	概要	特徴
feram	強誘電体 MD(長距離相互作用する擬スピン系)	OpenMP(MPI 使用はパラメータ並列版のみ), 3 次元 FFT
MARBLE	生体高分子 MD(クーロン力は PME)	MPI+OpenMP, 3 次元 FFT
para-TCCI	電子状態密度計算 (Hartree-Fock 計算)	MPI+OpenMP, 密行列対角化を逐次計算 (並列化予定なし)
FFVC	差分非圧縮熱流体 (直交等間隔格子)	MPI+OpenMP, SOR or GMRES
pSpatiocyte	細胞内シグナル伝搬計算 (反応拡散系)	MPI+OpenMP, 六方細密充填格子上のモンテカルロ, 反応過程は未実装
NEURON_K+	神経回路シミュレーション	MPI+OpenMP, Allgather を多用, 専用のモデル記述言語から C へのトランスレータ
GT5D	5 次元プラズマ乱流 (5 次元差分+2 次元 FEM)	MPI+OpenMP, 共役残差法, 1 次元 FFT
MODYLAS	汎用古典 MD(クーロン力は FMM)	MPI+OpenMP
STATE	第一原理 MD(密度汎関数法)	MPI+OpenMP(レプリカ並列, k 点並列, バンド or 平面波並列), FFT, 固有値計算 (RMM)
FrontFlow/blue	有限要素法非圧縮熱流体	MPI+自動並列 (京対応版), BiCGSTAB
SiGN-L1	遺伝子ネットワーク推定 (L1 正則化法)	MPI+OpenMP, ファイル出力が律速
NTChem/RI-MP2	電子相関計算 (分子軌道法)	MPI+OpenMP, DGEMM, 逐次計算部分が残っている, メモリ量 $O(N^3)$
OpenFMO	Hartree-Fock 法を基にしたフラグメント分子軌道 (FMO) 法による第一原理計算	MPI+OpenMP 動的負荷分散
CONQUEST	密度汎関数法による第一原理計算，$O(N)$ 法	MPI+OpenMP 疎行列 x 疎行列, FFT
NGS Analyzer	次世代シークエンサーの出力データ解析	
NICAM-DC	全球雲解像モデル NICAM の力学コア部 (有限体積法)	MPI 2〜4 段ルンゲクッタ

プロファイリングを通じて計算・メモリアクセス・ファイル I/O・通信のどれがボトルネックとなっているかを同定する．その上で，元アプリケーションのコードから入力データの範囲を絞ったうえで (例: シミュレーション対象の分子を水分子に固定する) アーキテクチャ特有の分岐・最適化を除き，“Don’t repeat yourself” などのコードのリファクタリングを行う．このようにして得られた実装は機能の制限はあるものの，アプリケーションとして一通り動作し，また元アプリケーションの実行時間に関する性質を保存しているものとなる．

なお，あくまでもこの実装は参照実装であるので，TOP500[4] における Linpack ベンチマーク [5] がそうであるように，特定のアーキテクチャへ向けて最適化を施してベンチマークを実行することを妨げるものではなく，むしろ OpenACC や CUDA などの計測対象アーキテクチャに適した最適化が行われることを期待している．実装の最適化作業時に，プログラムが科学的に正しい挙動を示しているかを検証するために，結果の検証手法をベンチマークに含める．これは計算順序の変化などによる誤差の影響があるためアプリケーションの出力が最適化前と同一であることとするのでは不十分であり，アプリケーションの知識がないユーザにも簡便に結果を検証できるものである必要がある．理想的には，結果の検証がユーザの目視を経ることなく自動的に行われる必要がある．

入力データセットはベンチマークとして実行するために十分な種類を提供する必要がある．特に実アプリケーションを元にしたベンチマークでは，従来のベンチマークのように入力が数次元のサイズを示すベクトル (例: 粒子数, X, Y, Z 各次元の領域分割数) で表せるものではなく，観測データなどの実データを要求することがある．理想的には任意のサイズに対応する入力データを生成するプログラムを作成することが考えられるが，そのようなことが不可能である場合にも，ベンチマークの実行結果からスケーラビリティを確認できるよう，たとえば同一の粒子分布で粒子数を変えたものなど，データ間で比較が行いやすいデータセットを提供する必要がある．

このように，実アプリケーションを縮退させてミニアプリケーションを作成する過程では，対象アプリケーションに対する深い知識が要求されるため，ミニアプリ化の過程は元アプリケーションの開発者との密な連携の下で，協力を得ながら行う必要がある．

最終的に，得られたミニアプリケーションについて性能モデルを作ることによって，そのミニアプリケーションが代表するアプリケーションの各アーキテクチャにおける性能を比較・推定することができるようになる．

現時点では，分子動力学ミニアプリケーションを 2 つと，量子色力学ミニアプリケーションを 1 つ整備済みであり，今後流体や気象などの他分野のアプリケーションもミニアプリ化していくことにより，ベンチマークセットとしての拡充を目指している．

3. ベンチマークスキーマの設計

ミニアプリケーションベンチマークから性能モデルを作り，その検証を行うためには，実行時間のような外形的な情報だけではなく，アプリケーションの各部分における演算数・メモリアクセス量・ファイルアクセス量・通信量を計測し，記録することが必要となる．また，そのように測定項目を定めるだけではなく，それらを実際に計測することが可能であることを示し，計測手法を公開することが必要となる．そのため，我々は以下示す指標について TSUBAME2.0 および京コンピュータの両方で Vampir[6], Scalasca[7], Tau[8] および PAPI[9] の各ツールを用いて情報を取得することができることを確認した．

- 時間
  - 実時間
  - CPU 時間
- 命令数・演算数
  - インストラクション数
  - 倍精度浮動小数点演算数
  - 単精度浮動小数点演算数
  - 倍精度浮動小数点演算命令数
  - 単精度浮動小数点演算命令数
  - 整数演算数
  - SIMD 命令数
  - 分岐命令数
  - 分岐命令予測ミス数
- メモリ・キャッシュアクセス
  - メモリロード命令数
  - メモリストア命令数
  - キャッシュアクセス命令数 (階層毎)
  - キャッシュミス数
  - キャッシュアクセスバイト数
  - キャッシュミスバイト数
- MPI・OpenMP
  - P2P 送信メッセージ数
  - P2P 送信バイト数
  - P2P 受信メッセージ数
  - P2P 受信バイト数
  - 集団通信実行数 (種類毎)
  - 集団通信バイト数
  - MPI コミュニケーショングラフ (メッセージ数)
  - MPI コミュニケーショングラフ (バイト数)
  - MPI オーバヘッド時間
  - OpenMP オーバヘッド時間
  - 同期待ち時間
- I/O
  - ファイル読込命令数
  - ファイル読込バイト数
  - ファイル書出命令数
  - ファイル書出バイト数

また，以下の項目は現時点ではアーキテクチャ非依存で取得する方法を確立していないが，性能モデル作成において将来的に必要となると考えられる．

- メモリ・キャッシュアクセス
  - メモリロード・ストアバイト数
  - メモリ局所性・アクセスパターン
  - ワーキングセットサイズ
  - NUMA におけるリモートメモリアクセス量
- 電力・実行環境
  - 消費電力
  - 気温
- ネットワーク
  - ホスト・スイッチポートの送受信データ量
  - ホスト・スイッチポートの送信待ち時間
  - ホスト・スイッチポートの輻輳率
  - ホスト・スイッチポートのエラー率
  - ルーティング情報
  - 集団通信を構成する 1 対 1 通信に関する情報
- ファイルシステム
  - リモートファイルシステムの総 I/O リクエスト数
  - リモートファイルシステムの送信バイト数
  - リモートファイルシステムの受信バイト数

これらの項目から，ベンチマーク時の計測必須項目の集合を定義し，それをベンチマークスキーマとする．本ベンチマークスキーマをベンチマークで取得する性能指標の最小セットとすることで，各種アーキテクチャ上でのベンチマーク結果を横断的に比較可能とすることができる．

4. ベンチマーキングリポジトリの設計

4.1 ベンチマークメタデータスキーマ

このようにして得られるベンチマーク結果を比較・検討し，性能モデルを確立するために，ベンチマーク結果を統一した形式で収集するためのリポジトリを作成する必要がある．また，ベンチマーク結果を収集するにあたり，ベンチマークの実行条件，つまりベンチマークのメタデータについても統一されたスキーマを定義する必要がある．そこで，我々はベンチマーク時に収集するデータを以下に示す階層に分け，それぞれを XML として表現し，データベースに格納することとした．ベンチマークスキーマの各要素の関連は図 **1** に示す通りである．

- Site: ハードウェア情報
- Environment: システムソフトウェア情報
- Applicaion: アプリケーション情報
- Scenario: ベンチマーク実行条件
- Result: ベンチマーク実行結果

Site は主にベンチマーク実行環境のハードウェア情報を

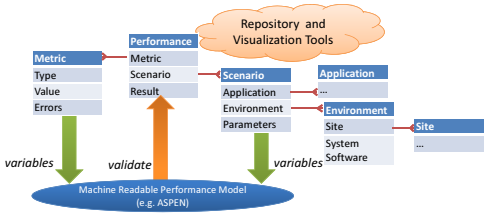

図 1　ベンチマーキングリポジトリスキーマの概要

格納する．図 2 にあるように，計算機の情報のうち，メンテナンスやソフトウェアアップデートでは変更されない部分のみを格納するため，計算機の供給者が 1 度書いたものをすべてのユーザで共有することができる．

```
<site>
  <name>TSUBAME 2.0 Thin</name>
  <nodes>
    <node>
      <processors>
        <processor>
          <name>Intel Xeon ⋯</name>
          <arch type="x86_64">
            <feature>SSE2</feature>
            ⋯
          </arch>
          <flops>⋯</flops>
          <numinstalled>2</numinstalled>
          ⋯
        </processor>
        <processor>⋯</processor>
      </processors>
      <memory>⋯</memory>
      <numinstalled>1408</numinstalled>
    </node>
  </nodes>
  <filesystems>⋯</filesystems>
  <interconnects>⋯</interconnects>
</site>
```

図 2　Site 情報の記述例

Environment は計算機の情報のうち，システムソフトウェアやライブラリなど，アプリケーションごとに共有されうる要素ではあるが，変更される可能性や，アプリケーションごとに個別のバージョンを用いたりする可能性があるものを格納する．このような情報は項目は事前に列挙しておくことができるが，実際のバージョンなどの情報はベンチマーク実行時になるまで不明であることも多いため，Site 情報作成時に，計算機に応じて図 3 に示すようなファイルを生成するためのスクリプトを作成することで生成できるようにする．

Application は，ベンチマーク対象のミニアプリケーションに関する情報を格納する．アプリケーションにおける入力の種類や，測定対象となるカーネルの範囲を示す情報を

```
<environment>
  <site>http://⋯</site>
  <date>2013/07/31 11:15:00+0900</date>
  <compilers>
    <compiler name="gcc-4">
      <type>gcc</type>
      <version>4.0</version>
      <path>/usr/apps/⋯</path>
    </compiler>
    <compiler name="icc">⋯</compiler>
  </compilers>
  <libraries>
    <library name="papi-5.0.0">
      ⋯
    </library>
  <libraries>
</environment>
```

図 3　Environment 情報の記述例

図 4 のように格納する．このファイルは，ミニアプリケーションと同時に提供され，ベンチマーク実施者はこのファイルを用いることで必要な範囲のベンチマークが自動的に実行できるようになることを意図している．

Scenario は，ベンチマークの実行シナリオを記述するために用いられる．ベンチマーク実施者は，実行環境に対応する Environment ファイルと，アプリケーションに対応する Application ファイルを指定し，適用できるライブラリ等が複数ある場合にはそれらを指定したうえで，どのような入力を与えるのかを図 5 に示すように記述する．ベンチマークの実行条件は Scenario ファイルが決まると一意に決まり，ベンチマーク実施後に第三者が実施条件を検証し比較実験を行うために有用となる．

Scenario に基づいて実施されたベンチマークの実行結果は，図 6 に示される Result ファイルに格納される．Result ファイルでは，Application で定義された全測定区間について，3 節に示す計測項目の値が格納されている．将来的には，Scenario ファイルを読みこんで，ベンチマークの実行スクリプトを自動生成し，そのスクリプトを実行することで必要な計測が行われ，結果を格納した Result ファイルが生成される仕組みを構築する予定である．

4.2　リポジトリのインタフェースと可視化

これらのスキーマで表されたベンチマークデータおよびベンチマークメタデータを格納するデータベースとしてベンチマーキングリポジトリを構築する．データベースへのデータの格納は CUI および web ベースの GUI ツールによって行われる予定である．格納した実行結果については，web ベースの可視化ツールを提供することにより以下に例示する比較が可能となる予定である．

- 入力の変化における実行時間の変化

```
<application>
  <name>…</name>
  <version>…</version>
  <parameters>
    <parameter>
      <name>numprocs</name>
      <type>integer</type>
    </parameter>
    <parameter>…
      <type>file</type>
    </parameter>
  </parameters>
  <configure>
    <option>--enable-XXX</option>
  </configure>
  <libraries>
    <require>mpi-c</require>
    <require>lapack
      <version-gt>XX.X</version>
    </require>
  </libraries>
  <execute>
    <option>-f <subst param="inputfile" /></option>
  </execute>
  <regions>
    <region>
      <start>….f90:xxxx</start>
      <end>…</end>
    </region>
    <region>
      <procname>matmul</procname>
    </region>
  </regions>
</application>
```

図 4　Application 情報の記述例

- 同一マシンにおける Arithmetic Intencity と演算性能の関係 (いわゆる Roofline モデル)
- 同一アプリケーションにおける複数マシンでの演算性能の比較

これらの可視化を通じて，性能モデルの作成や，アーキテクチャ間でのアプリケーション性能比較を容易に行えるようにする．

5. 性能モデルの構築と評価方法

ミニアプリベンチマークにおいては，性能モデルを構築することが重要となる．また，アーキテクチャにおける性能の差異を自動的に算出するためには，性能モデル式が機械可読であり，性能値の算出が自動的に行えるものである必要がある．この性質はモデルと実測値との比較を自動化するうえでも必要である．

性能モデルの構築手法としては以下の 2 通りが考えられる: i) 知識ベースモデリング ii) 解析的モデリング．知識ベースのモデリングでは，アプリケーションのソースコードから動作に本質的な部分について，それぞれに必要な演算数やメモリアクセス数をボトムアップに積み上げていくことでアプリケーションの性能モデルを構築する手法である．この方法では，モデルに出現する各変数や部分式の意味が明確となり，ボトルネックとなる箇所が分かりやすいものとなる利点があるが，その一方でモデルの構築にはアプリケーションに対する深い理解が必要であり，モデルを構築する際に着目しなかった要素の影響をモデル内に組み込むことができないため，モデル式と実測値との乖離が起こりうるという欠点がある．他方，後者の方法ではアプリ

```
<scenario>
  <environment>http://…</environment>
  <application>http://…</application>
  <compiler>
    <type>gcc</type>
    <version>4.0</version>
  </compiler>
  <library>
    <type>…</type>
    <version>…</version>
  </library>
  <option>
    <compilerflag>-DNODEBUG</>
    …
  </option>
  <parameter>
    <name>…</name>
    <value>…</value>
  </parameter>
  <parameter>…</parameter>
</scenario>
```

図 5　Scenario 情報の記述例

```
<result>
  <scenario>http://…</scenario>
  <date>2013/07/31 11:15:00+0900</date>
  <range name="fftkernel">
    <metric>
      <type>flop</type>
      <value>2503413415</value>
    </meric>
    <metric>
      <type>time</type>
      <value>3.141592653</value>
    </metric>
    …
  </range>
  <range name="…">
    …
  </range>
<result>
```

図 6　Result 情報の出力例

ケーションに関する知識は用いずに，アプリケーションの性能値をマシンの性能やその他計測した演算数等の指標を説明変数の候補として数理最適化の手法を用いてモデル式を作成する手法である．[10] この方式で作られたモデル式は，実測結果とよく合う結果をもたらすものの，場合によっては実際に性能とは相関のない変数を説明変数として採用するなどの over fitting の危険があり，最終的にアプリケーションに必要な性能指標が何であるかを決定できないモデル式しか得られない危険性がある．

ベンチマーク結果と性能モデルを比較するうえでは，これらの両方式の利点と欠点のバランスを考えたうえで，両手法およびその組み合わせにおいて性能モデルを表現できる形式を用いる必要がある．本研究では，性能モデルの表現方法としては，Aspen[11] をベンチマーキングリポジトリの可視化ツールに組み込んで利用する予定である．Aspen は主に上記区分の i) にあたる知識ベースのモデリングを目的とした性能モデル記述言語であり，性能モデルの中に `parameter` としてマシンの性能値や FLOPS 値などを組み込み，それらの値をもとにアプリケーションの処理の流れを記述することで性能モデル式を書き起こし，性能モデルをグラフの形で可視化することができる．この Apsen における `parameter` の値としてベンチマーキングリポジトリに格納されたデータを与えることで，Aspen で記述されたモデルにおける予測性能値を得ることができ，実測値と比較することができる．また，逆に Aspen の可視化機構を組み込んで拡張することにより，Aspen によって可視化された性能モデルによる予測値の曲線上に実測値のプロットを重ね，モデルの実測値に対する一致度合いを確認することができるようになる．

6. まとめ

本報告では，次世代スーパーコンピュータのアーキテクチャ評価のためのミニアプリベンチマークとベンチマーキングリポジトリおよび，性能モデルの構築方法について紹介した．複数のアーキテクチャを同一の実アプリケーションで横断的に評価するためにミニアプリベンチマークの整備が必要であり，実アプリケーションをもとにしたミニアプリベンチマークの整備をすすめているところである．ベンチマーキングリポジトリと性能モデルの可視化については，現段階ではいまだデザインの域を出ていないが，今後整備したミニアプリケーションを対象にベンチマークを取得し，結果を整理し性能モデルを構築していく過程においてデザインの妥当性を検証し，実装の細部を改善していく予定である．

謝辞　本研究は，文部科学省「将来の HPCI システムのあり方の調査研究」(平成 24 年度〜平成 25 年度) の支援による．また，本研究の結果の一部は，理化学研究所のスーパーコンピュータ「京」を利用するとともに，「京」以外の HPCI システム利用研究課題を遂行して得られたものである．(課題番号: hp120261)

参考文献

[1] Dongarra, J., Beckman, P. et al.: The International Exascale Software Roadmap, *International Journal of High Performance Computer Applications*, Vol. 25, No. 1 (2011).
[2] 石川　裕，丸山直也ほか：今後の HPCI 技術開発に関する報告書: HPCI 技術ロードマップ白書 (2012).
[3] 牧野淳一郎，富田浩文ほか：今後の HPCI 技術開発に関する報告書: 計算科学研究ロードマップ白書 (2012).
[4] TOP500.org: TOP500 Supercomputer Sites, `http://www.top500.org/`.
[5] Petitet, A., Whaley, R. C., Dongarra, J. and Cleary, A.: HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, `http://www.netlib.org/benchmark/hpl/`.
[6] Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S. and Nagel, W. E.: The Vampir Performance Analysis Tool-Set, *Parallel Tools Workshop* (Resch, M. M., Keller, R., Himmler, V., Krammer, B. and Schulz, A., eds.), Springer, pp. 139–155 (2008).
[7] Geimer, M., Wolf, F., Wylie, B. J. N., Ábrahám, E., Becker, D. and Mohr, B.: The Scalasca performance toolset architecture, *Concurrency and Computation: Practice and Experience*, Vol. 22, No. 6, pp. 702–719 (online), DOI: 10.1002/cpe.1556 (2010).
[8] Shende, S. S. and Malony, A. D.: The Tau Parallel Performance System, *The International Journal of High Performance Computing Applications*, Vol. 20, pp. 287–331 (2006).
[9] Mucci, P. J., Browne, S., Deane, C. and Ho, G.: PAPI: A Portable Interface to Hardware Performance Counters (1999).
[10] Bauer, G., Gottlieb, S. and Hoefler, T.: Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3 rmd, *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, IEEE Computer Society, pp. 652–659 (2012).
[11] Spafford, K. and Vetter, J. S.: Aspen: A Domain Specific Language for Performance Modeling, *SC12: ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis*, (online), DOI: http://dx.doi.org/10.1109/SC.2012.20 (2012).