システム評価法確立
スキーマとベンチマーク・プラットフォーム整備の現状

2012/11/20

野村 @ 東工大

アプリベンチマークの目的

- アプリの性能モデルを立てたときに、実際にそれが成り立っていることを確認する
  - 乖離している場合、どういう離れ方をしているか
    - モデルが不十分で考慮していないパラメータがある
    - 何らかの性能劣化要因がある
- 例: SCALE3ミニアプリのカーネル部分の性能モデル
  - FLOPS = システムのメモリバンド幅 / 実効B/F

目標関数　　カタログスペック　　実測 → How?

  - 例えばScalasca+PAPIでメモリアクセス(最外キャッシュミス)とFP演算を計測すれば測れる

ベンチマークメタデータの スキーマとは

- アプリの性能モデルに出現しうる変数の集合
  - 複数のアプリで必要な変数を集め、それらの被覆集合を取れば、他の大抵のアプリでも性能モデルを立てられる
  - そのような集合があれば、性能モデル未知のアプリが来たとしても、これらのメタデータを計測しておき、そのあとでモデルを考えることができるようになる
  - 測り方をドキュメント化し、誰でも測れるようにする

スキーマの探索空間

- とはいえ、現実的に既存のツール群で『測れる』ものでなければ役に立たない
- メタデータは、アプリケーションの任意のイベント間のカウンタの変化量として捉えられる
  - 例: アプリ実行時間
    - アプリ開始から終了までのWallclockの差分
  - 例: 通信バンド幅
    - メッセージサイズ / 通信開始から終了までのClock差分
- 測れるイベントと測れるカウンタを使う必要性
  - しかし全部は多すぎて測れないので絞る
- 逆にスキーマに出てくる要素は測ることが出来なければならない

利用できるProfiler/Tracer

- Scalasca
  - プロファイラとしての利用がメイン
    - 逐次的な情報ではなく、関数毎等の統計情報
  - 利用は比較的容易
  - CUDAやpthreadなどへの対応が不足している
- Vampir・VampirTrace
  - トレーサとしての利用がメイン
    - 逐次的なカウンタの変化を追うことに長ける
    - ログファイルはプロファイラより大きくなる
  - 可視化部分(Vampir)は商用製品
- Tau
  - トレーサとしての利用がメイン
  - 機能ごとに分化したツール群
  - 移植・利用は比較的難しい
    - Proof of Concept集のようなもので、適用環境ごとに細かな修正を要する
- Score-P
  - 上記Profiler/Tracerの機能・出力を統一化するプロジェクト
  - 2013年6月にCUDA・pthread他必要な機能が出揃う見込み
- いずれも英語情報は充実しているが、日本語情報は不足気味

計測できる/すべきイベント(1/2)

- 関数の開始・終了 (単にタイミング情報)
  - Scalasca, Tau VampireTrace(以下Tracer)でとれる
- I/O命令 (書き込み先ファイルやサイズなど)
  - Tracerで最低限はとれる
  - MPI-IOなど、特定のAPIを利用していればScalascaなどで詳細情報を取れる
  - fuseで命令フックしても取ることも考えられるが、自動化するツールはない?
    - 分野1ゲノム 分野3時間発展系 では欲しいのだけど…

計測できる/すべきイベント(2/2)

- 通信 (通信相手やメッセージサイズなど)
  - Tracer, MPE(MPI通信)
  - 通信に関するループがどれか、どういう通信をしてるか
- メモリアクセス (パターン)
  - Tracerでは無理なので、Valgrindなどを使う必要あり
    - プログラムの中のループを関数単位よりも細かくマークアップして、Hotspotがどれか判断し、カーネルのリスト(場所とOccupancy)を作る
      - Fujitsuのプロファイラでは出力可能
      - 一般のツールでできるか
      - どうやって記録するか(標準のAnnotation規則)
    - カーネル部分のメモリのアクセスパターンなどを拾う

計測できる/すべきカウンタ(1/2)

- CPU・メモリ関連
  - PAPI
    - papi_availやpapi_native_availで一覧
    - 前ページのTracerはすべて対応
    - ところでMICはPAPI使えるの?
- GPU関連
  - CUDA Profiler
    - TauやVampireTraceは対応を謳っている
- 通信
  - IBのカウンタ(port, switch)
    - Ibdiagnetなどのツールで根こそぎ収集可能
    - Tofuはどうやってとるの? 少しはとれるらしい
  - MPIなどの引数 (送受信量の積算etc)

計測できる/すべきカウンタ(2/2)

- I/O
  - LMT(Lustre Monitoring Tool)
  - iostat
  - GPFSは?
- 電力
  - IPMI, iLO, 分電盤の情報
    - 完全にベンダ依存で標準的なインタフェースはない
    - 京は電力情報を取れるか?
    - Interfaceを切って、高精度にとれる環境を作る必要がある
    - 既存マシンでも取れるだけのデータを取る
- 時刻
  - CPU Time・Wallclock

例: 実効B/F値の計測

- 特定の関数の実行中の実効B/F値を得たい
  - 実効 = キャッシュに当たった分は除く
- PAPIの以下のカウンタの値から求められる
  - PAPI_L3_TCM L3キャッシュミスの数
    - /proc/cpuinfo cache_alignmentでバイト数に変換
  - PAPI_DP_OPS 倍精度浮動小数点数演算数
- scan –m PAPI_L3_TCM:PAPI_DP_OPS mpirun …
  - 実効B/F = L3_TCM * cache_alignment / DP_OPS
- (Demo)

SCALEミニアプリの性能モデル: フェイズ毎のパターン解析

9/19 第2回全体会議より

- RK(ルンゲクッタ)フェイズ
  - ステンシルの3重ループ27個
  - 演算回数、メモリアクセス回数の調査
    - 人手で数え上げ
    - 自動化ツール?
  - メモリアクセス
    - プログラムの字面上のアクセス != 実際のアクセス
    - 実効メモリアクセス = 理想的なLLCを仮定したアクセス
      - 一度読んだデータは必ずキャッシュヒット
      - ≈最適化されたメモリアクセス
  - 見た目のB/F: 4.8
  - 実効B/F: 1.9
  - 明らかにメモリボトルネック

性能モデル ➔ システムのメモリバンド幅 / 実効B/F

まとめ

- ベンチマークのスキーマ
  - 性能モデルに出現する変数リスト
    - 我々はその集合を知らない → 知る必要がある
      - 測れるもの ⊇ スキーマ ⊇ 各アプリの性能モデル変数
    - 測り方をドキュメント化し、誰でも計測できるようにする
      - ベンチマークプラットフォーム
      - 誰がやっても同じ結果が出せるように
  - カーネルとなるループの情報
  - TODO
    - 測れるものの調査
    - 各アプリの性能モデルに出現する変数の調査
    - 具体的な計測手法のドキュメント化