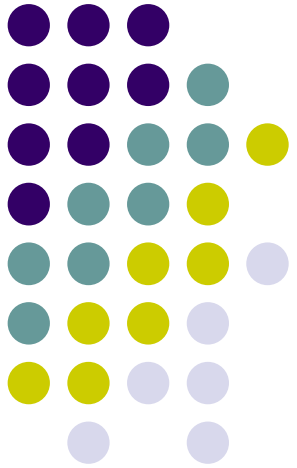
2014年度
実践的並列コンピューティング
第6回



OpenMPによる
共有メモリプログラミング (4)

遠藤 敏夫

endo@is.titech.ac.jp

プログラムはソフトウェア部品から成り立っている

- 一から十まで、自分で作ったソフトウェアということはほぼない

  例:

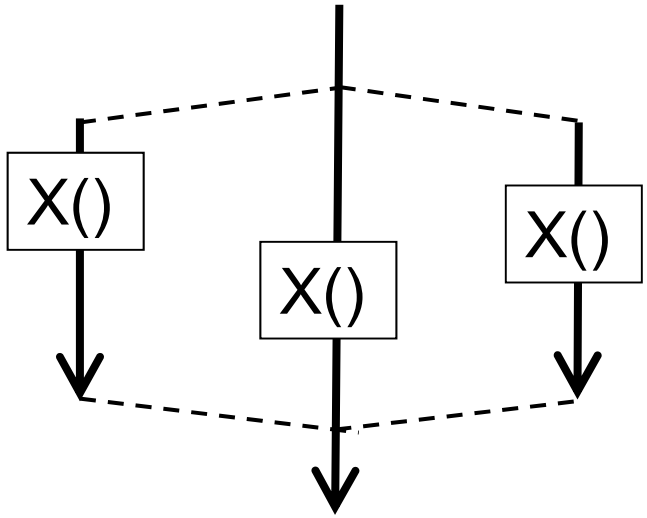
  - 標準ライブラリ
  - オープンソースのライブラリ
  - (フレームワーク)

  このような場合でもマルチスレッドでokなのか？

  マルチコアは広まる前に設計されたライブラリでは問題発生することも

Thread safe

- 複数スレッドが、排他制御なしに関数X()を呼び出しても、正常動作するとき、「関数Xはスレッドセーフ(thread safe)である」と呼ぶ
  - Multithread safe, MT-safeも同じ



標準ライブラリ関数はthread safeか？

- スレッドセーフなもの
  - printf, malloc, gethostname…
    - mallocは、共有ヒープに対して内部で排他制御をしていると予想される
  - rand_r, strtok_r, gethostbyname_r…
- スレッドセーフでないもの
  - rand, strtok, gethostbyname…

randがthread safeでない理由

- rand(): 擬似乱数を返す
  - あるseedによりsrand(seed)で初期化された後、randが繰り返し呼ばれると、同じ乱数列を返す

```
srand(123);
A = rand(); → 128959393
A = rand(); → 1692901013
A = rand(); → 436085873
A = rand(); → 748533630
   :
```

```
srand(456);
A = rand(); → 1929505231
A = rand(); → 1800653403
A = rand(); → 1030306120
A = rand(); → 2023122793
   :
```

```
srand(123);
A = rand(); → 128959393
A = rand(); → 1692901013
A = rand(); → 436085873
A = rand(); → 748533630
   :
```

seedが同じなら、同じ乱数列

randがthread safeでない理由

- 予想されるsrand, randの実装

```
int g_seed;

void srand(int seed)
{   g_seed = seed;  }

int rand()
{
    int res = XXX(g_seed);
    g_seed = YYY(g_seed);
    return res;
}
```

共有変数を使っている点が、thread safeでなくしている！

“man rand” より

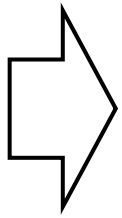
The function rand( ) is not reentrant or thread-safe, since it uses hidden state that is modified on each call.

rand単体の中で排他制御してもダメ！
守りたいのは複数のrand()からなる列なので

randのthread safe版: rand_r

- int rand_r(int *seedp)
  - seedは呼び出し側で管理すること
  - マルチスレッドプログラムからrandを使いたいなら、代わりにrand_rを使って書き換えること

```
srand(123);
A = rand();
A = rand();
A = rand();
A = rand();
    :
```



```
int seed = 123;
A = rand_r(&seed);
A = rand_r(&seed);
A = rand_r(&seed);
A = rand_r(&seed);
    :
```

※サンプルプログラムpi_ompもrand_rを使っている

マルチスレッド版rand_r

- 予想されるrand_rの実装

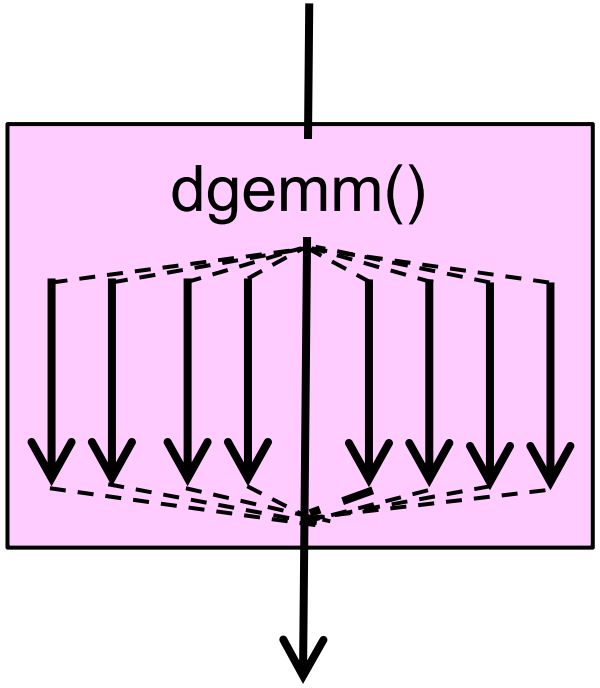
```
int rand_r(int seedp)
{
    int res = XXX(*seedp);
    *seedp = YYY(*seedp);
    return res;
}
```

共有変数なし。
状態(今のseed)をためて
おくのはユーザの責任とする

ライブラリ関数を「使う際」も「作る際」も、
Thread-safeか否かに
気をつけるべき

別のケース:ライブラリ関数内部で並列化されている場合

- Intel MKL, AMD ACML, GotoBLASなど
  - 行列演算などをとても速く実行するライブラリ



ユーザプログラムは並列化を意識しなくても、勝手にマルチコアを利用して速くなってくれる

複数スレッドがdgemmを呼んだらどうなるか？は実装・バージョン次第・・・

講演会のお知らせ

講演題目：Multiscale Flow Simulations: Particle, Grids and Supercomputers

講演者：　Petros Koumoutsakos, ETH Zurich, Switzerland

日時：　平成26年5月15日（木）13:30

場所：　東京工業大学・学術国際情報センター　情報棟2F 会議室

（大岡山キャンパス正門から 100m のイチョウ並木沿いの建物）

「Petros Koumoutsakos 教授のグループは、昨年、1.3兆格子を用いた圧縮性の二相流計算に対して、Sequoia で14.5 PFLOPS の実行性能を達成し、ゴードンベル賞を受賞しています。」

本授業のレポートについて

- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

  予定パート:

  - OpenMPパート
  - MPIパート
  - GPUパート

OpenMPパート課題説明 (1)

以下の[O1]—[O2]のどれか一つについてレポートを提出してください

[O1] diffusionサンプルプログラムを、OpenMPで並列化してください.

オプション:

- 配列サイズや時間ステップ数を可変パラメータにしてみる。引数で受け取って、配列をmallocで確保するようにする、など。
- より良いアルゴリズムにしてみる。ブロック化・計算順序変更でキャッシュミスを減らせないか？

OpenMPパート課題説明 (2)

[O2] 自由課題：任意のプログラムを，OpenMPを用いて並列化してください．

- 単純な並列化で済む問題ではないことが望ましい
  - スレッド・プロセス間に依存関係がある
  - 均等分割ではうまくいかない、など
- たとえば，過去のSuperConの本選問題
  http://www.gsic.titech.ac.jp/supercon/
  たんぱく質類似度(2003)，N体問題(2001)・・・
  入力データは自分で作る必要あり
- たとえば，自分が研究している問題

課題の注意

- いずれの課題の場合も，レポートに以下を含むこと
  - 計算・データの割り当て手法の説明
  - TSUBAME2などで実行したときの性能
    - プロセッサ(コア)数を様々に変化させたとき．大規模のほうがよい．XXコア以上で発生する問題に触れているとなお良い
    - 問題サイズを様々に変化させたとき(可能な問題なら)
  - 高性能化のための工夫が含まれているとなお良い
    - 「XXXのためにXXXをしてみたが高速にならなかった」のような失敗でも可
  - 作成したプログラムについても、zipなどで圧縮して添付
    - 困難な場合，TSUBAME2の自分のホームディレクトリに置き，置き場所を連絡

課題の提出について

- OpenMPパート提出期限
  - ~~5/26 (月)~~ ⇒ 6/2（月）23:50に変更
- OCW-i ウェブページから下記ファイルを提出のこと
- レポート形式
  - 本文：PDF, Word, テキストファイルのいずれか
  - プログラム：zip形式に圧縮するのがのぞましい

- OCW-iからの提出が困難な場合、メールでもok
  - 送り先： ppcomp@el.gsic.titech.ac.jp
  - メール題名： ppcomp report

次回: 5/19(月)

- ソフトウェア性能のためのコンピュータアーキテクチャ

  - スケジュールについてはOCW pageも参照
    - http://www.el.gsic.titech.ac.jp/~endo/

    → 2014年度前期情報(OCW) → 講義ノート