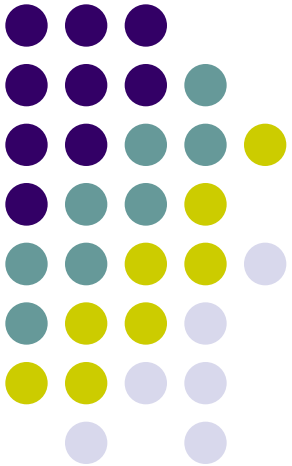
2013年度
実践的並列コンピューティング
第4回



OpenMPによる
共有メモリプログラミング (2)

遠藤 敏夫

endo@is.titech.ac.jp

2013年5月9日

前回の復習

- 共有メモリ並列プログラミングモデルの一つ
- プログラム中に
  - #pragma omp parallel

  と書くと、その次のブロック・文が並列に実行される (並列region)
- 並列region中に
  - #pragma omp for

  と書くと、その次のfor文は、スレッド間で分担して実行される

「正しい」並列プログラムに向けて

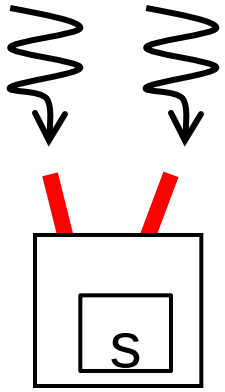
- OpenMPは指定個所を「並列に実行」してくれるが、「正しさの保証」はしてくれない ⇒プログラマの責任

sが共有変数のとき、s = s+1 と s = s+1 を並列に実行するとどうなるか？

このプログラムは正しいか？

```
   int s = 0;
#pragma omp parallel
   {
       s = s+1;
   }
```

ここがs++でも話は同じ

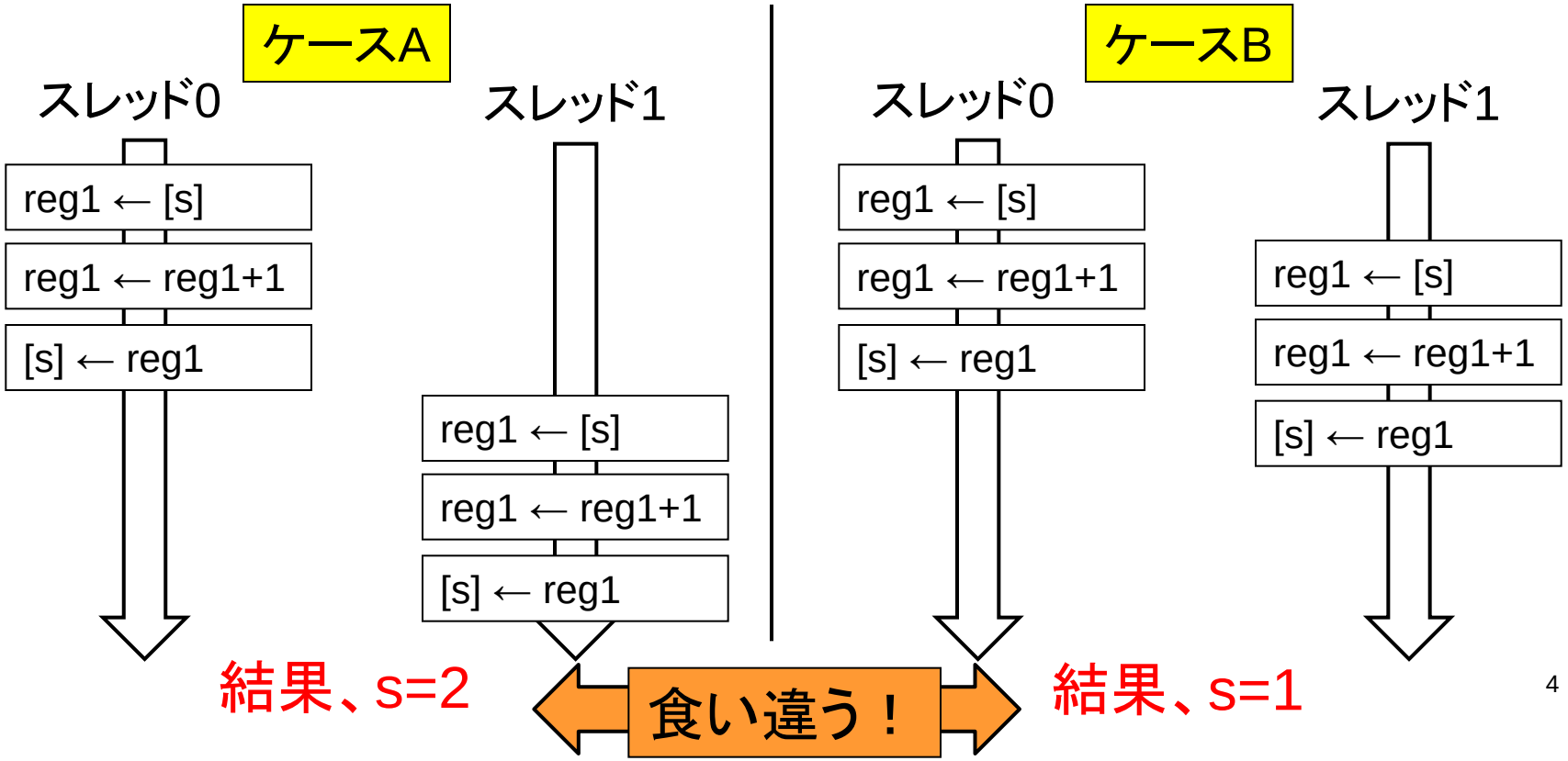


2スレッドがそれぞれsを1増やすので、結果はs=2 ??

正しい並列プログラムに向けて: Race condition

Race condition (競合状態)：処理結果が、イベントの順序やタイミングにより、予期しないことになること

- 共有メモリの場合、複数スレッドが共有変数に(調停せずに)アクセスすると起こる。s=s+1の例では：



正しい並列プログラムに向けて: Mutual exclusion

Mutual exclusion(排他制御):

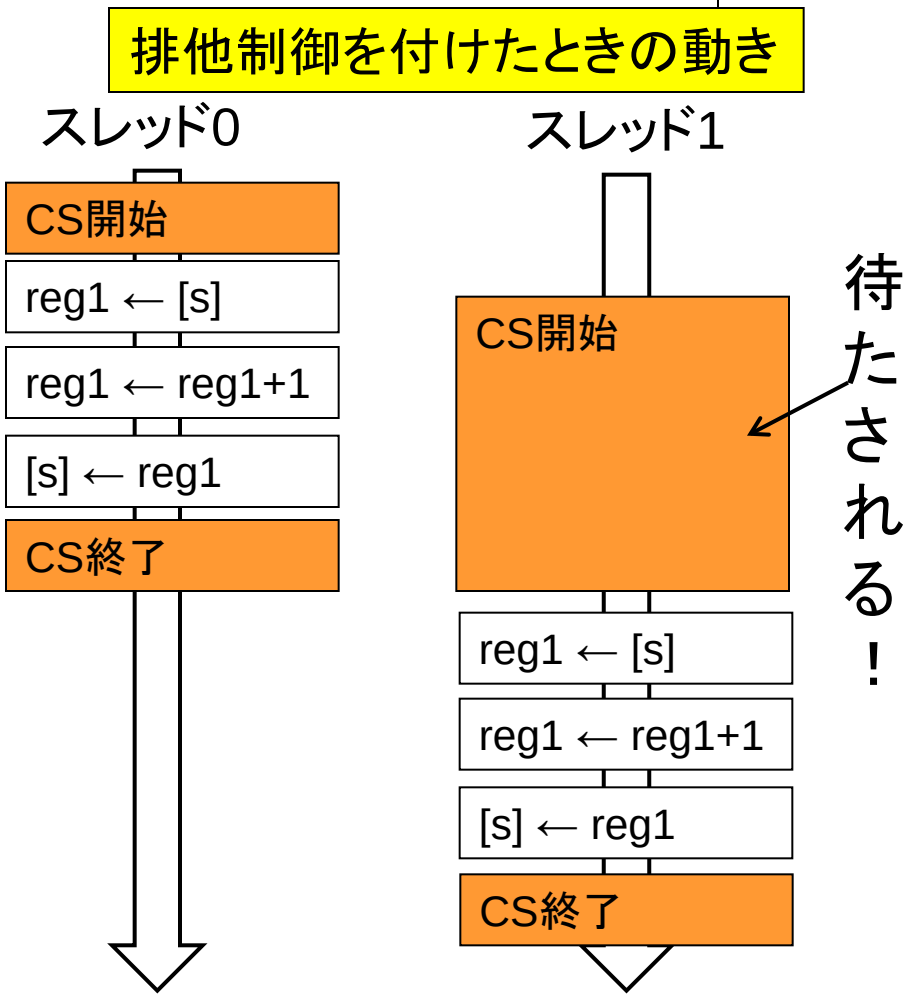
あるコード部分に、同時に1スレッドしか入れないようにする

- 1スレッドしか入れない部分: critical section

```
    int s = 0;
#pragma omp parallel
    {
#pragma omp critical
        s = s+1;
    }
```

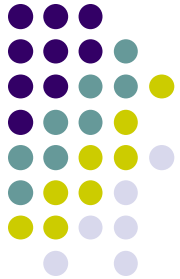
... criticalの直後の文・ブロックはクリティカルセクションになる

⇒ この結果、競合状態はなくなり、必ずs=スレッド数となる



必ず、s=2

Critical 指示文について補足情報



- プログラム文中で複数criticalセクションがあるときの挙動は？

⇒ どれかのcriticalセクションに入れるのは，同時に1スレッドのみ

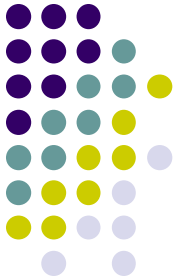
補足：

#pramga omp critical (name)

という文法があり，「同じnameを持ったcriticalセクションに入れるのは，同時に1スレッドのみ」

デフォルトでは，「名無し」という名前を持つとみなされる

正しいプログラムでも速いとは限らない



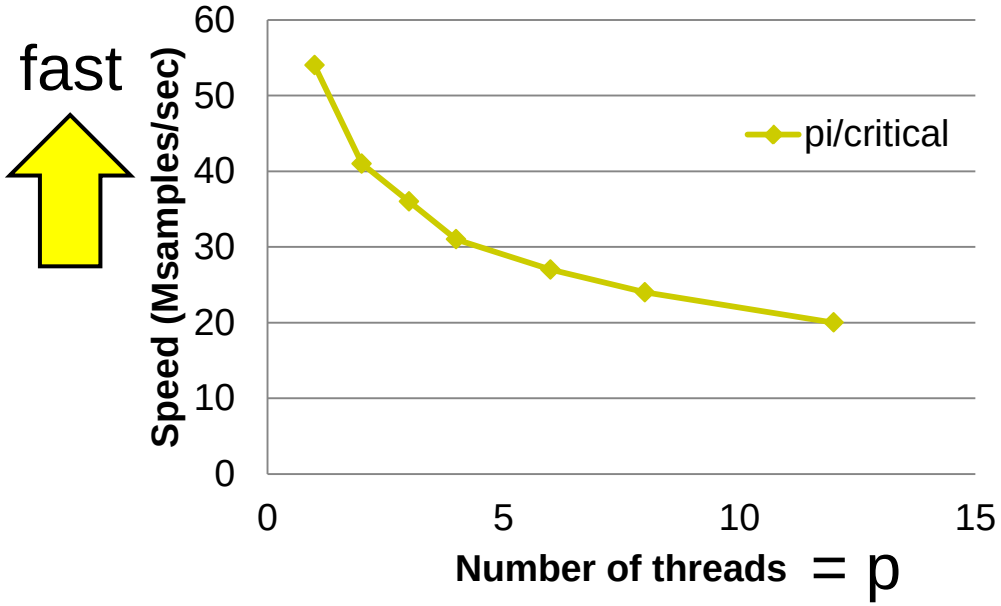
- 排他制御は、スレッドが「何もできずに待つ」状況を生み出す ⇒ ボトルネックとなり、性能向上の妨げに
- アムダールの法則(Amdahl's law)
  - プログラムのうち、
    - 並列実行できる部分の割合をα
    - 逐次にしか実行できない部分の割合を1-α

  ⇒　並列度pでの速度向上率は

  1 / (1 - α + α / p)　である
  - クリティカルセクションだらけのプログラムは α≒0 なので、どんなにpを増やしても速くならない

事態はアムダールの法則より さらに悪い

- piサンプル(omp版)を、reductionの代わりにcriticalを使った場合の性能
  - TSUBAME2.0の1ノードで実行, PGIコンパイラ12.8

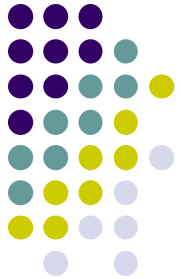


アムダールの法則どおりなら、どんなにαが悪くても(0に近くても)、pに対して単調増加のはず

⇒実際には遅くなってしまっている！

※ この理由の説明のためには、キャッシュコヒーレンシなどの理解が必要

Criticalとreductionの関係って？

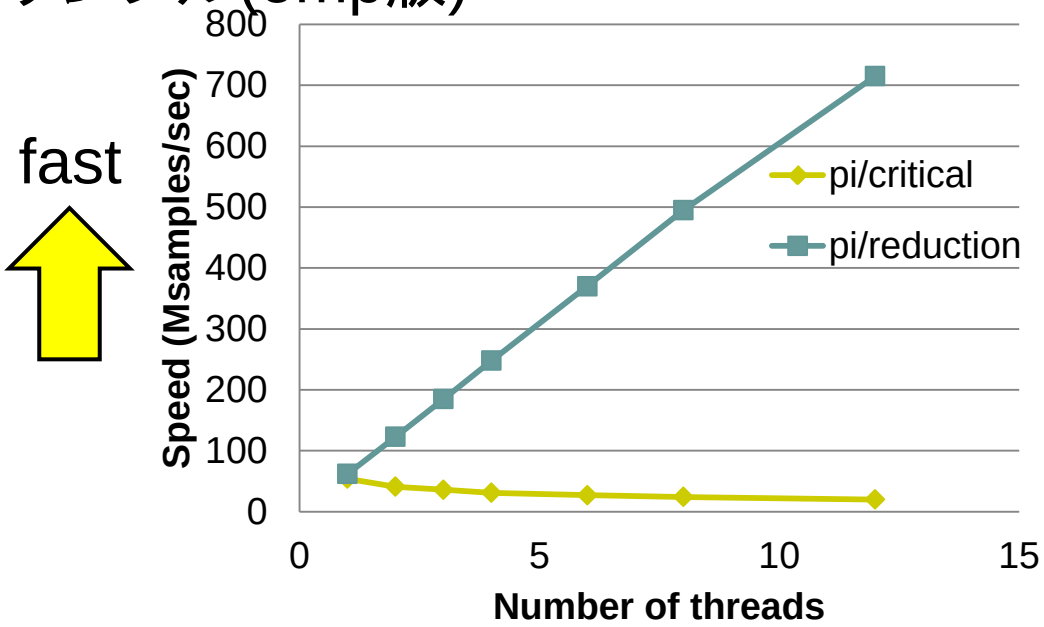


どちらも競合状態を防ぐことができる

- Critical:
  - 一般的な排他制御であり、クリティカルセクションの中にはどんな計算でも記述できる
    - 例: Wikipedia「排他制御」にリスト操作の例
  - Reductionで記述可能なことは、criticalでも可能・ただし遅い
- Reduction:
  - #pragma omp forでのみ可能
  - 可能な計算は、共有変数への加算・積算・AND・ORの繰り返しのみ (reduction処理)
  - はまれば速い

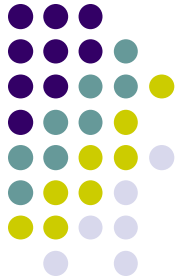
CriticalとReductionの性能比較

- piサンプル(omp版)



- どちらも意味は同じで、正しいプログラム
- Reductionのほうがはるかに速く、12スレッドどうしだと35倍もの差
- pを増やすと性能向上することを、スケーラブルであるという

For指示文の補足情報: #pragma omp forが書ける条件



- 直後のfor文が「canonical form (正準形)」であること

```
#pragma omp for
  for (var = lb; var rel-op b; incr-expr)
    body
```

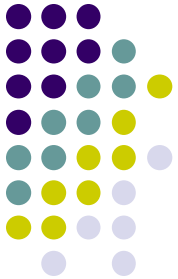
ここでincr-exprは ++var, --var, var++, var--, var+=c, var-=cなど

for (i = 0; i < n; i++) ⇒ For指示文可能！

for (p = head; p != NULL; p = p->next) ⇒ For指示文不可

Canonical formであっても、プログラムの挙動の正しさは
やはりプログラマの責任

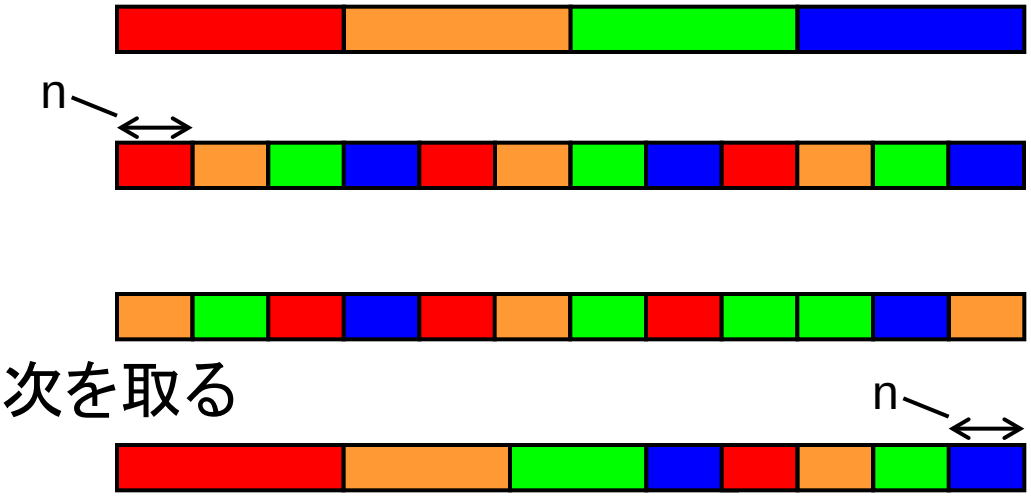
For指示文のオプション：スケジューリング



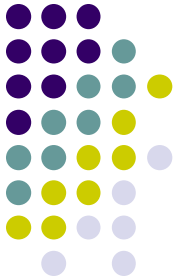
- 通常，スレッドに割り当てられる反復回数は均等分割
  - 例： 1000回を4スレッドでなら、250ずつ
- 各反復の仕事量が違うと非効率．たとえば三角行列の演算 ⇒ 様々なスケジューリング手法が用意されている

  #pragma omp for schedule(・・・)

schedule(static)
　均等分割(default)
schedule(static, n)
　ブロックサイクリック分割
schedule(dynamic, n)
　仕事が終わったスレッドが次を取る
schedule(guided, n)
　“chunk size”が等比的に小さく



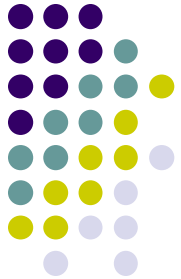
参考：OpenMPとpthreadの比較



- OpenMPでは
  - 原則的にスレッド数は一定
- Pthreadでは
  - 好きなときにスレッドの生成・終了
  - スレッド内の局所変数はスレッドプライベート

	OpenMP	pthread
クリティカルセクション	あり	あり(pthread_mutex_lockなど)
バリア同期	あり	なし
条件変数による同期 (パイプライン並列など)	なし	あり(pthread_cond_waitなど)
タスク並列	あり(限定的)	なし(明示的に記述)
ワークシェアリング構文	あり (omp for)	なし

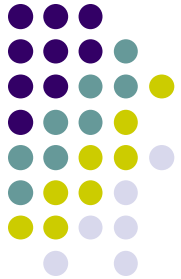
本授業のレポートについて



- 基礎編から一問＋応用編から一問、計二問のレポート提出を必須とします
- 基礎編
  - OpenMP+MPIの、選択問題の中から一問以上
- 応用編
  - CUDA
  - PGAS (予定)

  } この中から一問以上

基礎編課題説明/Report (1)



以下の[1]—[3]のどれか一つについてレポートを提出してください．二つ以上でも良い．

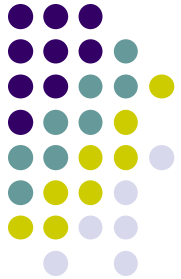
[1] Advectionサンプルプログラムを，以下のいずれかの方法で並列化してください．

a) OpenMP

b) MPI

- MPIで一般のサイズに対応するには端数処理が必要である。本レポートではそれは必須ではない
- より良いアルゴリズムにしてもよい．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？

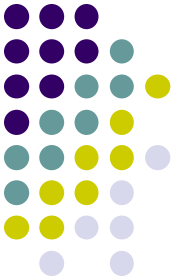
基礎編課題説明/Report (2)



[2] MPIで並列化され，メモリ利用量を抑えた行列積プログラムを実装してください (予定)

- mm-mpiサンプルの改造でよい
- データ分割は本授業の通りでもそれ以外でもよい
- 端数処理はあった方が望ましいが，必須ではない

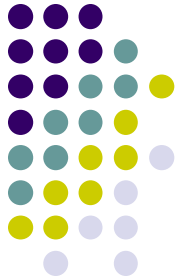
基礎編課題説明/Report (3)



[3] 自由課題：任意のプログラムを，OpenMPまたはMPI(MPI-2も可)を用いて並列化してください．

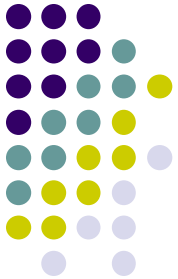
- 単純な並列化で済む問題ではないことが望ましい
  - スレッド・プロセス間に依存関係がある
  - 均等分割ではうまくいかない、など
- たとえば，過去のSuperConの本選問題
  http://www.gsic.titech.ac.jp/supercon/
  たんぱく質類似度(2003)，N体問題(2001)・・・
  入力データは自分で作る必要あり
- たとえば，自分が研究している問題

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
  - プログラムについては，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．分かりにくいディレクトリ名推奨

基礎編の課題の提出について



- 提出期限
  - 7/1 (月)
- レポート形式
  - PDF, Word, テキストファイルのいずれか (その他は相談)
  - プログラムも添付（前ページ参照）
- 送り先： endo@is.titech.ac.jp
- メール題名： ppcomp report

About Account

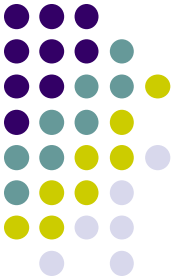
- TSUBAME2のアカウントができたら，連絡してください．授業用のTSUBAMEグループへ登録します．

Subject: TSUBAME2 ppcomp account

To: endo@is.titech.ac.jp

- 専攻・研究室
- 学年
- 氏名
- アカウント名

次回/Next Lecture



- 5/13(Mon)
  - MPI (1)
- 次々回5/20は特別講義予定
  - GSIC下川辺先生による超並列アプリに関する講演
  - TSUBAME2.0スパコン見学

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

    → 2013年度前期情報(OCW) → 講義ノート