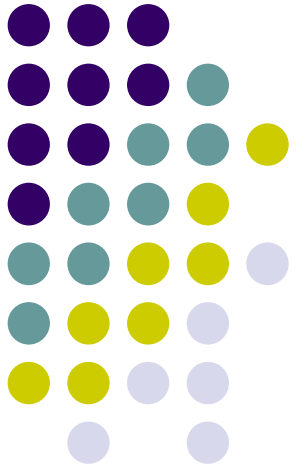
2014年度
実践的並列コンピューティング
第5回

OpenMPによる
共有メモリプログラミング (3)



遠藤 敏夫

endo@is.titech.ac.jp

並列ソフトウェアを作るポイントは？

- 正しい逐次アルゴリズムか？
- 速い逐次実装か？
- 正しい並列アルゴリズムか？
  - 依存関係を壊していないか
  - Race conditionはないか
- 速い並列実装か？
  - ノード内で速いか
  - ノード間で速いか

「正しい」並列プログラムに向けて

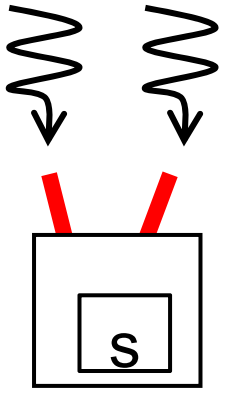
- OpenMPは指定個所を「並列に実行」してくれるが、「正しさの保証」はしてくれない ⇒プログラマの責任

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

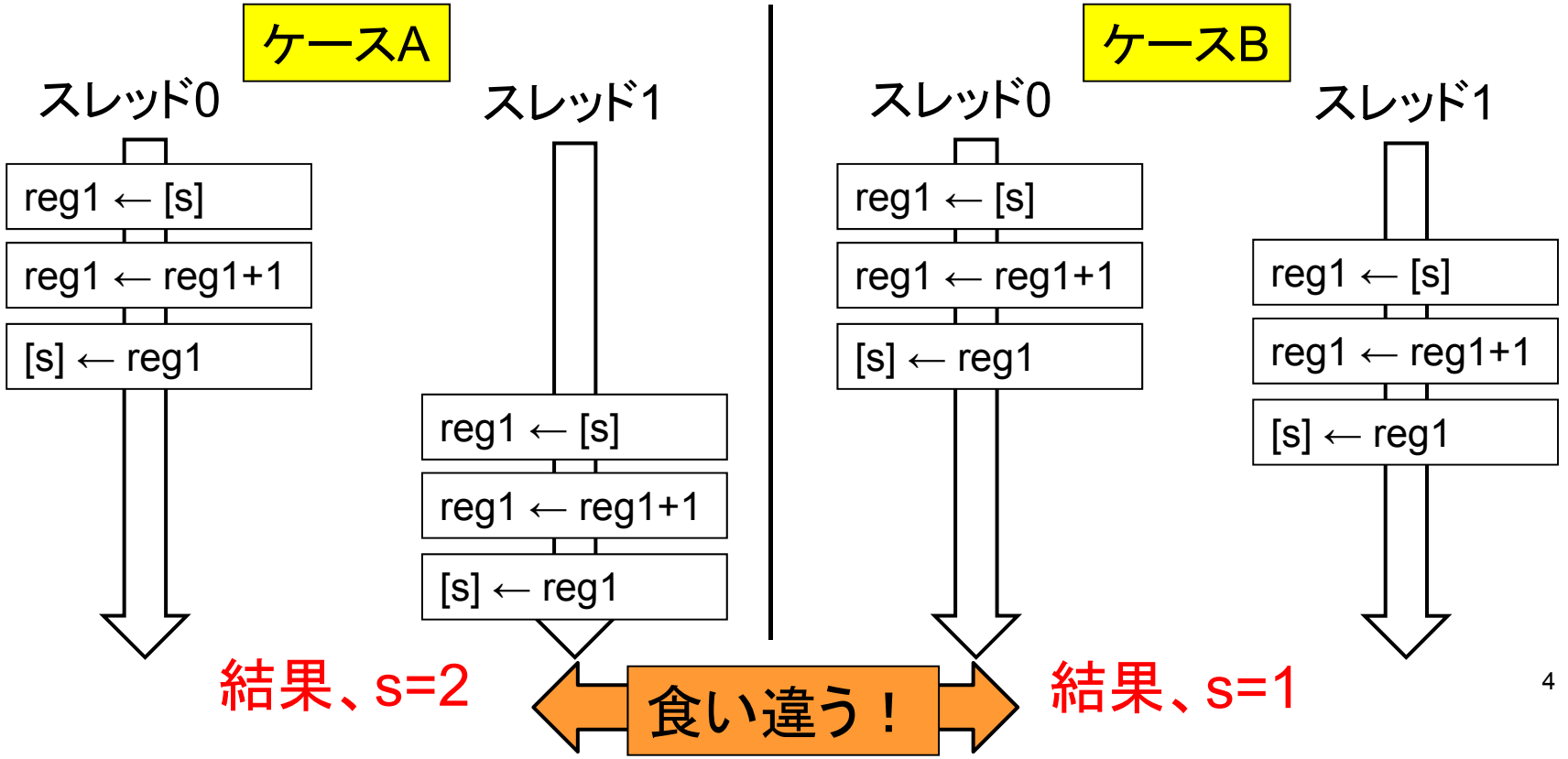


2スレッドがそれぞれsを1増やすので、結果はs=2 ??

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：



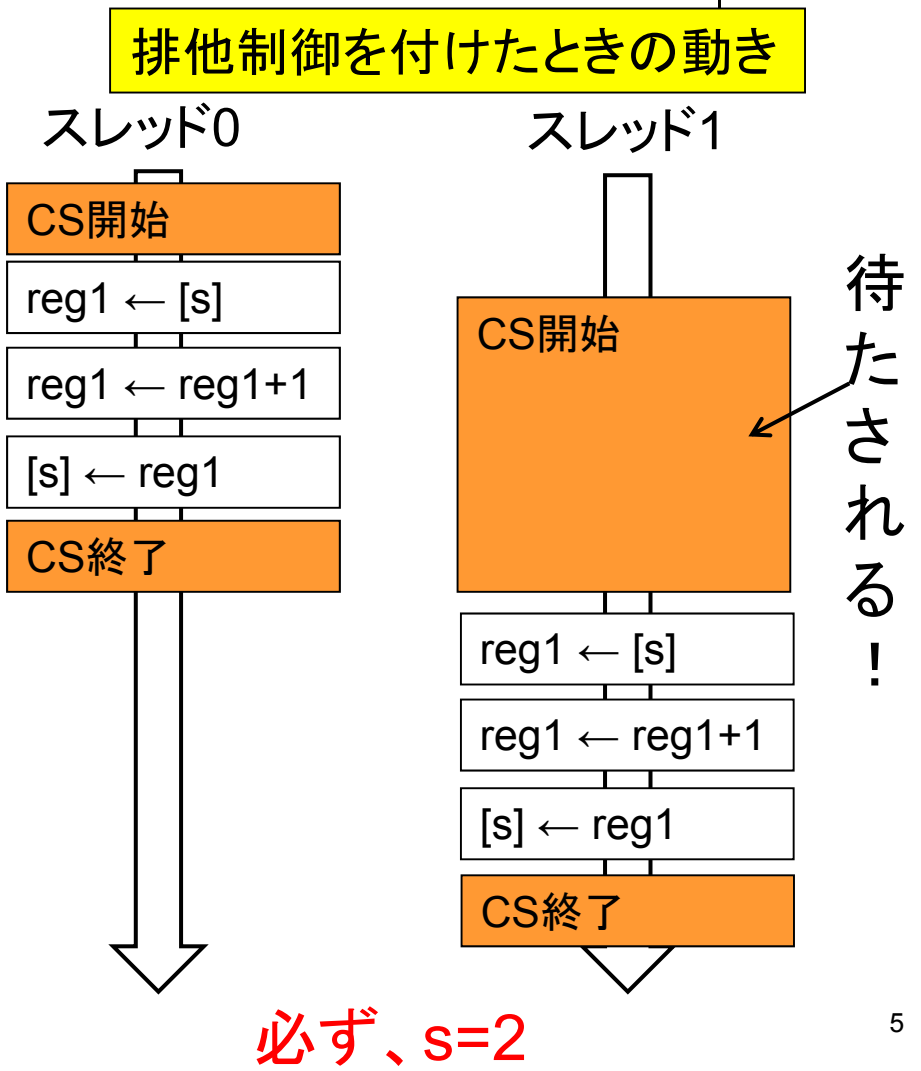
正しい並列プログラムに向けて: Mutual exclusion

Mutual exclusion(排他制御):

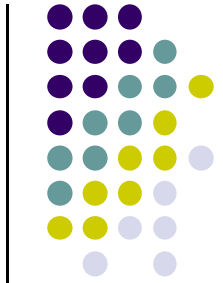
あるコード部分に、同時に1スレッドしか入れないようにする

- 1スレッドしか入れない部分をcritical sectionと呼ぶ

⇒　右の例では競合状態はなくなり、必ずs=スレッド数となる



OpenMPにおける Mutual exclusion



OpenMPでは「omp critical」をつけると、直後の文・ブロックはクリティカルセクションになる

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

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

並列化の敵：ボトルネック

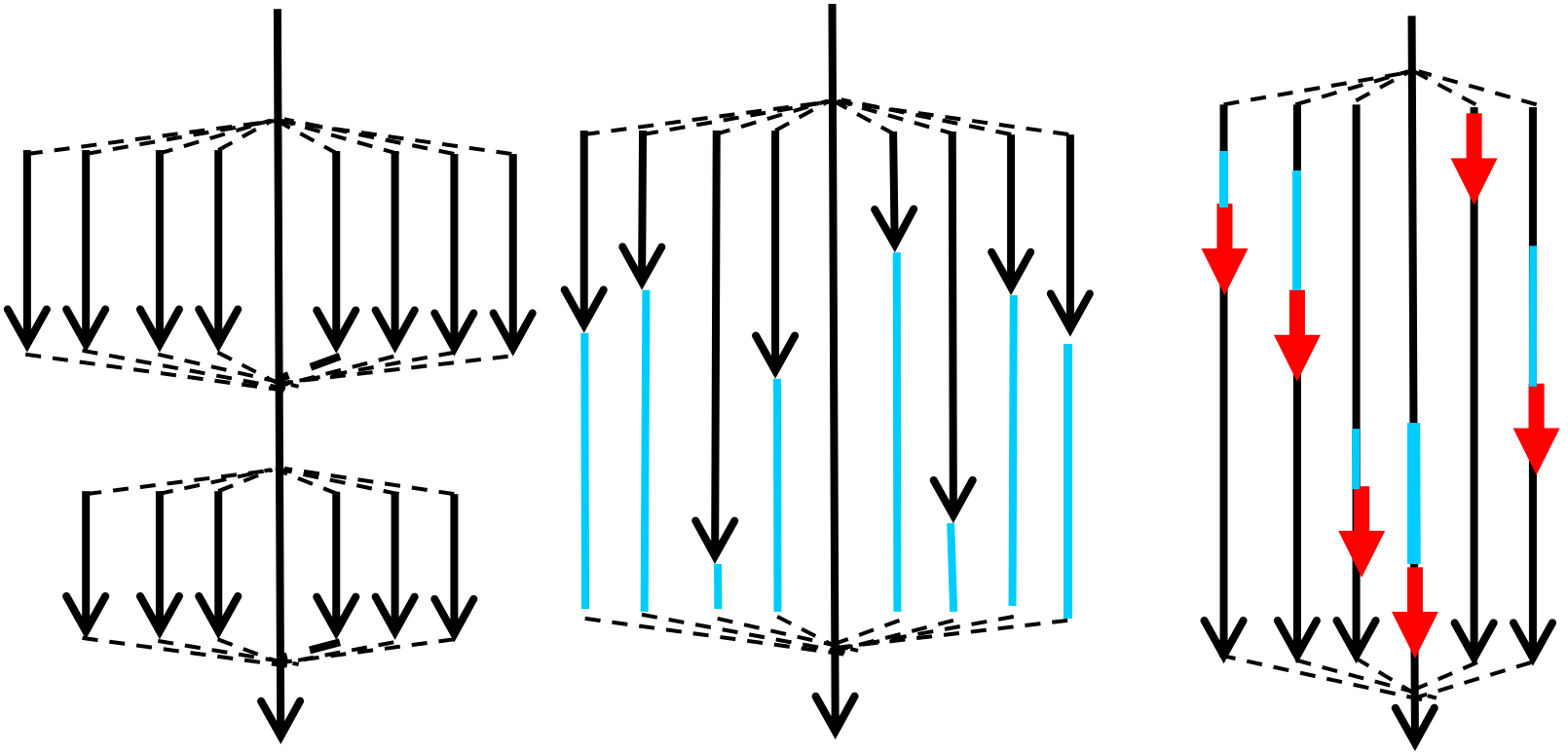
- 多くのアルゴリズムは、並列化できる場所とできない場所を含む⇒できない場所をボトルネックと呼ぶ

さまざまなボトルネック

逐次部分による
ボトルネック

負荷不均衡による
ボトルネック

Critical section
によるボトルネック



ほかにも、アーキテクチャ上のボトルネックなども

アムダールの法則

- アムダールの法則(Amdahl's law)
  - アルゴリズムのうち、
    - 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

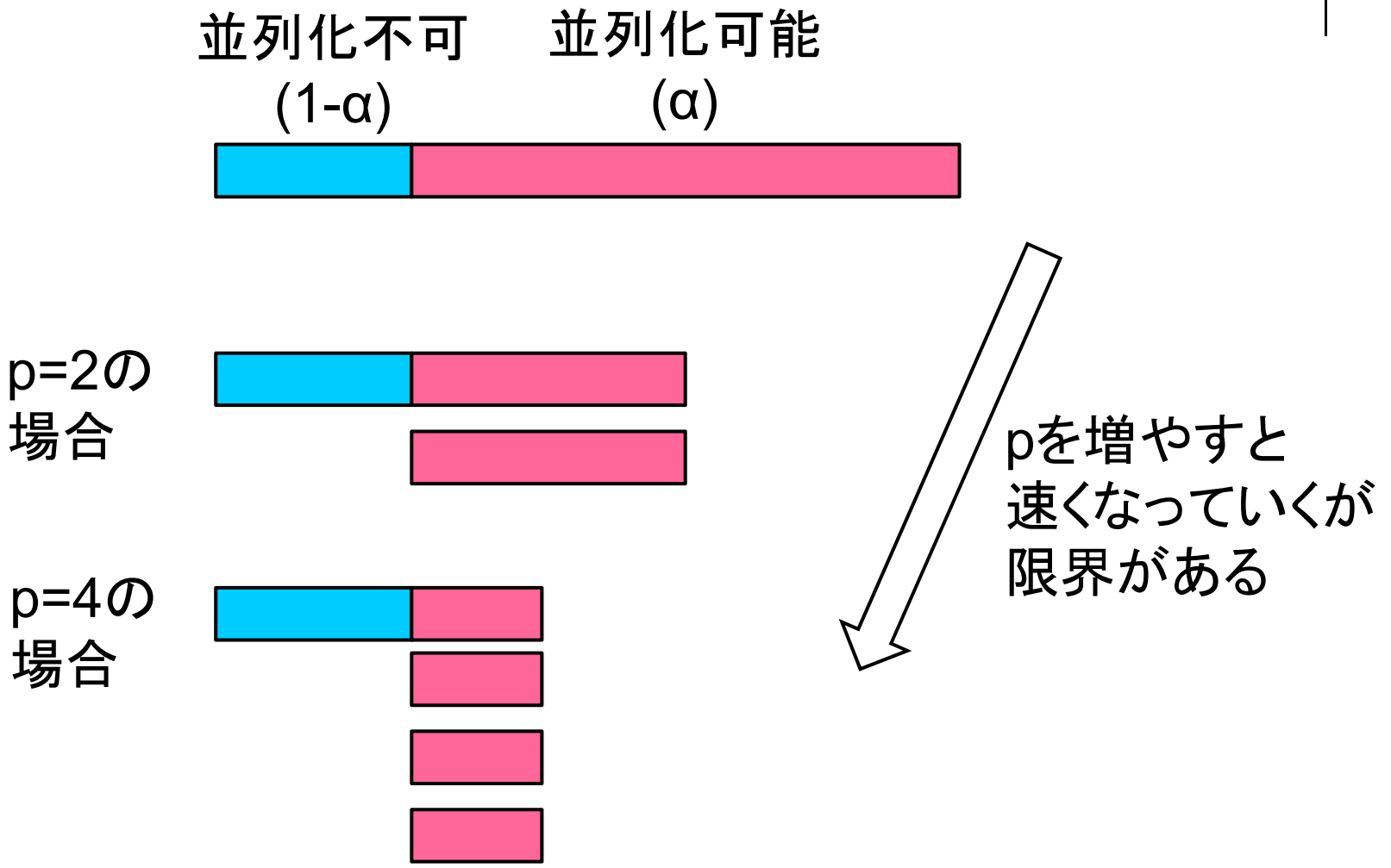
⇒ pプロセッサでの相対実行時間(逐次=1)は

$(1 - \alpha + \alpha / p)$

速度向上率は逆数の$1 / (1 - \alpha + \alpha / p)$

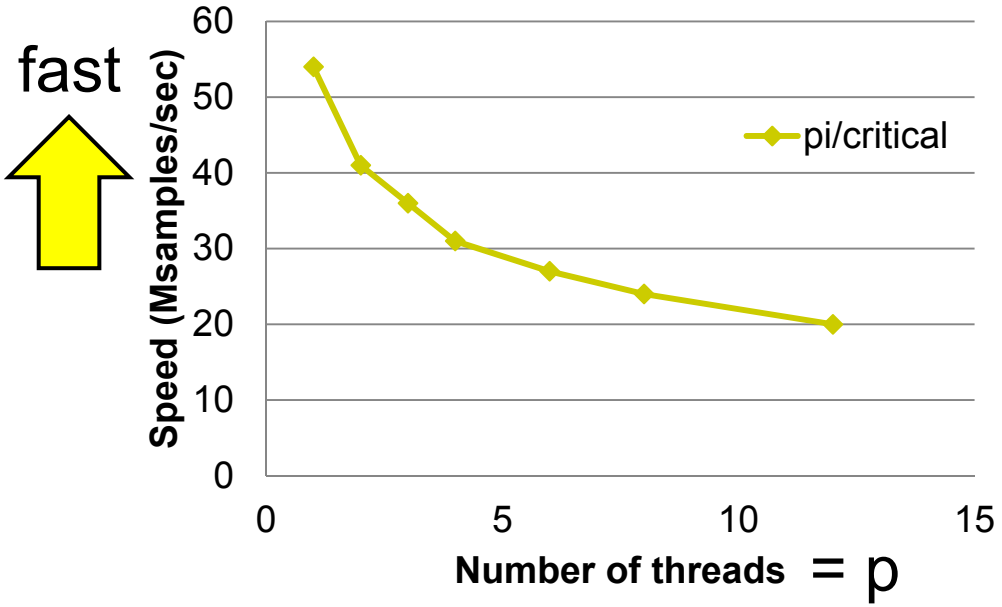
  - ボトルネックだらけのプログラムは $\alpha \fallingdotseq 0$ なので、どんなにpを増やしても速くならない

アムダールの法則



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



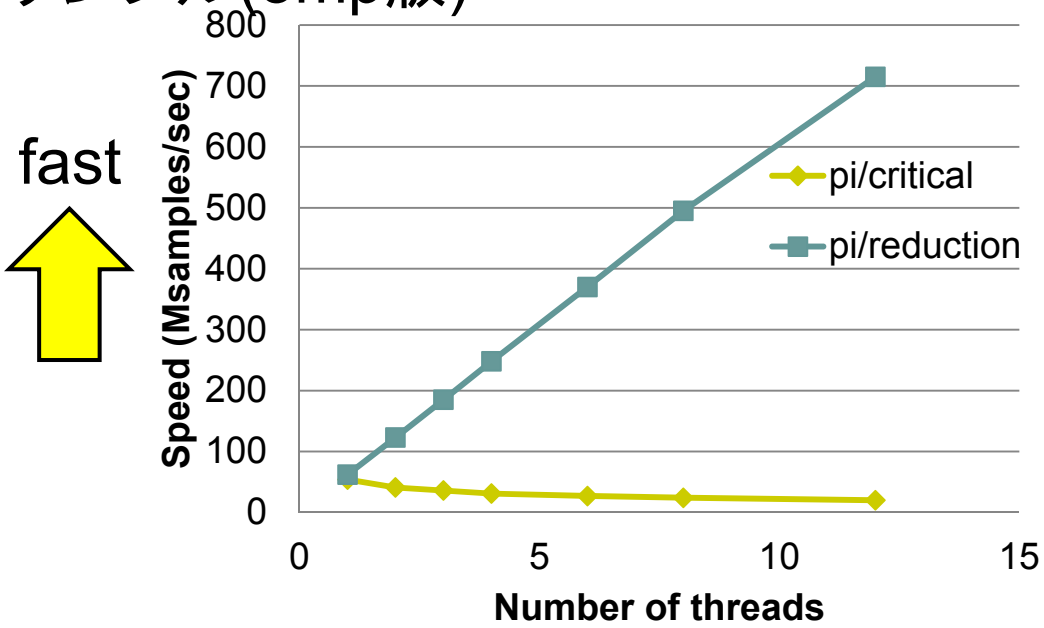
アムダールの法則どおりなら、どんなにαが悪くても(0に近くても)、pに対して単調増加のはず

⇒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

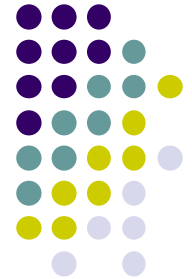
CriticalとReductionの性能比較

- 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

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

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:

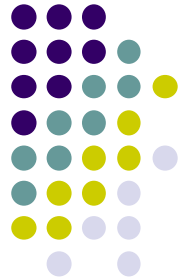
- 配列サイズや時間ステップ数を可変パラメータにしてみる。引数で受け取って、配列をmallocで確保するようにする、など。
- より良いアルゴリズムにしてみる。ブロック化・計算順序変更でキャッシュミスを減らせないか？

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: 5/12(月)

- OpenMP (4)
- 次々回: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