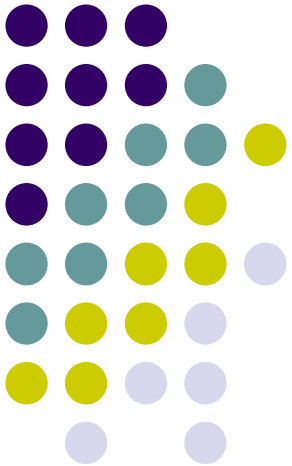
# 2015年度
# 実践的並列コンピューティング
# 第5回

OpenMPによる
共有メモリプログラミング (3)

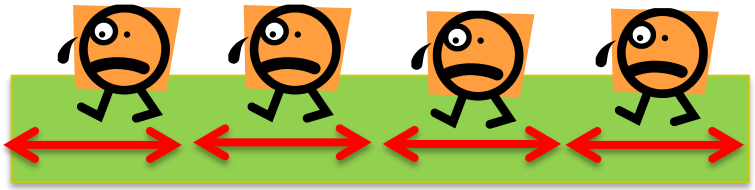
遠藤 敏夫

endo@is.titech.ac.jp

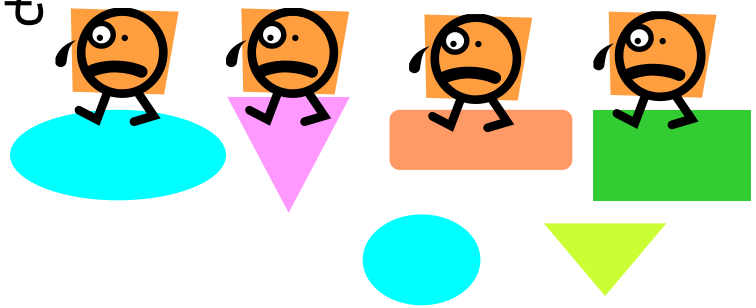


# 今回のテーマ:タスク並列
# データ並列 vs タスク並列

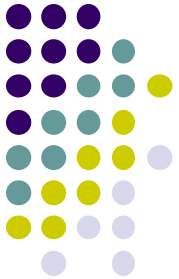
- データ並列性:
  - 各スレッドが、同じ処理をデータの別の部分に対して行うことによる並列性



- タスク並列性:
  - 各スレッドが、別の処理を行うことによる並列性
    - 再帰的な並列性の場合も多い → 仕事の量があらかじめ分からないことも



# OpenMPにおけるデータ並列とタスク並列



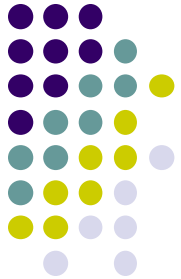
各スレッドの仕事を逐一記述すればなんでも(データ並列だろうがタスク並列だろうが)記述できるのだが…

ここでは比較的高レベルのOpenMP機能を利用

多数の仕事をどうやってスレッドに割り当てているかに注目

- #pragma omp for
  - (だいたい)データ並列に対応
  - 仕事の個数は前もって(forの開始時に)分かっている
    - for (i = 0; i < n; i++) … → n個の仕事をスレッド間で分け合う
- #pragma omp task
  - (だいたい)タスク並列に対応
  - 並列リージョン実行中に、仕事の個数が増えてもよい

# 注意：“task”に対応するOpenMPバージョン



- 今回の内容をコンパイル・実行するには、OpenMP 3.0(2008/5)以降対応のコンパイラである必要
  - OpenMP 3.0からはタスク並列が追加
  - → 再帰関数などの並列化も可能に

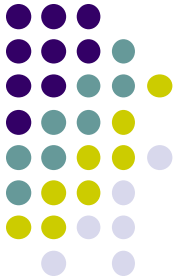
printf(″%d¥n″, _OPENMP);

→ 200505 が表示 → OpenMP 2.5　×　(gcc 4.3.4など)

→ 200805 が表示 → OpenMP 3.0　○　(pgcc 15.4など)

→ 201307 が表示 → OpenMP 4.0　○　(icc 15.0など)

# task構文/taskwait構文



```
#pragma omp task
{
    A;
}

#pragma omp task
     B;

#pragma omp taskwait
```

「task」構文：直後のブロック・文(AやB)を実行するようなタスクを生成する

- タスクは、どれかのスレッドによって実行される
- 子タスクたちと親タスクは論理的に並行される
- タスク生成は再帰的に可能

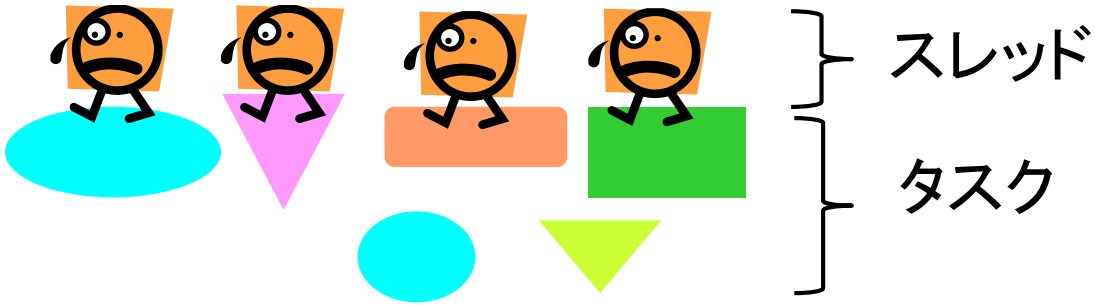
「taskwait」構文：自タスクが生成した子タスクの全ての終了を待つ

# スレッドとタスクの違い

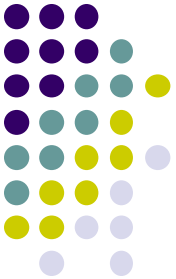
- 「スレッドAとスレッドBは並列に実行される」
- 「タスクAとタスクBは並列に実行される」

何が違う？

- スレッドの数は原則的にOMP_NUM_THREADSで決まる
- タスクの数はプログラム実行時にいくらでも増えるかもしれない(再帰生成可)
- ひまになったスレッドがタスクを次々に実行する

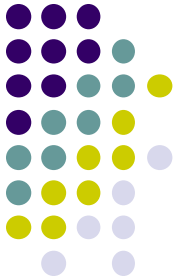


# サンプルプログラム: fib



- フィボナッチ数を求めるサンプルプログラム
  - ~endo-t-ac/ppcomp/15/fib/
- ./fib [n] と実行 → fib(n)を求める
  - ./fib 42 など
- 関数の再帰呼び出しで実現
  - fib(n) = fib(n-1) + fib(n-2)
- 計算量はO(fib(n)) → 計算量は前もってわからない

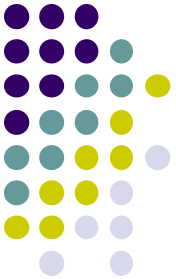
# fibのOpenMPによる並列化 (簡易版)



```
long fib_r(int n)
{
  long f1, f2;
  if (n <= 1) return n;

#pragma omp task shared(f1)
  f1 = fib_r(n-1);

#pragma omp task shared(f2)
  f2 = fib_r(n-2);

#pragma omp taskwait

  return f1+f2;
}
```

- (略) /fib-omp/ ディレクトリ
- 方針:
  - 再帰関数の呼び出し = タスク
- 返り値の変数f1, f2について:
  - 子タスクが代入した値を親タスクが見られるようにshared指定

# Task構文の落とし穴その1



- タスクは多量に作ってもよいが…
- タスクを作るコスト(時間)はやはり大きい！

だいたい:

スレッド生成コスト > タスク生成コスト

>> 関数呼び出しコスト > 一命令のコスト

- 上記の実装では、fib(n)の実行時に、O(fib(n) )個のタスク生成を行っている → 多すぎ
- タスク生成の回数を減らしたサンプル:
  - (略) /fib-omp2/ ディレクトリ
  - 木構造の先のほう(nが一定値以下)では、タスク生成をやめる

# task構文の落とし穴その2

- ひまになったスレッドがタスクを実行し、助けてくれるというルール

→ ひまなスレッドがいる状況を作る必要がある

```
long fib(int n)
{
  long ans;
#pragma omp parallel
  {
#pragma omp single
    ans = fib_r(n);
  }
  return ans;
}
```

←複数スレッドが起動 (並列リージョン開始)

←あえて一人だけが以下を実行

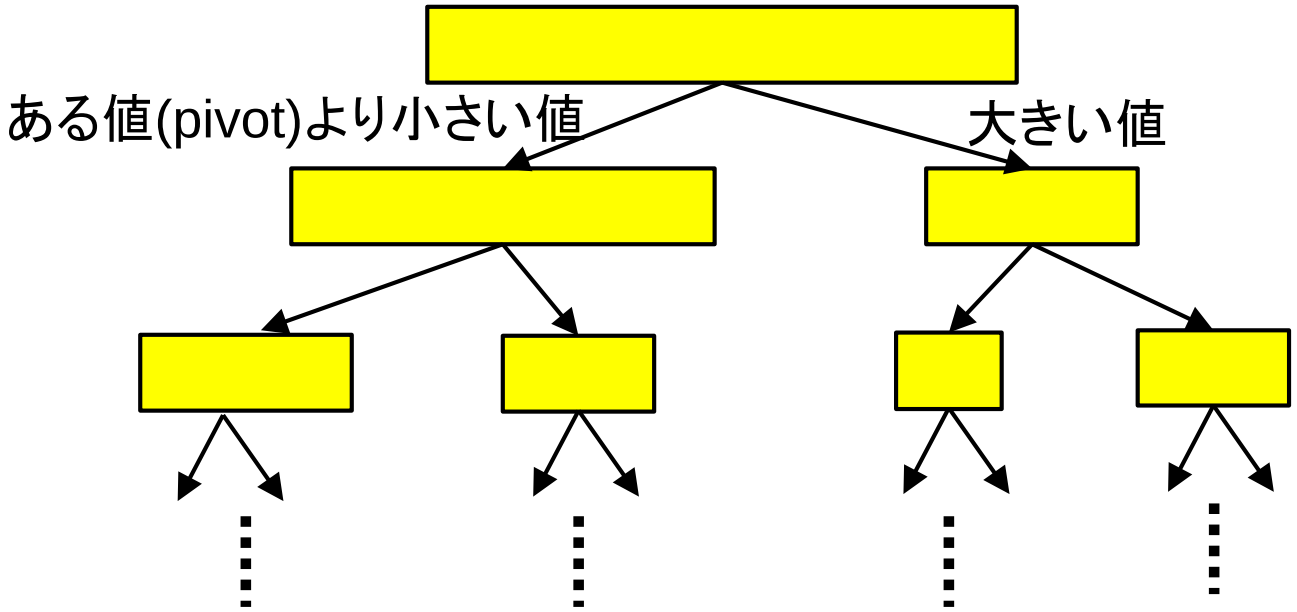
←最初の関数呼び出し(タスク木の根)

←並列リージョン終了

「single」がないと、ただ複数人のスレッドがそれぞれfib_r全体を実行するだけ → 高速になる理由がない

# サンプルプログラム：sort

- クイックソートアルゴリズムにより、n個のdouble要素の配列を昇順にソート
- ~endo-t-ac/ppcomp/15/sort ディレクトリ
- 再帰的に、大小の二分を行うアルゴリズム
  - 計算量：平均的には O(n log n)
  - バブルソートなどのO($n^2$)より効率的



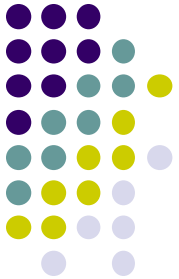
# sortの構造

```
int sort(double *data, size_t n)
{
  size_t i, j;
  double pivot;

  if (n <= 1)  return 0;

  [pivot選択]

  [data配列を二分]   /* O(n) のコスト */
  /* この時点で i が境界 */

  sort(data, i);    /* 左半分を再帰実行 */
  sort(data+I, n-i);    /* 左半分を再帰実行 */
}
```

この並列化は難しい・効果が少ない

これらの二つの仕事を並列に行いたい

# [再掲] 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

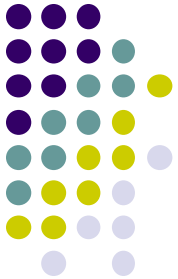
```
#pragma omp for
  for (var = lb; var rel-op b; incr-expr)
    body
```

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



# task構文でできるようになること

- リスト構造の探索は、仕事数があらかじめ不明なのが問題 → task構文の応用で可能に

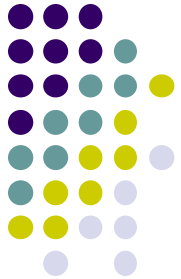
```
#pragma omp parallel
{
#pragma omp single
  {
     for (p = head; p != NULL;
          p = p->next) {
#pragma omp task
        [pに対する仕事]
     }
#pragma omp taskwait
  }
}
```

- リストノード1個に対する仕事 = 1 OpenMP タスク

注意：

- 「リスト長」個のタスクを生成するので、コストに注意
  - 一方、“omp for”のコストは、だいたいスレッド数に比例なので軽い

# 本授業のレポートについて

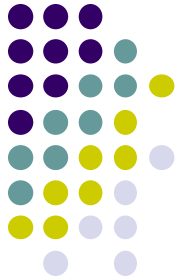


- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

  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)



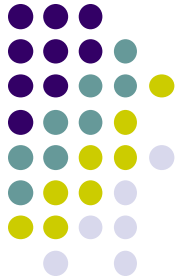
以下の[O1]—[O3]のどれか一つについてレポートを提出してください

[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)



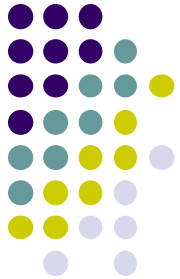
[O2] sortサンプルプログラムを、OpenMPで並列化してください.

- 注意：OpenMP3.0以上のtask対応コンパイラである必要
  - TSUBAMEではpgcc (-mpつき)や icc (-openmpつき)

オプション：

- クイックソート以外のアルゴリズムではどうか?
  - ヒープソート？マージソート？

# OpenMPパート課題説明 (3)



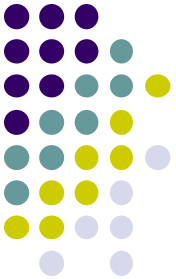
[O3] 自由課題：任意のプログラムを，OpenMPを用いて並列化してください．

- 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
    - 問題サイズを様々に変化させたとき(可能な問題なら)
    - 「XXコア以上で」「問題サイズXXX以上で」発生する問題に触れているとなお良い
  - 高性能化・機能追加など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good
  - 作成したプログラムについても、zipなどで圧縮して提出
    - 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

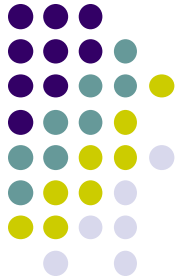
# 課題の提出について



- OpenMPパート提出期限
  - 6/8 (月) 23:59 (変更しました)
- 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
  - レポート本文：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: 6/1(月)



- OpenMP(4)
  - Race condition (競合状態)
  - Mutual exclusion (排他制御)
  - スレッドセーフ
  - アムダールの法則

- 5/25(月)は休講