OpenMP による並列計算の基礎

はじめまして OpenMP

河合 佑太

神戸大学大学院理学研究科

2014/01/10

Outline

Outline

並列化？

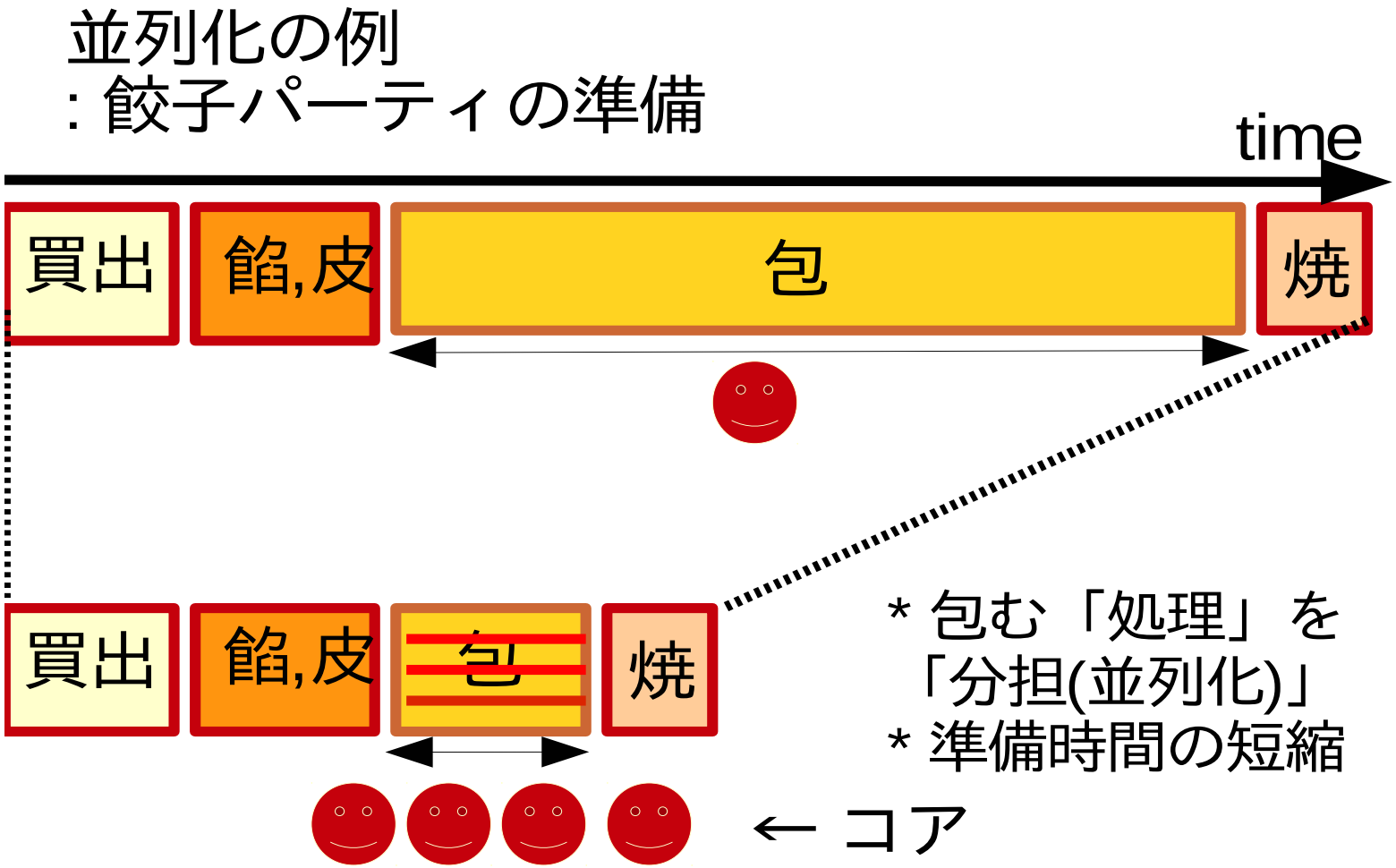

Figure : 並列化のイメージ (by ykawai)

演算処理の高速化の歴史 1

- クロック周波数の高速化, 製造プロセスの微細化
  - クロック周波数は, 10 MHz から 3 GHz へ
  - プロセス・ルールは約 30 年間で数十 μ m から数十 nm へ
  - 近年限界に来ている => Intel の戦略転換 (マルチコア)

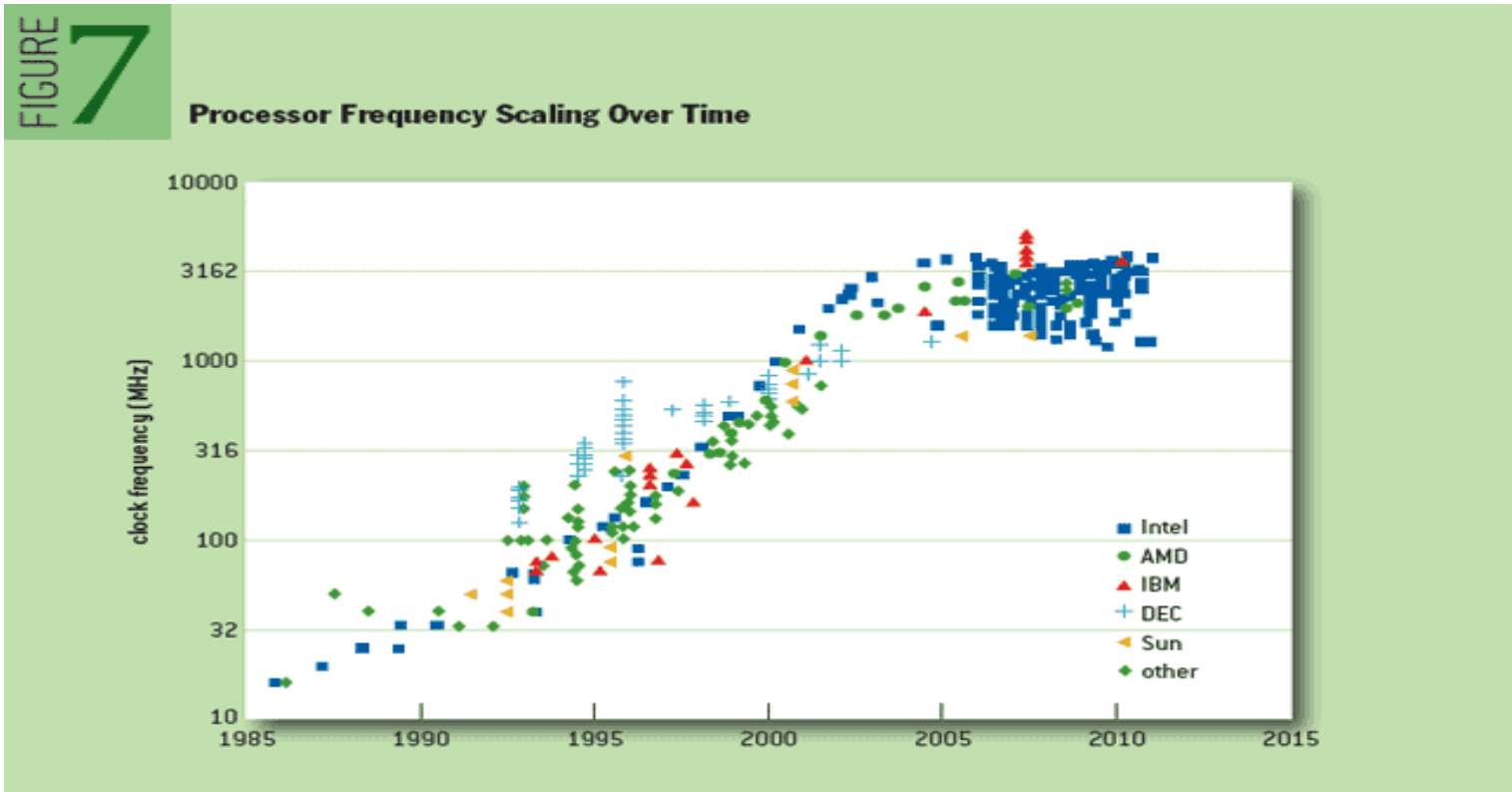

Figure : プロセッサの周波数のスケーリング (Danowitz et al.(2012) : CPU DB: Recording Microprocessor History)

演算処理の高速化の歴史 2

- アーキテクチャの改良
  - メモリ構造の階層化: キャッシュ多段階化
  - ベクトル化: パイプライン, SIMD(シムド)
  - マルチコア: 一つのチップに複数の CPU

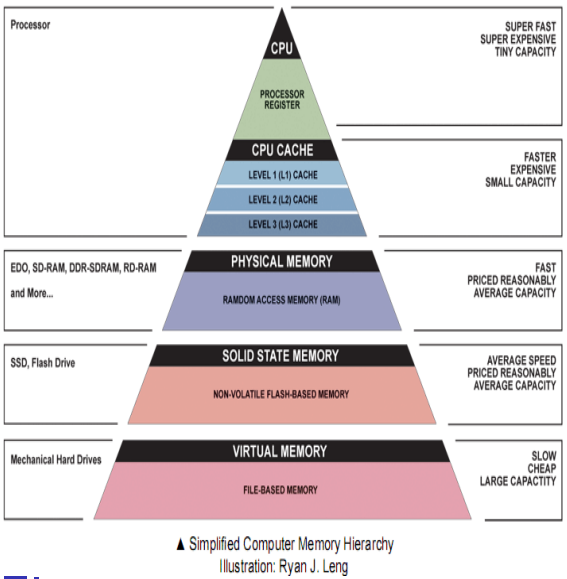

Figure : 簡単化されたメモリの階層構造 (Ryan J. Leng(2007) : The Secrets of PC Memory: Part 1)

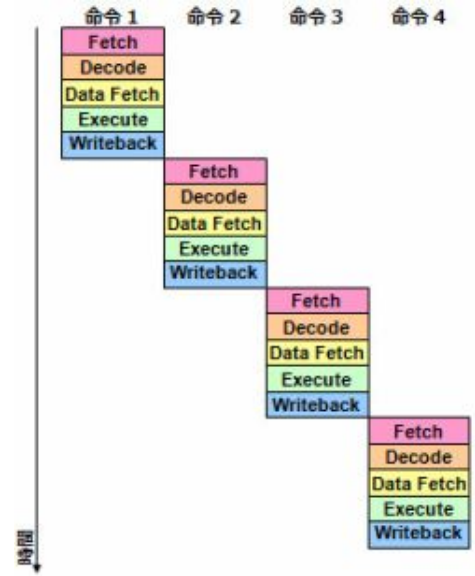

Figure : パイプライン処理 (http://ascii.jp/elem/000/000/552/552029/index-3.html)

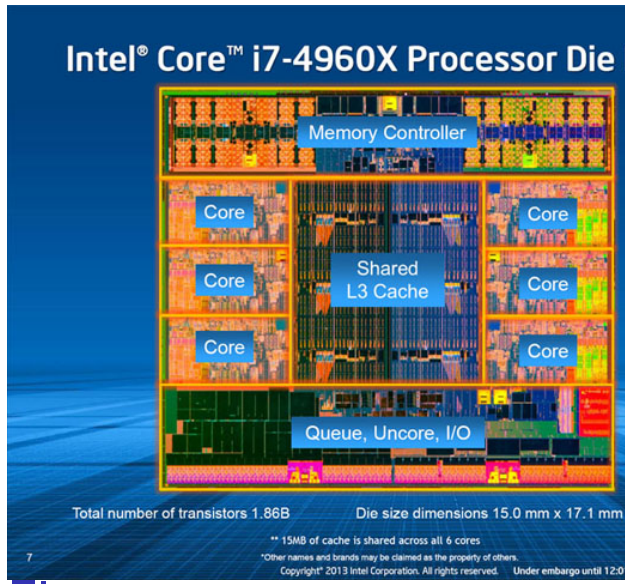

Figure : Intel Corei7-4960X のダイの詳細 (http://news.mynavi.jp/photo/special/2013/ibesp/images/photo002l.jpg)

プロセスとスレッド

- 計算機上で複数の処理を同時に実行する際の, 処理の分割単位
- プロセス
  - ユーザ的には, 実行中の個々のアプリケーション
  - 個々のプロセスは基本的には独立している
  - OS のマルチタスク機能によって, 同時並行的に動作 (マルチプロセス)
- スレッド
  - プロセス ⊃ スレッド
  - 1 プロセス内で複数のスレッドを走らせられる (マルチスレッド)

違いは?

- プロセス間: (基本的に) メモリ共有できない
- スレッド間: メモリ共有できる
  - 同じデータに簡単にアクセスできる

並列化の方式

- スレッド並列
  - 1 プロセス内でスレッドを複数生成し, 各々に CPU コアを割り当てる.
  - 同じメモリ空間にアクセス
  - OpenMP を使ったプログラミングが標準的
- プロセス並列
  - プロセスを複数起動し, 各々に CPU コアを割り当てる.
  - 異なるメモリ空間にアクセス
  - MPI を使ったプログラミングが標準的
- ハイブリッド並列: スレッド並列 + プロセス並列

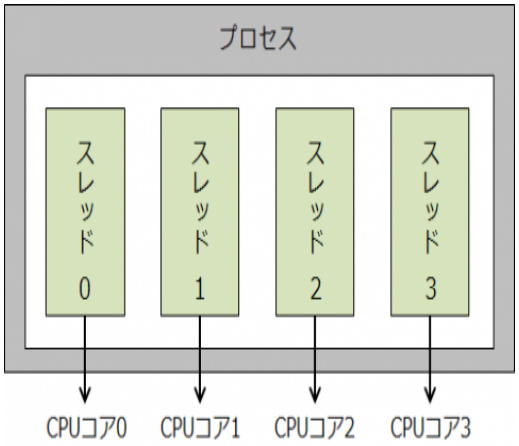

Figure : スレッド並列のイメージ (http://web.kudpc.kyoto-u.ac.jp/manual/ja/parallel)

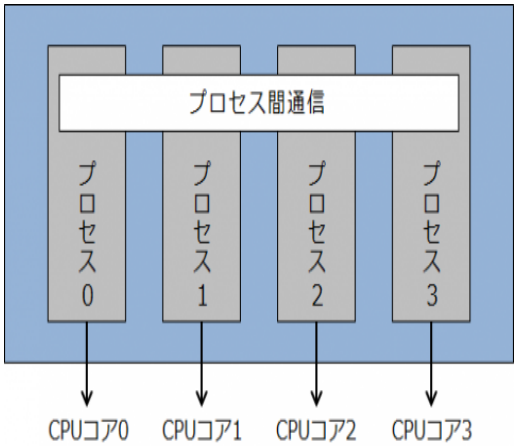

Figure : プロセス並列のイメージ (http://web.kudpc.kyoto-u.ac.jp/manual/ja/parallel)

並列化に関する留意点

アムダールの法則

(高速化率) $\leq 1/(f + (1 - f)/p)$

- p:プロセッサ数, f:逐次処理の割合
- 並列化効率は, 逐次処理の割合で制限される

並列化に伴うオーバーヘッド

- 計算負荷の不均衡: 同期待ち時間の増大
- 並列アルゴリズムのオーバーヘッド
  - (例) データのコピー, スレッド生成
- データ通信のオーバーヘッド

Outline

OpenMP(Open Multi-Processing) とは

- 共有メモリ型計算機向けの並列プログラミングのための API(`http://openmp.org/wp/`)
  - C,C++,Fortran から利用できる
  - OpenMP Architecture Review Board(ARB) が管理
  - 1997 年に最初の API 仕様を発表. 現在, Version 4 が最新.
- 特徴
  - 逐次コードに指示行を挿入するだけで並列化可能
    - 逐次プログラムと並列プログラムを共通にできる
    - 比較的デバックが容易 (?)
  - 移植性が高い
    - 修正なしに様々な共有メモリ型計算機で実行可
  - MPI によるプロセス並列化の方がしばしば高速 (?)
    - 共有メモリ型の並列処理の問題: データの局所性, メモリアクセスの衝突など

OpenMP の実行モデル

- fork-join モデル
  - 最初: 一本のスレッド (マスタースレッド)
  - 並列部分
    - 開始時: 複数のスレッドに分岐 (Fork)
    - 終了時: マスタースレッドのみに戻る (Join)

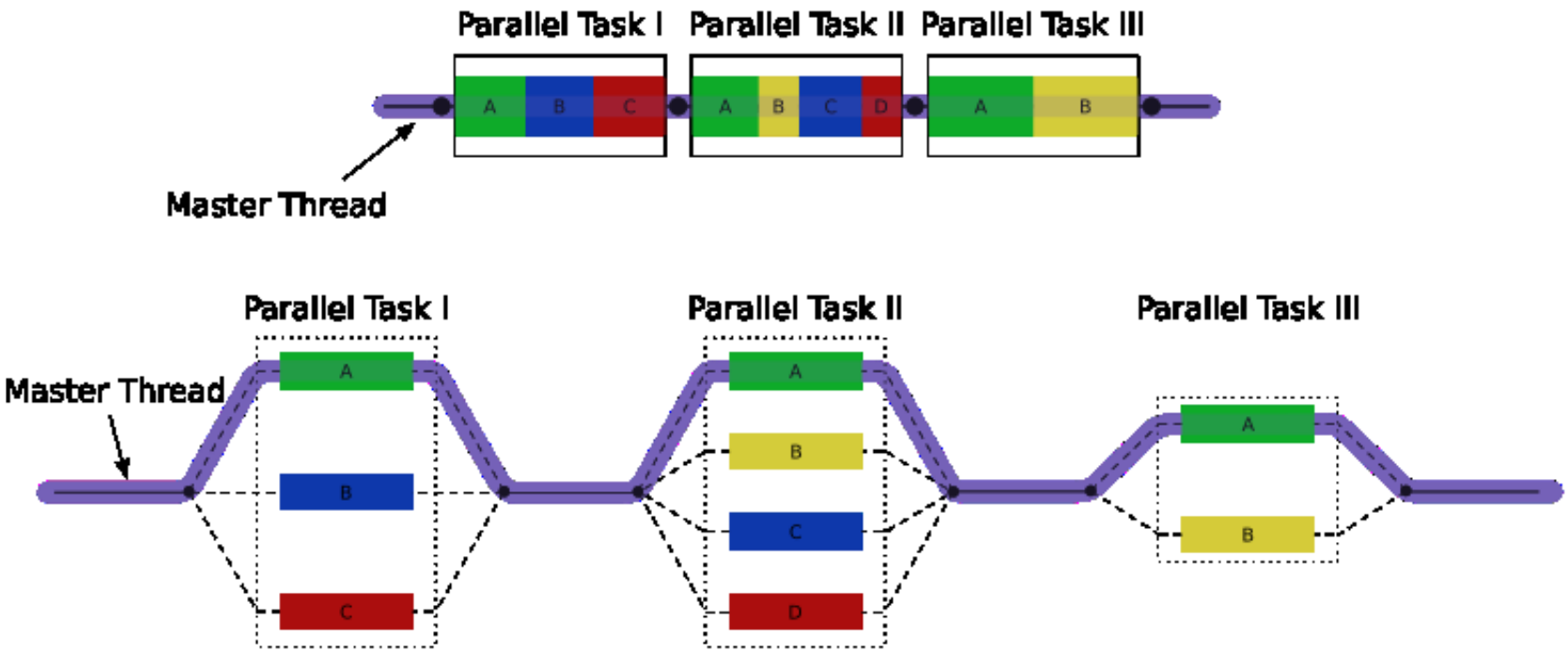

Figure : fork-join モデル (http://en.wikipedia.org/wiki/OpenMP)

OpenMP の構成要素

- C/C++/Fortran で書かれたもとの逐次プログラム
- 指示文, 指示節
  - 並列化すべき場所, 並列化方法を指定
  - Fortran では !$omp で開始

    （例） `!$omp parallel do`
- 実行時ライブラリ関数
  - 実行環境ルーチン, ロックルーチン, 時間計測ルーチンなど

    （例） `omp_set_num_threads, omp_get_wtime`
- 環境変数
  - 並列実行部分で使うスレッド数などを指定

    （例） `OMP_NUM_THREADS, OMP_SCHEDULE`

OpenMP による並列プログラミング 1

- 並列実行領域構文 (parallel 構文)
  - parallel ~ end parallel で囲まれた領域を並列実行

逐次コード

```
program hello_world
 implicit none

 write(*,*) "Hello␣World!"

program hello_world
```

並列コード

```
program hello_world
 use omp_lib
 implicit none
 integer :: myID

 !$omp parallel
 write(*,*) "Hello␣World!␣My␣id␣is␣", &
   & omp_get_thread_num()
 !$omp end parallel

program hello_world
```

OpenMP による並列プログラミング2

- ワークシェアリング構文
  - 並列実行領域での作業負荷を各スレッドに分担させる
- ワークシェアリング構文
  - ループ構文：do ループを分割実行
  - single 構文：全スレッドの内一つのスレッドのみで実行
  - sections 構文：依存関係のない異なる処理を各々のスレッドで実行
  - workshare 構文

OpenMP による並列プログラム例 2

- ループ構文: do 構文
  - 並列実行領域において do ループを分割し, 各スレッドに割り当てる.
  - デフォルトでは均等に分割される. (参考: schedule 句)

逐次コード

```
do i=1, 4000
  a(i) = b(i) + c(i)
end do

do i=1, 4000
  a(i) = b(i)/c(i)
end do
```

並列コード

```
!$omp parallel
!$omp do
do i=1, 4000
  a(i) = b(i) + c(i)
end do
!$omp end do
!$omp end parallel

!$omp parallel do
do i=1, 4000
  a(i) = b(i)/c(i)
end do
```

OpenMP による並列プログラミング (補足)

- 並列実行領域構文
- ワークシェアリング構文
- データ環境構文: データの格納属性の指定
  - (例) private, shared 他
- 同期構文: スレッド間の同期をとる
  - (例) barrier, critical, single 他
- 実行時関数, 環境変数

コンパイル・実行方法

コンパイル方法

- コンパイラにより異なる．基本的にオプションを付ける．
  - GCC(gcc, g++, gfortran): -fopenmp
  - Intel Compiler(icc, icpc, ifort): -openmp

```
$gfortran -fopenmp -O2 -o sample1 sample1.f90
```

実行方法

- 使用するスレッド数を環境変数 OMP_NUM_THREADS に指定．

```
$export OMP_NUM_THREADS=4 #Set the number of thread 4
$./sample1
```

Outline

まとめ

- マルチコア CPU では, 時間のかかる処理を各コアに分担させる (並列化) ことによって, 処理時間を短縮することができる.
  - 並列プログラミングが必要である.
- マルチコア CPU 上の並列化でよく使われる OpenMP による並列プログラミングの基礎を紹介した.

Outline

参考文献

- スケーラブルシステムズ株式会社. OpenMP プログラミング入門 (Part1).
- 谷口 隆晴. OpenMP を用いた並列計算 (1).
- 松本 洋介. スカラーチューニングと OpenMP によるコードの高速化.
- http://www.nag-j.co.jp/openMP/ OpenMP 入門.

Outline

準備

- `http://itpass.scitec.kobe-u.ac.jp/~ykawai/` から,
  - omp_helloworld.f90
  - omp_calcPI.f90
  - itpass-seminar_20140110.pdf(本日の資料)

  をダウンロードしてください.

演習 1

- omp_helloworld.f90 に !$omp parallel 等を追加

omp_helloworld_parallel.f90

```
program hello_world
 use omp_lib
 implicit none

 !$omp parallel
 write(*,*) "Hello world!"
 write(*,'(a,i1)') "My world is ", &
  & omp_get_thread_num()
 !$omp end parallel

end program hello_world
```

コンパイル・実行

```
$gfortran -fopenmp -O2 omp_helloworld.
$export OMP_NUM_THREADS=4
$./a.out
```

演習 2

- omp_calcPI.f90 に !$omp parallel do 等を追加

omp_calcPI.f90(subroutine numInt の一部)

```
PI = 0
!$omp parallel do private(x) reduction(+:PI)
do i=1, N
 x = h*(i - 0.5d0)
 PI = PI + 4d0/(1d0 + x**2)
end do
!$omp end parallel do

PI = PI*h
```

コンパイル・実行

```
$gfortran -fopenmp -O2 omp_calcPI.f90
$export OMP_NUM_THREADS=4
$./a.out
```