KOBE HPC サマースクール 2022（初級）

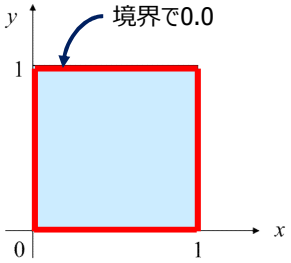
# 11. 熱伝導問題の並列計算
# 12. ハイブリッド並列

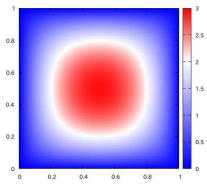


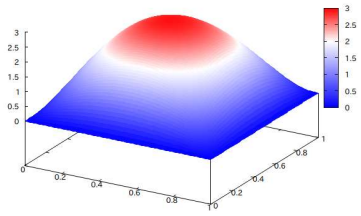
## 2次元定常熱伝導問題

- 2次元正方形領域 [0,1]×[0,1] に一定の熱を加え続けた時の領域の温度がどうなるか？

$$\Delta u + 10 = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 10 = 0$$





- 境界条件

$u(0, y) = 0.0 \quad (0 \le y \le 1)$

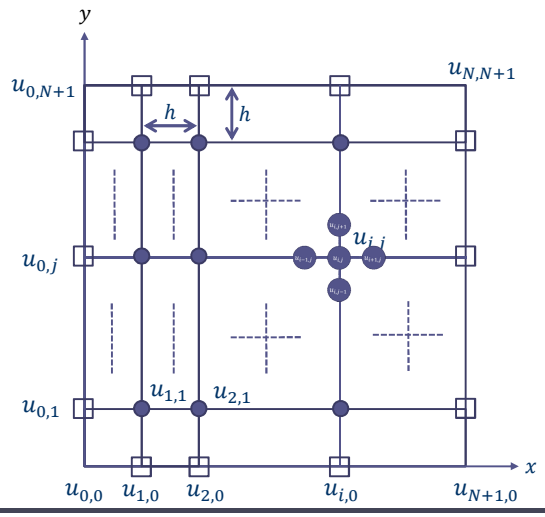
$u(1, y) = 0.0 \quad (0 \le y \le 1)$

$u(x, 0) = 0.0 \quad (0 \le x \le 1)$

$u(x, 1) = 0.0 \quad (0 \le x \le 1)$

## 離散化格子と変数

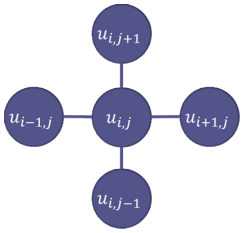
$x$方向，$y$方向どちらも $(N+1)$ 等分する．$h = 1/(N+1)$




$u_{i,j} \triangleq u(ih, jh)$ $(i,j = 0,1,2,\ldots,N+1)$

□ 境界条件として既知の値
● 内部の格子点（未知数）

内部の格子点に関する関係式を作る．

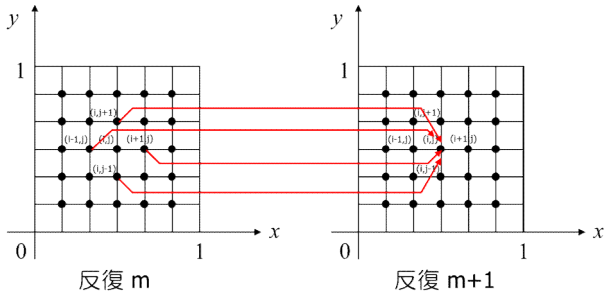





## 熱伝導問題へのヤコビ法の適用

- 反復ベクトル生成のアルゴリズム

```
for( i=1; i<=N; i++ ) {
  for( j=1; j<=N; j++ ) {
    u_{i,j}^{(m+1)} = (u_{i-1,j}^{(m)}+u_{i+1,j}^{(m)}+u_{i,j-1}^{(m)}+u_{i,j+1}^{(m)})*0.25 + 10.0*dx*dx;
  };
};
```



## 演習11-1 ：プログラム heat2d.c の実行

- 2次元定常熱伝導問題の逐次プログラム heat2d.c をコンパイルして実行せよ.

```
$ cp /home/guest60/share/heat2d.c ./
$ icc -O3 heat2d.c
$ ./a.out
```

```
[Fortran]
$ cp /home/guest60/share/heat2d.f90 ./
$ Ifort -O3 heat2d.f90
$ ./a.out
```

- プログラムでは N = 128 の時に，反復ベクトルの差のノルム $\left\|u_{i,j}^{m+1} - u_{i,j}^{m}\right\|_2$ がある閾値$\epsilon = 1.0 \times 10^{-3}$ より小さくなったら反復を停止している.
- プログラム実行が正常に完了したら「To visualize data,…」というメッセージが出てくる．これの指示に従うと計算結果を図示できる（gnuplot と X11 が必要. 事前に「module load gnuplot」で gnuplot 使用環境を load しておく）.

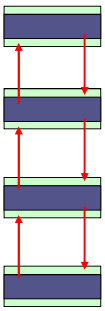


## heat2d.c のMPIによる並列化（Cの場合）

- 並列化のヒント

1. 2次元配列 u（$u^m$），u_new（$u^{m+1}$）をブロック行分割しておく.
2. u[is:ie][] の計算をする前に,
   - 上のプロセスの u の ie 行を下のプロセスの (is-1) 行へ,
   - 下のプロセスの u の is 行を上のプロセスの (ie+1) 行へ,

   送る.
   - 並列化 sr_matrix.c を参考にして，MPI_Sendrecv を用いて送受信.
3. u の is ～ ie 行の計算を行う．計算結果は u_new に格納.
4. 反復ベクトルの差のノルムの部分和を計算し, MPI_Allreduce で総和を取り，収束の判定を行うようにする.
   - ローカルで残差の部分和を計算し，各プロセスでノルムを計算する.
5. u_new を u に入れなおして，手順2に戻る.



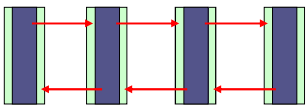
上下のプロセスから1行を受信
（受信用の領域を確保しておく）

# heat2d.f90 のMPIによる並列化（Fortranの場合）

- 並列化のヒント

1. 2次元配列 u（$u^m$），u_new（$u^{m+1}$） をブロック列分割しておく．
2. U(:,js:je) の計算をする前に，
   - 左のプロセスの u の je 列を右のプロセスの (js-1) 列へ，
   - 右のプロセスの u の js 列を左のプロセスの (je+1) 列へ，

   送る．
   - 並列化 sr_matrix.f90 を参考にして，MPI_Sendrecv を用いて送受信．
3. u の js ～ je 列の計算を行う．計算結果は u_new に格納．
4. 反復ベクトルの差のノルムの部分和を計算し，
   MPI_Allreduce で総和を取り，収束の判定を行うようにする．
   - ローカルで残差の部分和を計算し，各プロセスでノルムを計算する．
5. u_new を u に入れなおして，手順2に戻る．

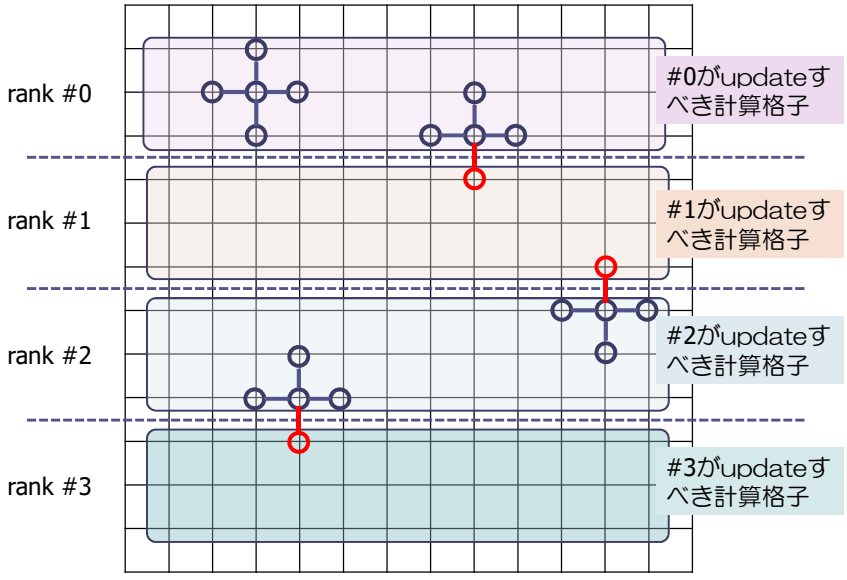


左右のプロセスから1行を受信
（受信用の領域を確保しておく）



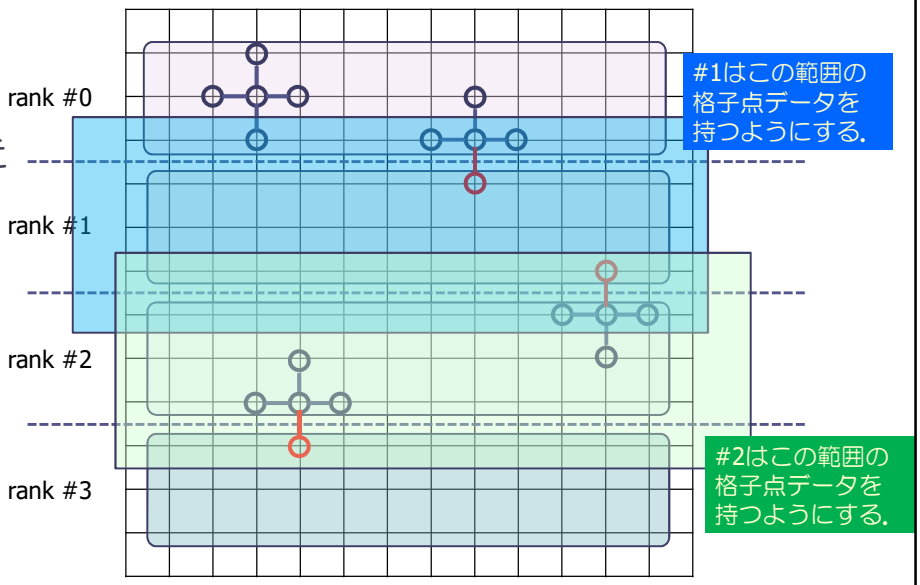
# Cの場合の処理のイメージ

※Fortranの場合はこれを横に寝かしたようなイメージになる



## Cの場合の処理のイメージ

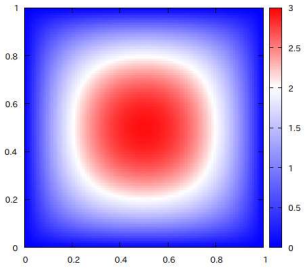
※Fortranの場合はこれを横に寝かしたようなイメージになる






## 演習11-2 heat2d.{c/f90} の並列化

- heat2d.c, heat2d.f90 をMPIを用いて並列化せよ.
- 計算結果を gnuplot 等で図示し，非並列版と同じ結果になることを確認せよ.
  - 計算結果の出力は，プロセス 0 にデータを集め※，ファイルに出力する.
  - 集め方（通信の仕方）は複数ある．考えてみよ.



- N=128とし，プロセス数1, 2, 4, 8 として，反復開始から終了までの計算時間を計測し，速度向上率を求めよ．また，それをグラフにせよ.
- 時間計測時は反復計算途中データのファイル出力を OFF にすると良い.
  - プログラム中のパラメータ INTV の値を 1 にするとデータのファイル出力が抑制される.
  - ファイル出力がないので，通信によるデータの収集（上述の※）も反復計算中は必要ない.

## 演習12-1 ハイブリッド並列化

- MPI を用いて並列化した heat2d.{c/f90} に，さらに OpenMP のディレクティブを挿入し，タスク並列とプロセス並列によりさらに実行時間が削減されることを確認せよ.
  - #pragma omp parallel などを利用する（C言語）.
  - 冒頭で omp.h をインクルードするのを忘れずに（C言語）.
  - !$OMP parallel などを利用する（Fortran言語）.
  - 冒頭で !$ use omp_lib を宣言するのを忘れずに（Fortran言語）.
- コンパイル
  - `icc -qopenmp xxxxx.c -lmpi`
  - `ifort -qopenmp xxxxx.f90 -lmpi`



## 【サンプル】Hybrid並列プログラムの実行シェル（jobh.sh）

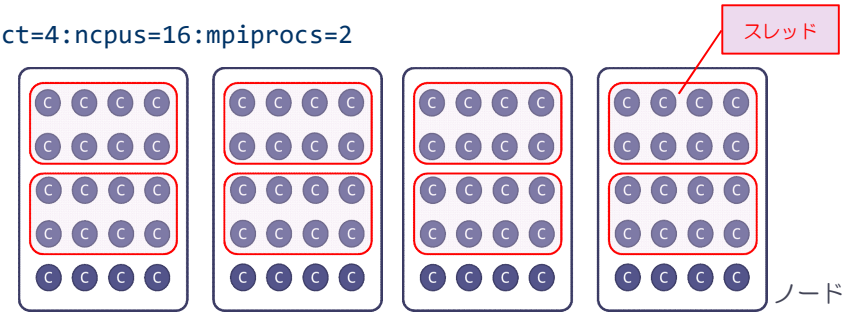
```
#!/bin/bash

#PBS -N hybrid                                  ジョブ名の指定
#PBS -q S
#PBS -j oe
#PBS -l select=4:ncpus=16:mpiprocs=2            select:   計算ノード数
                                                ncpus :   1計算ノード内のコア数
source /etc/profile.d/modules.sh                mpiprocs: 1計算ノード内のMPIプロセス数
module load intel
module load mpt

export KMP_AFFINITY=disabled
export OMP_NUM_THREADS=8

cd ${PBS_O_WORKDIR}
mpiexec_mpt omplace -nt ${OMP_NUM_THREADS} ./a.out
```

```
#PBS -l select=N:ncpus=T:mpiprocs=M
```

- T： 1ノード（共有メモリ）のコア数（≦40）
- M： 1ノードのMPIプロセス数
- N： 全ノード数
- NM: 全MPIプロセス数

例えば，コア数を 16，全MPIプロセスを 8 とすると，以下のようなマッピングが考えられる．この場合，

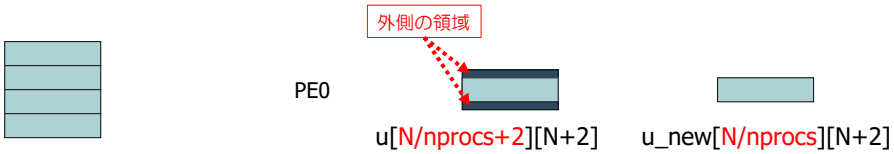
```
#PBS -l select=4:ncpus=16:mpiprocs=2
```






# 上級者になるために．．．

- 大規模問題を解く場合には，1プロセスのメモリ使用容量を少なくする必要が出てくる．
- 今回のプログラムでは，どのプロセスも u[N+2][N+2], u_new[N+2][N+2] として，計算全領域の変数を宣言したが，




各プロセスは，これだけの大きさをもてば良いはず．

- 実行前から並列数（nprocs）が固定されていれば，上記のように，プログラムの先頭で小さい領域を宣言することも出来るが，汎用のプログラムとしては，nprocs を実行開始時に決めたい．
- 配列を動的に確保する必要がある（C言語：malloc 関数、Fortran言語：allocate文）．
  - サンプルプログラム /home/guest60/share/alloc_heat2d.c もしくは alloc_heat3d.f90