講義2＆演習1
プログラム並列化と性能評価

神戸大学大学院システム情報学研究科
横川 三津夫
yokokawa@port.kobe-u.ac.jp

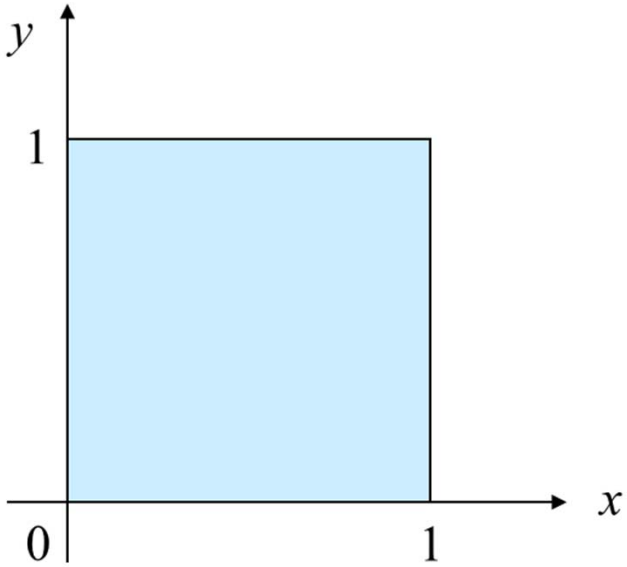
2次元温度分布の計算（定常問題）

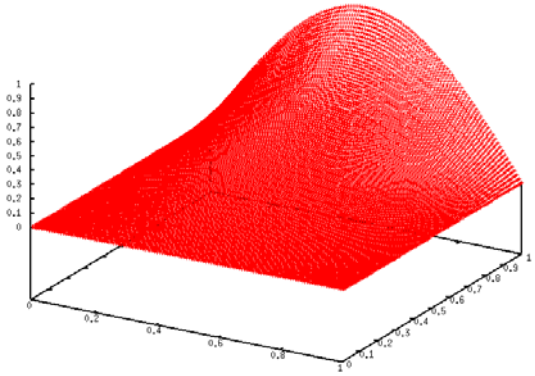
- 問題
  - 2次元正方形領域 [0,1]×[0,1] での熱伝導問題

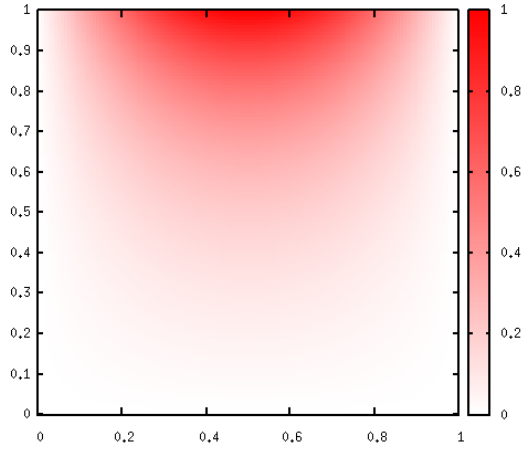
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

  - 境界条件

$$u(0, y) = 0$$
$$u(1, y) = 0$$
$$u(x, 0) = 0$$
$$u(x, 1) = \sin(\pi x)$$







十分な時間が経った後での温度分布はどうなるか？

※ 演習用のプログラムファイルは，以下のディレクトリに置いてありますので，利用して下さい．

```
Fortran言語版 /tmp/school/ex1/heat2d.f90
c言語版       /tmp/school/ex1/heat2d.c
```

演習：ヤコビ法の並列化と性能評価

- OpenMPによる並列化
  - 弱スケーリング　⇒　実行時間計測
  - 強スケーリング　⇒　実行時間計測
- MPIによる並列化
  - mpi_sendrecv関数
  - 強スケーリング，弱スケーリング　⇒ 実行時間計測
- Hybrid（MPI＋OpenMP）による並列化
  - 強スケーリング，弱スケーリング　⇒ 実行時間計測

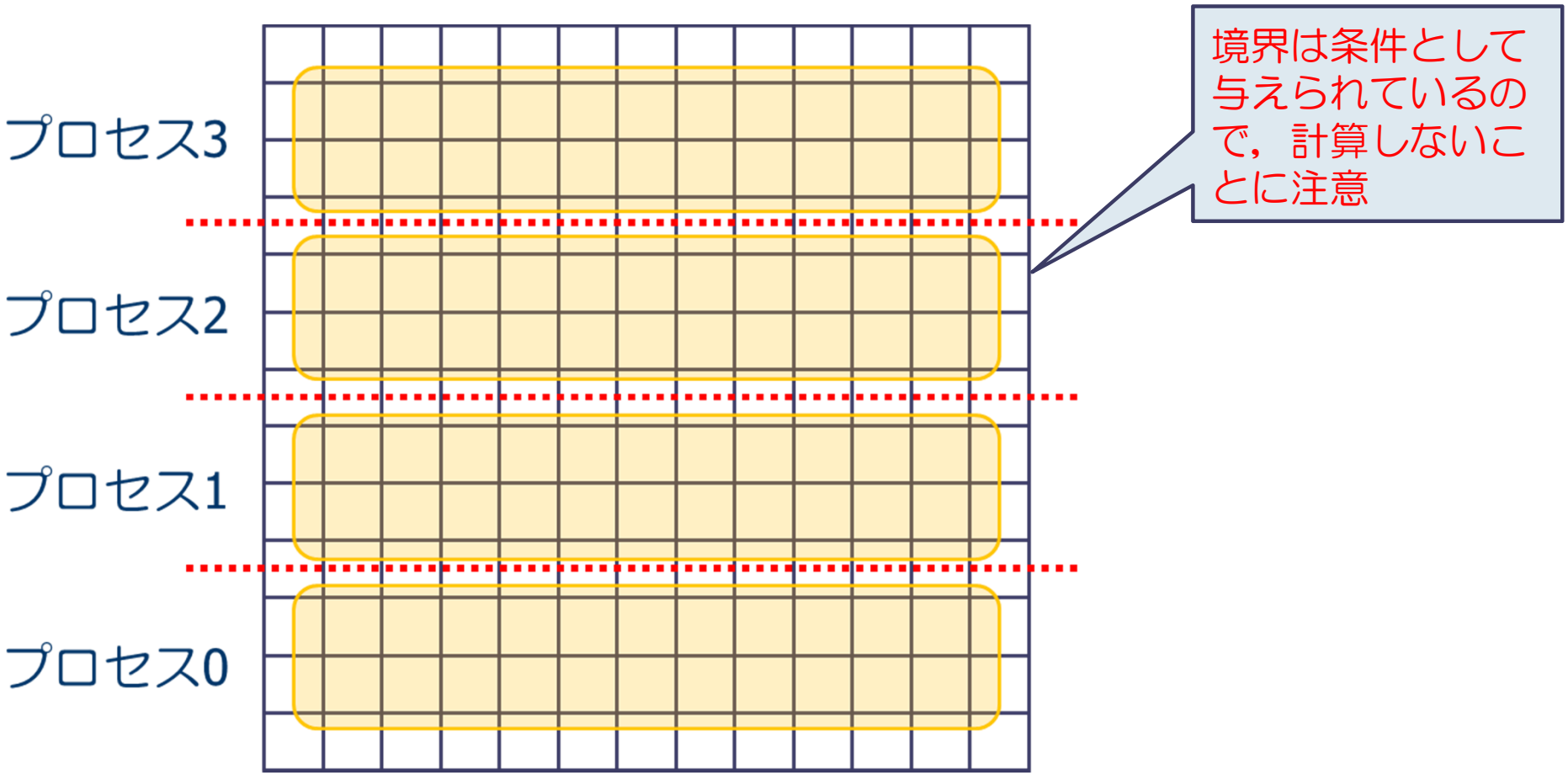
※ 強スケーリング並列化では反復回数が変わらないことを確認すること

OpenMPによる並列化

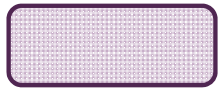
- 緩和ループ内部のループに，OpenMP指示行を挿入して並列化する.
  - !$omp parallel do，または #pragma omp parallel for
    - reduction変数があることに注意！！
- 緩和ループ部分の実行時間を計測する.
  - omp_get_wtime() 関数
- 強スケーリングの評価
  - 環境変数 OMP_NUM_THREADSを1，2，4，8，16と変化させて実行時間を計測し，実行時間が短縮されていることを確認する.
- 弱スケーリングの評価
  - 一つのスレッドの担当するデータの大きさを同じにして，スレッド数を1，2，4，8，16と変化させて実行時間を計測する.
    - 緩和ループの回数が変化することに注意！！

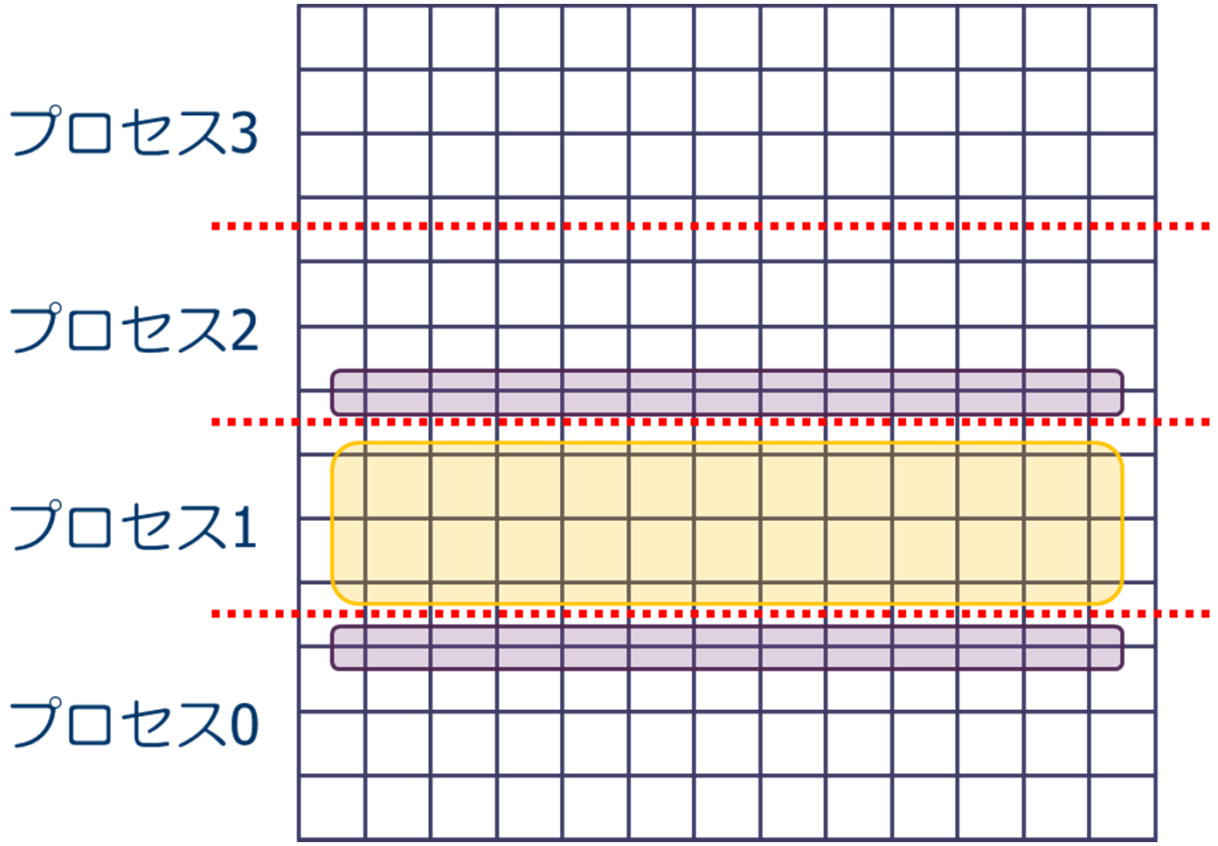
MPIによる並列化

- 領域分割によりMPI並列化（プロセス並列化）を行う.
  - 各プロセスは, [ ] 部分の計算を担当する.



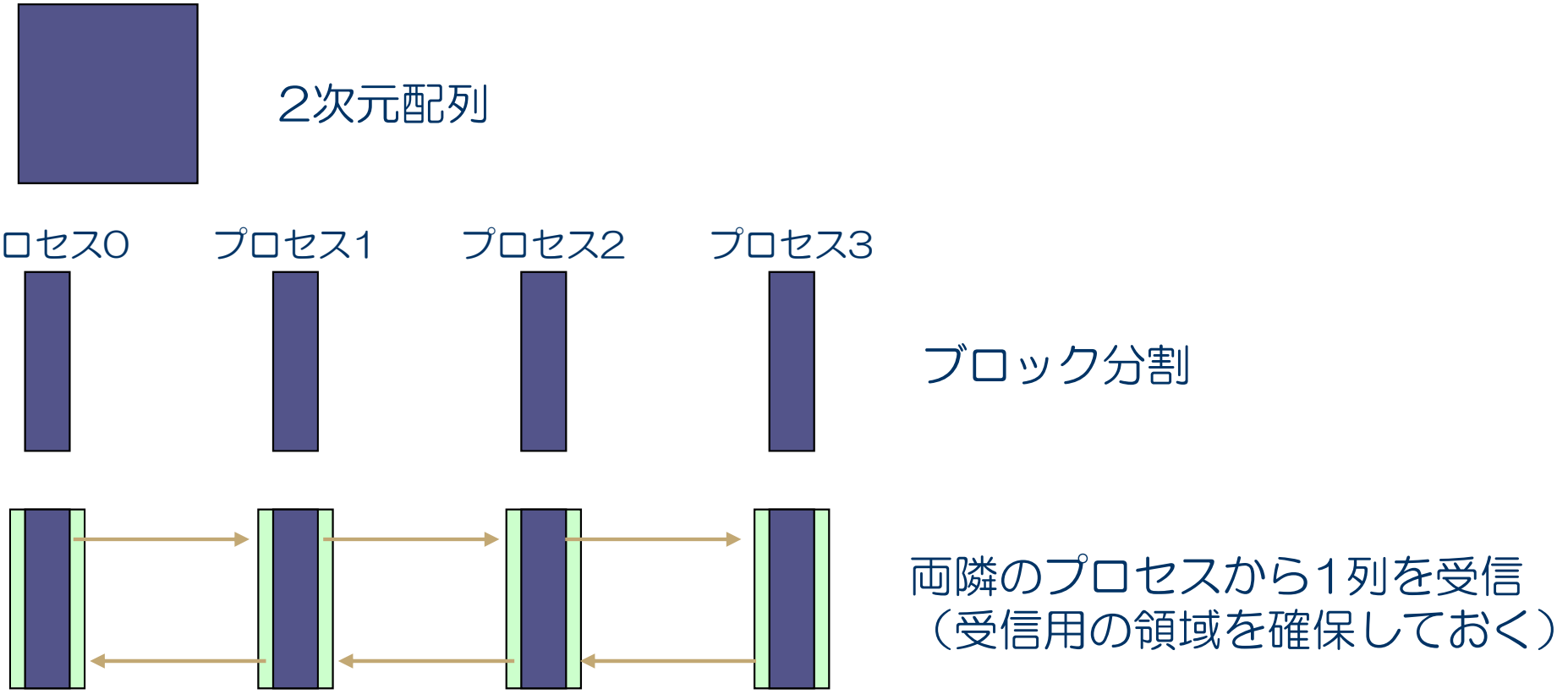
MPIによる並列化

- 各プロセスでの計算には，上下の部分の値  が必要であることに注意．
  - 上下部分の値は，mpi_sendrecv関数で転送する．



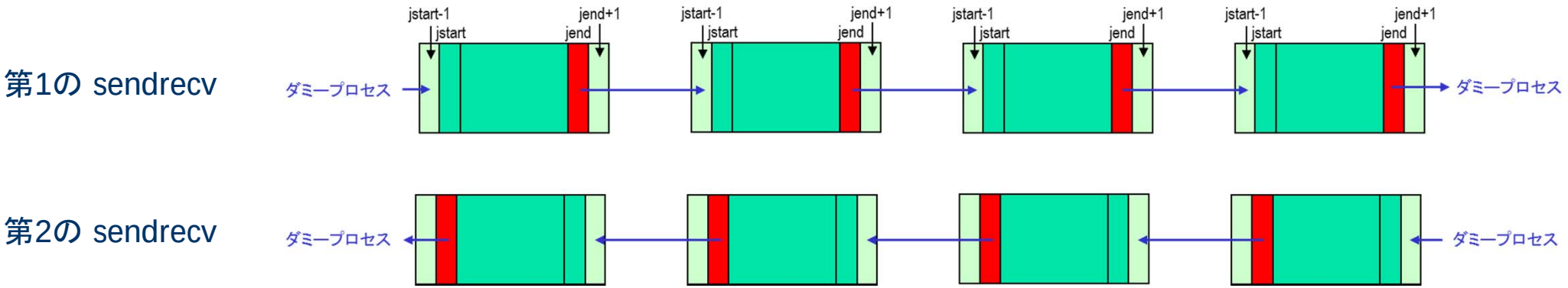
mpi_sendrecv関数の使い方（1/3）

- 2次元配列がブロック分割されているとする.
- このとき，自分の要素に隣接する隣プロセスの要素を持ってくるようなプログラムを作る.



mpi_sendrecv関数の使い方（2/3）

- 配列の確保
  - 自プロセスの担当範囲は jstart ～ jend 列
  - 受信領域を考慮し，jstart-1 ～ jend+1 列の領域を確保
- mpi_sendrecv による送受信
  - まず，上隣に jend 列を送り，下隣から jstart-1 列に受信
  - 次に，下隣に jstart 列を送り，上隣から jend+1 列に受信
  - 両端のプロセスは，ダミープロセス（MPI_PROC_NULL) と送受信するようにする.
    - MPI_sendrecvで同じように記述できる.



mpi_sendrecv関数の使い方（3/3）

上下のプロセスのプロセス番号を計算
（存在しない場合は MPI_PROC_NULL とする）

```
bottom = myrank-1
if( myrank == 0) bottom = MPI_PROC_NULL
top = myrank+1
if( myrank == nprocs-1) top = MPI_PROC_NULL
```

```
  call mpi_sendrecv( sendbuf, count, MPI_type, bottom, tag,  &
&                    redvbuf, count, MPI_type, top,    tag,  &
&                    MPI_COMM_WORLD, istat, ierr)

  call mpi_sendrecv( sendbuf, count, MPI_type, top,    tag,  &
&                    redvbuf, count, MPI_type, bottom, tag,  &
&                    MPI_COMM_WORLD, istat, ierr)
```

【Fortran】 mpi_sendrecv関数

```
mpi_sendrecv( sendbuf, sendcount, sendtype, dest,  sendtag, &
              recvbuf, recvcount, recvtype, source, recvtag, &
              comm, status, ierr )
```

- sendbuf:　送信するデータのための変数名（先頭アドレス）
- sendcount:　送信するデータの数（整数型）
- sendtype:　送信するデータの型
  - MPI_INTEGER, MPI_REAL8, MPI_CHARACTER など
- dest:　送信する相手のプロセス番号
- sendtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- recvbuf:　受信するデータのための変数名（先頭アドレス）
- recvcount:　受信するデータの数（整数型）
- recvtype:　受信するデータの型
- source:　送信してくる相手のプロセス番号
- recvtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- comm:　コミュニケータ（例えば，MPI_COMM_WORLD）
- status:　受信の状態を格納するサイズMPI_STATUS_SIZEの配列（整数型）
- ierr:　戻りコード（整数型）

【c】 MPI_Sendrecv関数

```
int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 int dest, int sendtag, void *recvbuf, int recvcount,
                 MPI_Datatype recvtype, int source, int recvtag, MPI_Comm
                 comm, MPI_Status *status)
```

- sendbuf:　送信するデータのための変数名（先頭アドレス）
- sendcount:　送信するデータの数（整数型）
- sendtype:　送信するデータの型
  - MPI_INTEGER, MPI_REAL8, MPI_CHARACTER など
- dest:　送信する相手のプロセス番号
- sendtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- recvbuf:　受信するデータのための変数名（先頭アドレス）
- recvcount:　受信するデータの数（整数型）
- recvtype:　受信するデータの型
- source:　送信してくる相手のプロセス番号
- recvtag:　メッセージ識別番号．送られて来たデータを区別するための番号
- comm:　コミュニケータ（例えば，MPI_COMM_WORLD）
- status:　受信の状態を格納するサイズMPI_STATUS_SIZEの配列（整数型）

MPIによる並列化（続き）

- 強スケーリングの評価
  - プロセス数 2，4，8 (，16)と変化させて，実行時間を計測し，実行時間が短縮されていることを確認する．
  - mpi_wtime() 関数
- 弱スケーリングの評価
  - 一つのプロセスが担当するデータサイズを同じにして，プロセス数を 2，4，8（，16）と変化させて実行時間を計測する．
    - 緩和ループの回数が変化することに注意！！

Hybrid（MPI+OpenMP）による並列化

- MPI化したプログラムに対し，緩和ループ内部のループにOpenMP指示行を挿入して並列化する.
- OpenMPの評価と同じことをする.