コミュニケータとデータタイプ

辻田　祐一
(RIKEN AICS)

講義・演習内容

- MPIにおける重要な概念
  - コミュニケータ
  - データタイプ
- MPI-IO
  - 集団型I/O
  - MPI-IOの演習

コミュニケータ

- MPIにおけるプロセスの“集団”
- 集団的な操作などにおける操作対象となる。

- MPIにおける集団的な操作とは？
  - 集団型通信(Collective Communication)
  - 集団型I/O(Collective I/O Operation)
  - Window操作（For One-Sided Communication)

MPI_Comm_rank

C: int MPI_Comm_rank(MPI_Comm comm, int *rank);
F: MPI_COMM_RANK(communicator, rank, ierr)

- 指定されたコミュニケータにおける自プロセスのランク番号を取得

```
  ...
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
...
```

プログラム例(C)

- 最初の引数 = “コミュニケータ”
  - MPI_COMM_WORLDあるいはユーザ独自に生成したコミュニケータを指定

MPI_Comm_size

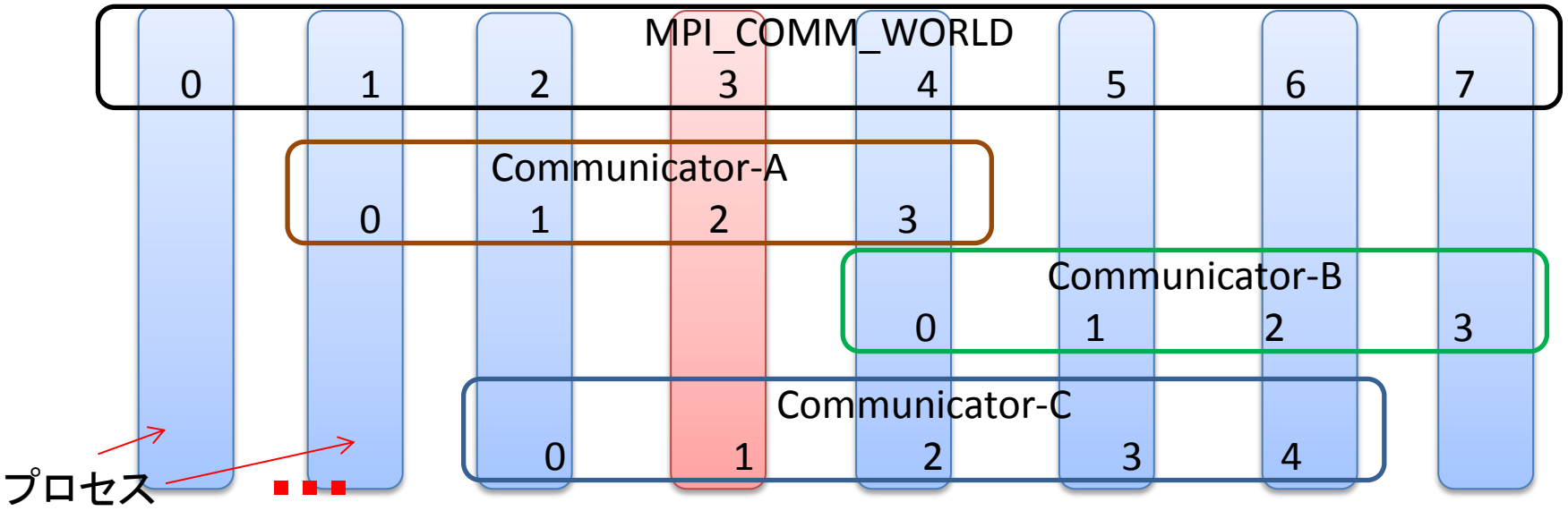
C: int MPI_Comm_size(MPI_Comm comm, int *size);
F: MPI_COMM_SIZE(communicator, size, ierr)

- 与えられたコミュニケータに属するランク数を取得

```
...
MPI_Comm_size(MPI_COMM_WORLD, &procs);
...
```

プログラム例（C）

コミュニケータの例



MPI_COMM_WORLD:　プロセス全てを含む最初に作られる基本のコミュニケータ

コミュニケータ操作関数を使って自由にコミュニケータを生成可能
（上の例だと、Communicator-A, B, C）

- 赤いプロセスでMPI_Comm_rankおよびMPI_Comm_sizeを実行すると…

コミュニケータ	MPI_Comm_rank()	MPI_Comm_size()
MPI_COMM_WORLD	3	8
Communicator-A	2	4
Communicator-B	----	----
Communicator-C	1	5

グループとコミュニケータ

- グループ：　プロセスの順序集団

- コミュニケータ
  - 互いに通信が可能なプロセス群のグループ
  - グループ内のプロセス群の情報や通信状態などを保持

  - MPIプログラム起動後に用意されているコミュニケータ
    - MPI_COMM_WORLD：　プロセス群全体
    - MPI_COMM_SELF：　　自プロセスのみ

コミュニケータに関する処理

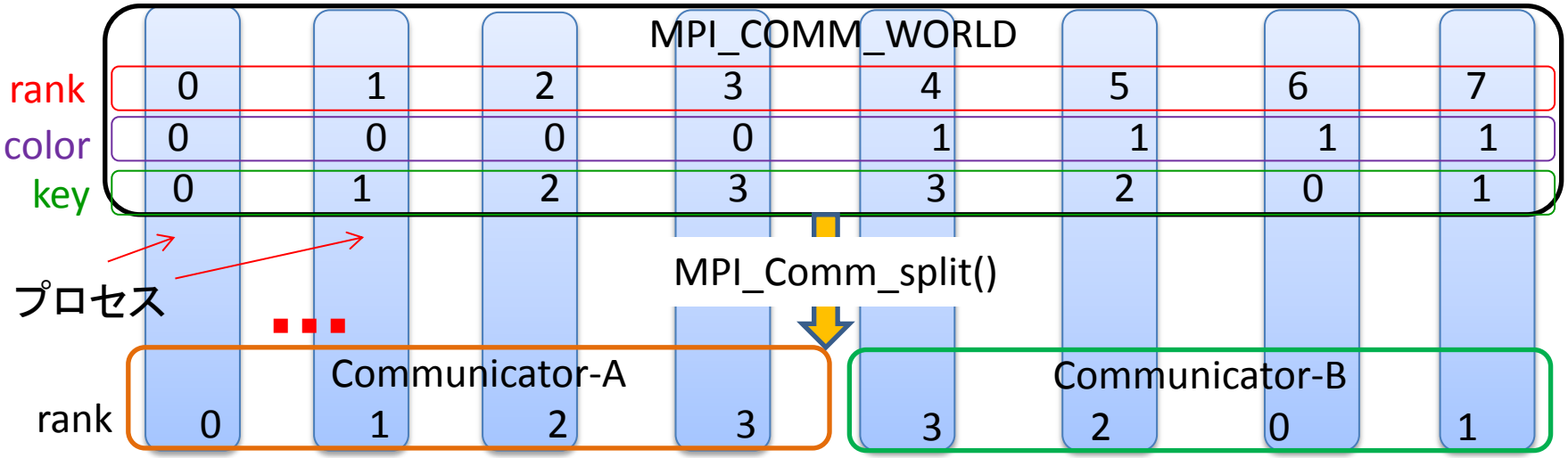
```
C:  int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *new_comm);
    int MPI_Comm_split(MPI_Comm comm, int color, int key,
                       MPI_Comm *new_comm);
    int MPI_Comm_free(MPI_Comm *comm);

F:  MPI_COMM_DUP(comm, new_comm, ierr)
    MPI_COMM_SPLIT(comm, color, key, new_comm, ierr)
    MPI_COMM_FREE(comm, ierr)
```

- MPI_Comm_dup(): コミュニケータの複製を生成
  - 親コミュニケータと同じプロセスグループで異なるコミュニケータを生成
  - 親コミュニケータと生成されたコミュニケータそれぞれで通信を分離できる。

コミュニケータの分割

- MPI_Comm_split()
  - コミュニケータを分割し新たなコミュニケータを生成
  - 同じcolorを持つプロセス群で新コミュニケータを生成
  - keyの値の昇順により同一コミュニケータ内のランク順が決まる。（keyが同じ場合、システム側で適当に決められる。）



コミュニケータの解放

- MPI_Comm_free()
  - 指定されたコミュニケータを解放
  - この関数の呼び出し以降は、当該コミュニケータは使用不能

- その他のコミュニケータ操作
  - グループ操作関数によって複数のグループを作り、そこから新たなコミュニケータを生成する方法もある。（今回は省略）

データ型

- MPIで扱うデータ型
  - 基本データ型
    - 整数型・文字型や実数型などの基本となるデータ型
  - 派生データ型
    - 基本データ型の組合せにより新たに生成されるデータ型
    - 派生データ型生成を行う関数により作成可能

基本データ型

＜C言語（代表的なものを抜粋）＞

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short
MPI_INT	signed int
MPI_LONG	signed long
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

＜FORTRAN（代表的なものを抜粋）＞

MPI datatype	FORTRAN datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

MPI datatype	C datatype	FORTRAN datatype
MPI_AINT	MPI_Aint	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_OFFSET	MPI_Offset	INTEGER (KIND-MPI_OFFSET_KIND)

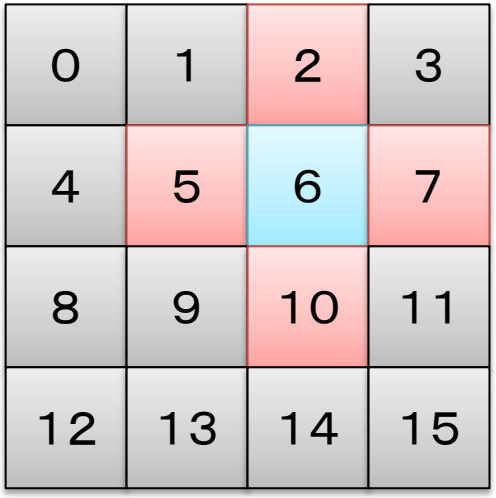
派生データ型

- 派生データ型（Derived Datatype）
  - 不連続なデータレイアウトをまとめたデータ型
  - 不連続なレイアウトにあるデータ群を1回の通信で処理が可能。高速化の対応を可能にする。
  - 基本データ型とオフセットの集合で表現される。
  - 派生データ型生成を行う関数により作成可能

２次元配列のブロック分割の例

- 多次元配列
  - ある一つの次元のみデータの並びが連続
  - それ以外の次元での並びは不連続

  - ２次元配列のブロック分割を考えてみる。



ステンシル計算における隣接ブロックとの通信
（赤色のブロックが通信対象）

不連続なデータの通信が発生

一つのデータ型（派生データ型）にして、１回で通信できるようにすることを考える。

派生データ型の生成と登録および解放

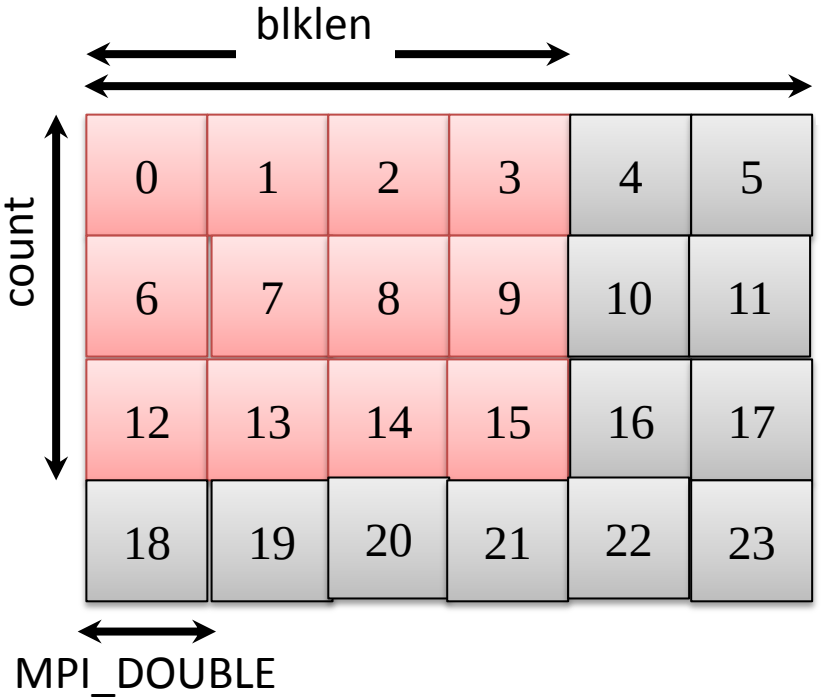
- 派生データ型の作成に使われる関数群（抜粋）
  - MPI_Type_contiguous
  - MPI_Type_vector
  - MPI_Type_indexed
  - MPI_Type_create_indexed
  - MPI_Type_create_subarray
  - MPI_Type_create_darray
- 作成した派生データ型の登録（データ型作成後に必ずこれを行わないといけない！）
  - MPI_Type_commit
- 作成したデータ型の解放（不要になった際に解放するために使う。）
  - MPI_Type_free

MPI_Type_vector

C: int MPI_Type_vector(int count, int blklen, int stride,
MPI_Datatype otype, MPI_Datatype *ntype);

F: MPI_TYPE_VECTOR(count, blklen, stride, otype, ntype, ierr)

＜赤色の領域をアクセスする派生データ型の作成例＞




```
MPI_Datatype ntype;

MPI_Type_vector(3, 4, 6, MPI_DOUBLE,
                &ntype);
```

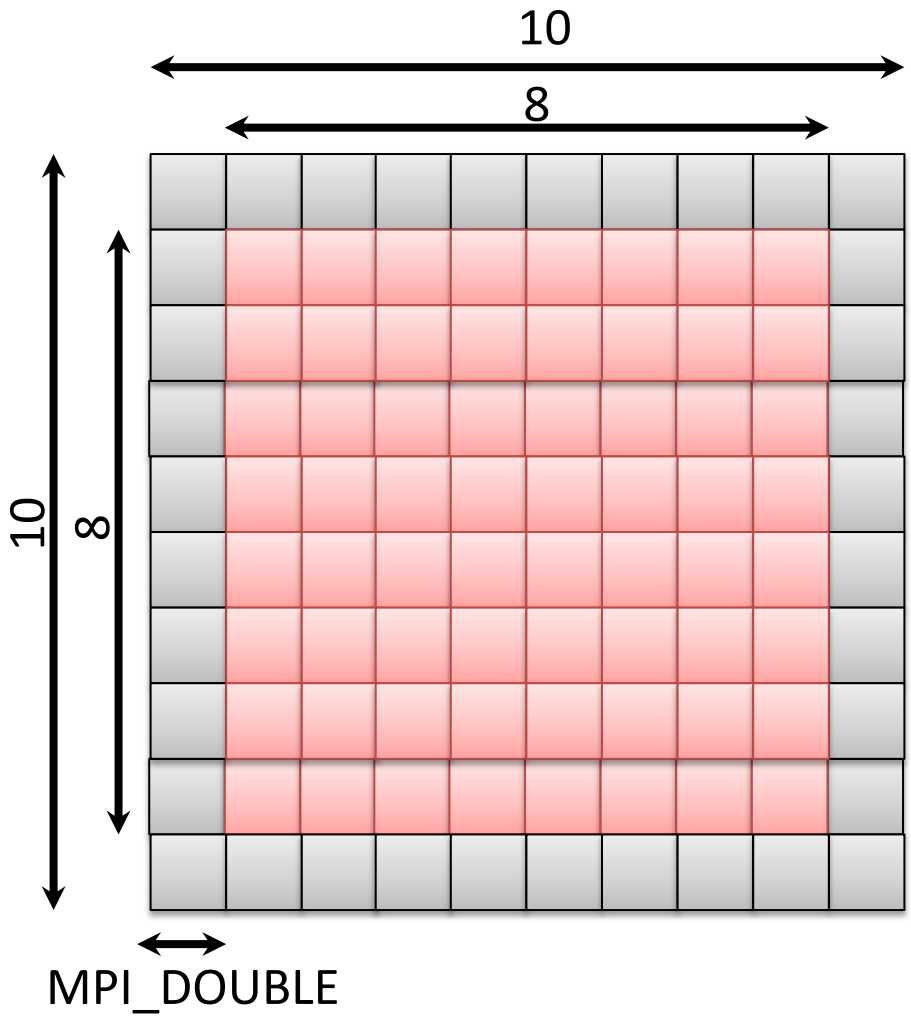
MPI_Type_create_subarray

C: int MPI_Type_create_subarray (int ndims, int array_of sizes[], int array_of_subsizes[], int array_of_starts[], int order, MPI_Datatype otype, MPI_Datatype *ntype);

F: MPI_TYPE_CREATE_SUBARRAY (ndims, array_of_sizes, array_of_subsizes, array_of_starts, order, otype, ntype, ierr)

- ndims: ベースになる配列の次元数
- array_of_sizes: ベースになる配列の各次元の大きさ
- array_of_subsizes: 部分配列の大きさ
- array_of_starts: 部分配列の開始地点（＊注意： FORTRANでも0から始まります。）
- order: MPI_ORDER_CまたはMPI_ORDER_FORTRAN

MPI_Type_create_subarray（続き）



赤い部分配列をアクセスする
派生データ型の作成例

```
int  o_sizes[2];
int  n_sizes[2];
int  starts[2];
MPI_Datatype ntype;

o_sizes[0] = o_sizes[1] = 10;
n_sizes[0] = n_sizes[1] = 8;
starts[0] = starts[1] = 1;

MPI_Type_create_subarray(2, o_sizes,
   n_sizes, starts, MPI_ORDER_C,
   MPI_DOUBLE, &ntype);
```

MPI_Type_commit/MPI_Type_free

```
C:   int MPI_Type_commit (MPI_Datatype type);

F:   MPI_TYPE_COMMIT (type, ierr)
```

- 作成したデータ型は必ずMPI_Type_commitにより登録する。
- 以後、このデータ型を用いた通信・I/Oなどが利用可能。

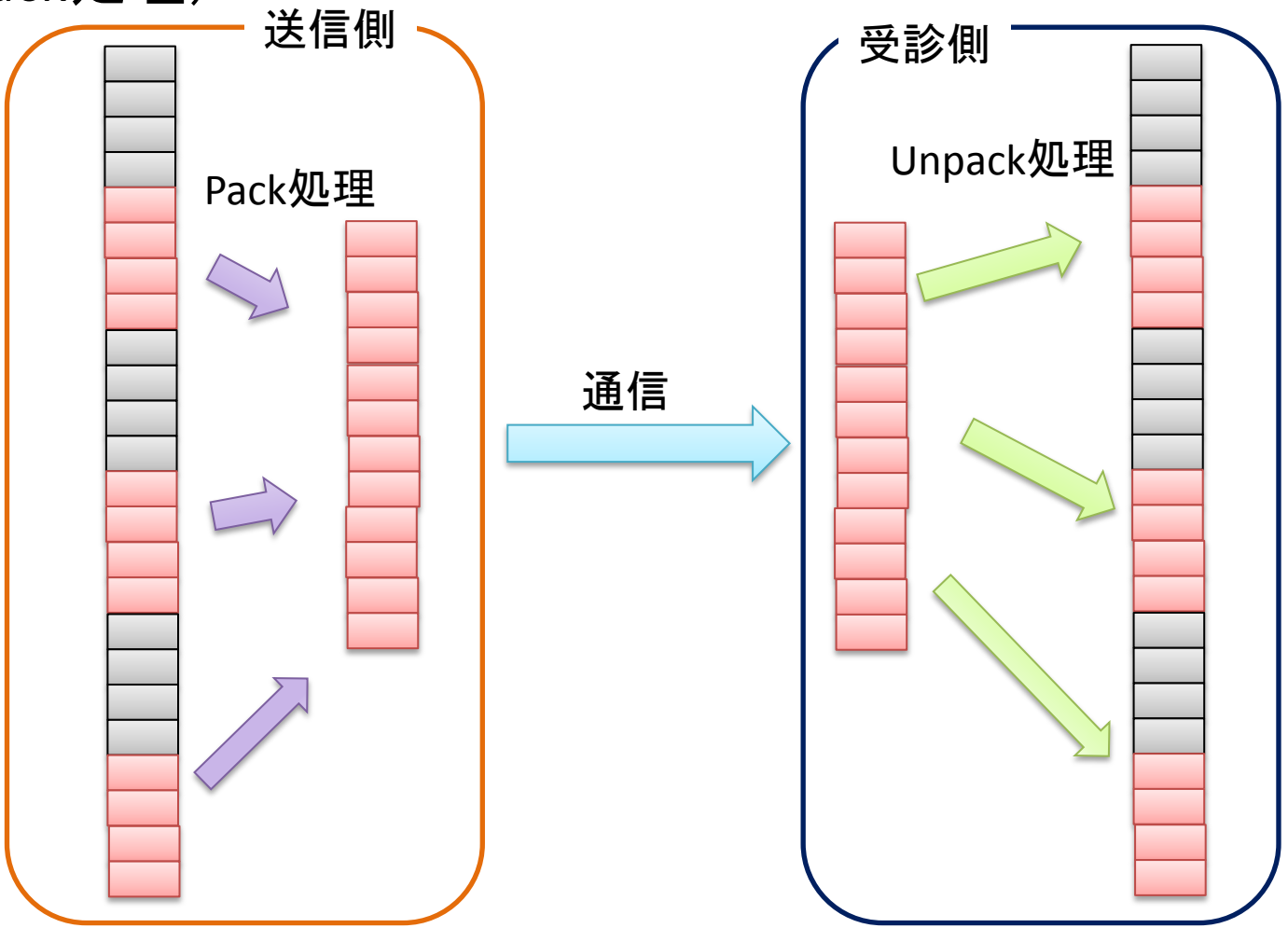
```
C:   int MPI_Type_free (MPI_Datatype *type);

F:   MPI_TYPE_FREE (type, ierr)
```

- 必要なくなったら作成したデータ型をMPI_Type_freeにより解放。
- 以後、このデータ型は使えなくなる。

派生データ型における内部処理

- 送信側： 不連続なデータを連続なデータに並び替え（Pack処理）した後に送信
- 受診側： 受け取ったデータを元の不連続なデータに並べ替え（Unpack処理）



派生データ型のまとめ

- 基本データ型から派生データ型を作成
- 派生データ型作成後に必ずMPI_Type_commitで登録
- 作成された派生データ型はMPIでの通信・I/Oで利用可能
- 不連続なデータを一纏めに扱うことにより、性能向上の可能性あり。
- 不要になった派生データ型はMPI_Type_freeで解放する。