Derived datatypes

Introduction

- In communications, exchanged data have different datatypes : MPI_INTEGER, MPI_REAL, MPI_COMPLEX, etc.
- We can create more complex data structures by using subroutines such as
- MPI_TYPE_CONTIGUOUS(), MPI_TYPE_VECTOR(), MPI_TYPE_INDEXED() or MPI_TYPE_CREATE_STRUCT()
- Derived datatypes allow exchanging non-contiguous or non-homogenous data in the memory and limiting the number of calls to communications subroutines

Derived datatypes

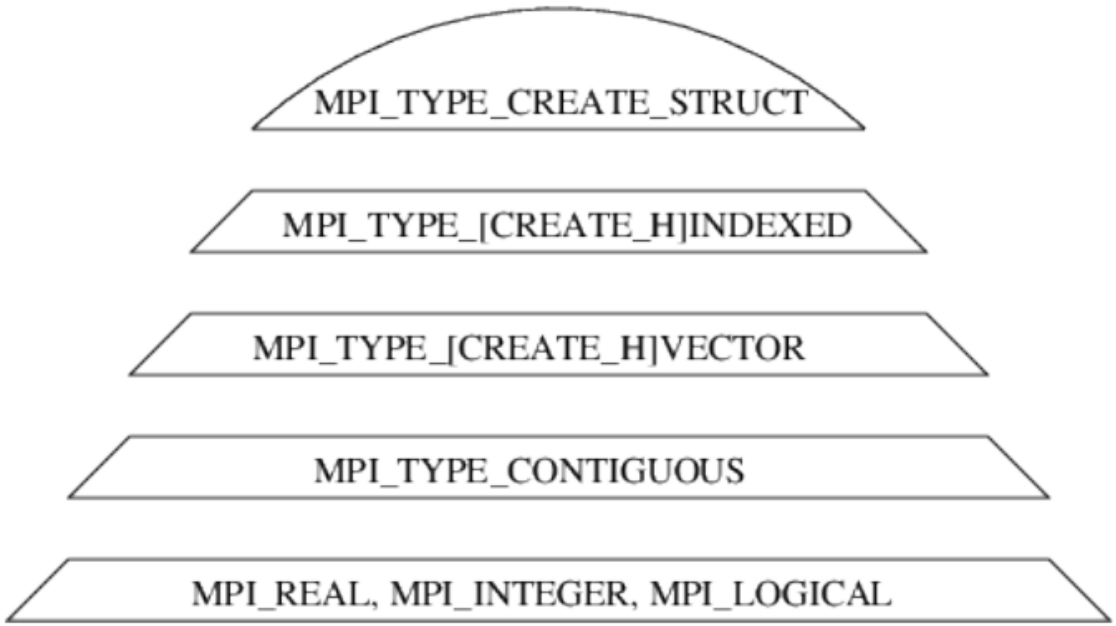

Figure: Hierarchy of the MPI constructors

Derived datatypes

Contiguous datatypes : MPI_TYPE_CONTIGUOUS()

- MPI_TYPE_CONTIGUOUS() creates a data structure from a homogenous set of existing datatypes contiguous in memory.

```
1 MPI_TYPE_CONTIGUOUS(count,old_type,new_type,code)
2
3 integer, intent(in) :: count, old_type
4 integer, intent(out) :: new_type,code
```

```
1 Datatype.Create_contiguous(self, int count)
```

Derived datatypes

MPI_TYPE_COMMIT and MPI_TYPE_FREE()

- Before using a new derived datatype, it is necessary to validate it with the MPI_TYPE_COMMIT() subroutine.

```
MPI_TYPE_COMMIT(new_type,code)

integer, intent(inout) :: new_type
integer, intent(out) :: code
```

```
new_type.Commit()
```

- The freeing of a derived datatype is made by using the MPI_TYPE_FREE() subroutine.

```
MPI_TYPE_FREE(new_type,code)
integer, intent(inout) :: new_type
integer, intent(out) :: code
```

```
new_type.Free()
```

Derived datatypes

Contiguous datatypes : MPI_TYPE_CONTIGUOUS()

- Example :

```
1  count = 5
2  size = 10
3
4  type_ligne = MPI.DOUBLE.Create_contiguous(count)
5  type_ligne.Commit()
6
7  if rank==0:
8      data = np.array([i for i in range(size)], dtype=np.float64)
9      comm.Send([data,1,type_ligne],dest=1)
10     print("Original data",data, rank)
11
12 elif rank == 1:
13     data = -1*np.ones(size, dtype=np.float64)
14     comm.Recv([data,1,type_ligne],source=0)
15     print("Received data",data, rank)
16
17 type_ligne.Free()
```

```
mpirun -n 2 python3 create_contiguous.py
Original data [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.] 0
Received data [ 0.  1.  2.  3.  4. -1. -1. -1. -1. -1.] 1
```

Derived datatypes

Constant stride : MPI_TYPE_VECTOR()

- MPI_TYPE_VECTOR() creates a data structure from a homogenous set of existing datatypes separated by a constant stride in memory. The stride is given in number of elements.

```
1 MPI_TYPE_VECTOR(count,block_length,stride,old_type,new_type,code)
2
3 integer, intent(in) :: count,block_length
4 integer, intent(in) :: stride
5 integer, intent(in) :: old_type
6
7 integer, intent(out) :: new_type,code
```

```
1 Datatype.Create_vector(self, int count, int blocklength, int stride)
```

Derived datatypes

Constant stride : MPI_TYPE_VECTOR()

- Example :

```
1  stride = 2
2  count = 5
3  blocklen = 1
4  size = 10
5
6  type_colum = MPI.DOUBLE.Create_vector(count,blocklen,stride)
7  type_colum.Commit()
8
9  if rank==0:
10     data = np.array([i for i in range(size)], dtype=np.float64)
11     comm.Send([data,1,type_colum],dest=1)
12     print("Original data",data, rank)
13 elif rank==1:
14     data = -1*np.ones(size, dtype=np.float64)
15     comm.Recv([data,1,type_colum],source=0)
16     print("Received data",data, rank)
17
18 type_colum.Free()
```

```
mpirun -n 2 python3 create_vector.py
Original data [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.] 0
Received data [ 0. -1.  2. -1.  4. -1.  6. -1.  8. -1.] 1
```

Derived datatypes

Constant stride : MPI_TYPE_CREATE_HVECTOR()

- MPI_TYPE_CREATE_HVECTOR() creates a data structure from a homogenous set of existing datatype separated by a constant stride in memory. The stride is given in bytes.
- This call is useful when the old type is no longer a base datatype (MPI_INTEGER, MPI_REAL,...) but a more complex datatype constructed by using MPI subroutines, because in this case the stride can no longer be given in number of elements.

```
1 MPI_TYPE_CREATE_HVECTOR(count,block_length,stride,old_type,new_type,code)
2
3 integer, intent(in) :: count,block_length
4 integer(kind=MPI_ADDRESS_KIND), intent(in) :: stride
5 integer, intent(in) :: old_type
6
7 integer, intent(out) :: new_type, code
```

```
1 Datatype.Create_hvector(self, int count, int blocklength, Aint stride)
```

Derived datatypes

Homogenous datatypes of variable strides

- MPI_TYPE_INDEXED() allows creating a data structure composed of a sequence of blocks containing a variable number of elements separated by a variable stride in memory. The stride is given in number of elements.
- MPI_TYPE_CREATE_HINDEXED() has the same functionality as MPI_TYPE_INDEXED() except that the strides separating two data blocks are given in bytes. This subroutine is useful when the old datatype is not an MPI base datatype(MPI_INTEGER, MPI_REAL, ...). We cannot therefore give the stride in number of elements of the old datatype.
- For MPI_TYPE_CREATE_HINDEXED(), as for MPI_TYPE_CREATE_HVECTOR(), use MPI_TYPE_SIZE() or MPI_TYPE_GET_EXTENT() in order to obtain in a portable way the size of the stride in bytes.

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_INDEXED()

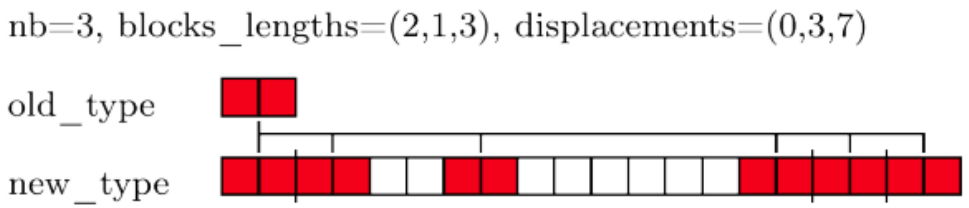

Figure: The MPI_TYPE_INDEXED constructor

```
1 MPI_TYPE_INDEXED(nb,block_lengths,displacements,old_type,new_type,code)
2
3 integer,intent(in) :: nb
4 integer,intent(in),dimension(nb) :: block_lengths
5 integer,intent(in),dimension(nb) :: displacements
6 integer,intent(in) :: old_type
7
8 integer,intent(out) :: new_type,code
```

```
1 Datatype.Create_indexed(self, blocklengths, displacements)
```

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_INDEXED()

- Example :

```
1  size = 10
2  count = 3
3
4  counts = [2, 1, 3]
5  displacements = [0, 3, 7]
6
7  indexedtype = MPI.INT64_T.Create_indexed(counts, displacements)
8  indexedtype.Commit()
9
10 if rank==0:
11     data = np.array([i for i in range(size)], dtype=np.float64)
12     comm.Send([data,1,indexedtype],dest=1)
13     print("Original data",data, rank)
14 elif rank==1:
15     data = -1*np.ones(size, dtype=np.float64)
16     comm.Recv([data,1,indexedtype],source=0)
17     print("Received data",data, rank)
18
19 indexedtype.Free()
```

```
mpirun -n 2 python3 create_indexed.py
Original data [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.] 0
Received data [ 0.  1. -1.  3. -1. -1. -1.  7.  8.  9.] 1
```

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_CREATE_HINDEXED()

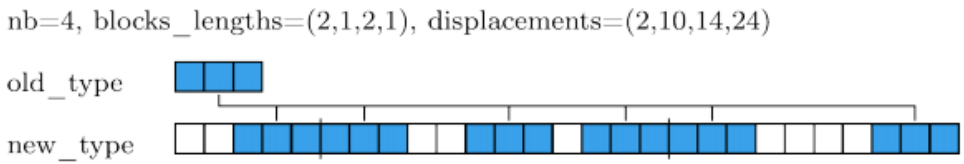

Figure: The MPI_TYPE_CREATE_HINDEXED constructor

```
1 MPI_TYPE_CREATE_HINDEXED(nb, block_lengths,displacements, old_type,new_type,code)
2
3 integer,intent(in) :: nb
4 integer,intent(in),dimension(nb) :: block_lengths
5 integer(kind=MPI_ADDRESS_KIND),intent(in),dimension(nb) :: displacements
6 integer,intent(in) :: old_type
7
8 integer,intent(out) :: new_type,code
```

```
1 Datatype.Create_hindexed(self, blocklengths, displacements)
```

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_INDEXED()

- Example : triangular matrix

In the following example, each of the two processes :

1. Initializes its matrix (positive growing numbers on process 0 and negative decreasing numbers on process 1).
2. Constructs its datatype : triangular matrix (superior for the process 0 and inferior for the process 1).
3. Sends its triangular matrix to the other process and receives back a triangular matrix which it stores in the same place which was occupied by the sent matrix.
4. Frees its resources.

Derived datatypes

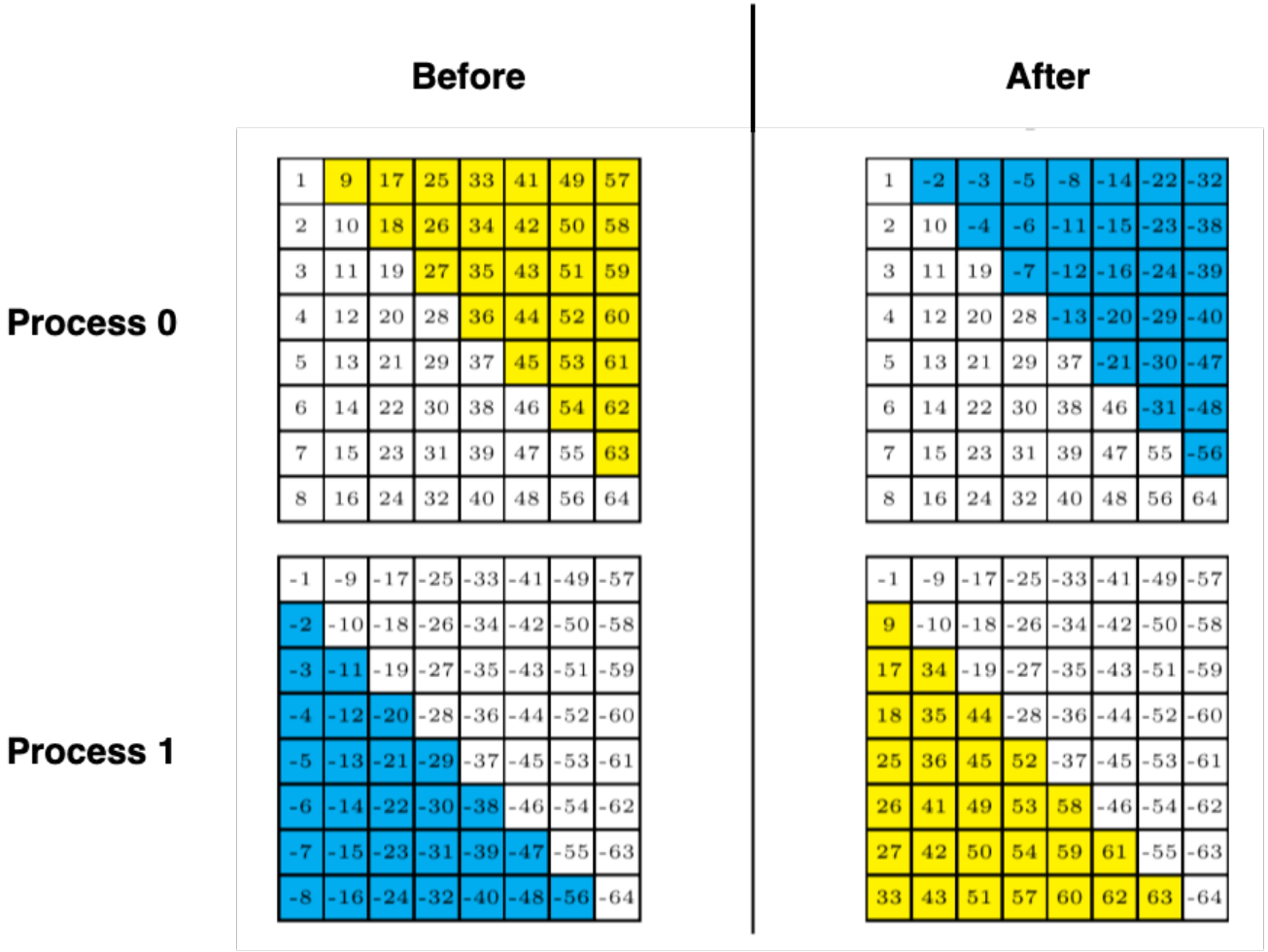

Figure: – Exchange between the two processes

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_INDEXED()

- Example :

```
1  from mpi4py import MPI
2  import numpy as np
3  comm = MPI.COMM_WORLD
4  nb_procs = comm.Get_size()
5  rank = comm.Get_rank()
6  n = 8; sign = -1
7  if rank == 0: sign = 1
8
9  a = [sign*i for i in range(1,n*n+1,1)]
10 Matrix = np.array(a)
11 Matrix = np.reshape(Matrix, (n,n)).transpose()
12
13 if rank == 0:
14     displacements = [n*i for i in range(n)]
15     block_lengths = [i for i in range(n)]
16 else:
17     displacements = [n*i+i+1 for i in range(n)]
18     block_lengths = [n-i-1 for i in range(n)]
19
20 type_triangle = MPI.DOUBLE.Create_indexed(block_lengths, displacements)
21 type_triangle.Commit()
22
23 num_proc = (rank+1)%2
24 comm.Send([Matrix,1,type_triangle],dest=num_proc)
25 comm.Recv([Matrix,1,type_triangle],source=num_proc)
26
27 type_triangle.Free()
28 print(Matrix, rank)
```

Derived datatypes

Homogenous datatypes of variable strides : MPI_TYPE_INDEXED()

- Example : Matrix after permutation

```
mpirun -n 2 python3 matrixExchange.py
[[  1  -2  -3  -5  -8 -14 -22 -32]
[  2  10  -4  -6 -11 -15 -23 -38]
[  3  11  19  -7 -12 -16 -24 -39]
[  4  12  20  28 -13 -20 -29 -40]
[  5  13  21  29  37 -21 -30 -47]
[  6  14  22  30  38  46 -31 -48]
[  7  15  23  31  39  47  55 -56]
[  8  16  24  32  40  48  56  64]] 0

[[ -1  -9 -17 -25 -33 -41 -49 -57]
[  9 -10 -18 -26 -34 -42 -50 -58]
[ 17  34 -19 -27 -35 -43 -51 -59]
[ 18  35  44 -28 -36 -44 -52 -60]
[ 25  36  45  52 -37 -45 -53 -61]
[ 26  41  49  53  58 -46 -54 -62]
[ 27  42  50  54  59  61 -55 -63]
[ 33  43  51  57  60  62  63 -64]] 1
```