24th Summer School on PARALLEL COMPUTING

MPI Derived Data Types

Gabriele Fatigati – g.fatigati@cineca.it

SuperComputing Applications and Innovation Department

Derived Data Types

- What are they?
  - Data types built from the basic MPI datatypes. Formally, the MPI Standard defines a general datatype as an object that specifies two things:
  - a sequence of basic datatypes
  - a sequence of integer (byte) displacements

Derived Data Types

- Why use them?
  - Sometimes more convenient and efficient. For example, you may need to send messages that contain
    1. non-contiguous data of a single type (e.g. a sub-block of a matrix)
    2. contiguous data of mixed types (e.g., an integer count, followed by a sequence of real numbers)
    3. non-contiguous data of mixed types.
- As well as improving program readability and portability they may improve performance.

How to use

1. Construct the datatype using a template or *constructor*.
2. Allocate the datatype.
3. Use the datatype.
4. Deallocate the datatype.

You must construct and allocate a datatype before using it. You are not required to use it or deallocate it, but it is recommended (there may be a limit).

Datatype constructors

- MPI_Type_contiguous
  - Simplest constructor. Makes count copies of an existing datatype
- MPI_Type_vector, MPI_Type_hvector
  - Like contiguous, but allows for regular gaps (stride) in the displacements. For MPI_Type_hvector the stride is specified in bytes.

Allocating/deallocating and using datatypes

Allocate and deallocate

- C

```
int MPI_Type_commit (MPI_datatype *datatype)
int MPI_Type_free (MPI_datatype *datatype)
```

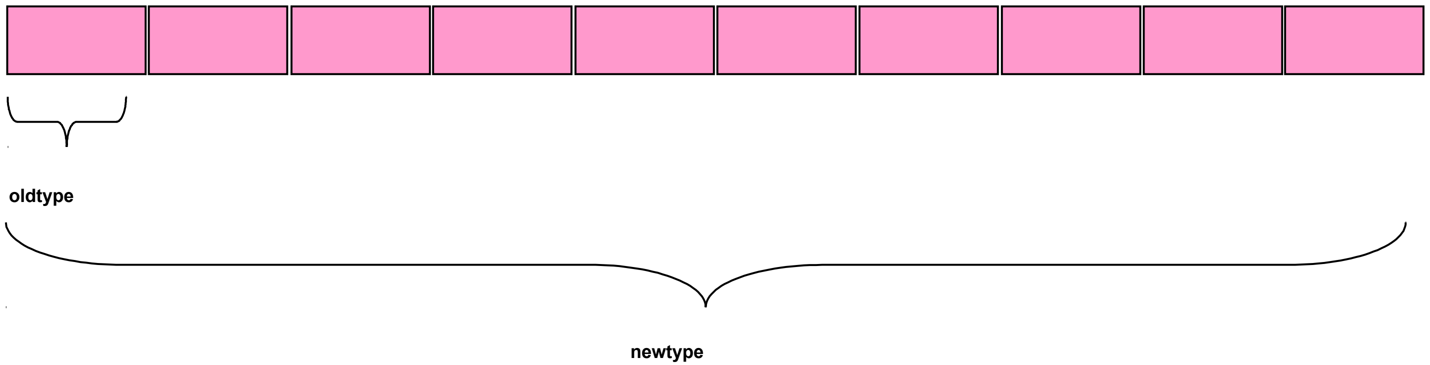
- FORTRAN

```
INTEGER DATATYPE, MPIERROR
MPI_TYPE_COMMIT(DATATYPE, MPIERROR)
MPI_TYPE_FREE(DATATYPE, MPIERROR)
```

MPI_TYPE_CONTIGUOUS

```
MPI_TYPE_CONTIGUOUS (count, oldtype, newtype)
            IN count: replication count (non-negative integer)
            IN oldtype: old datatype (handle)
            OUT newtype: new datatype (handle)
```

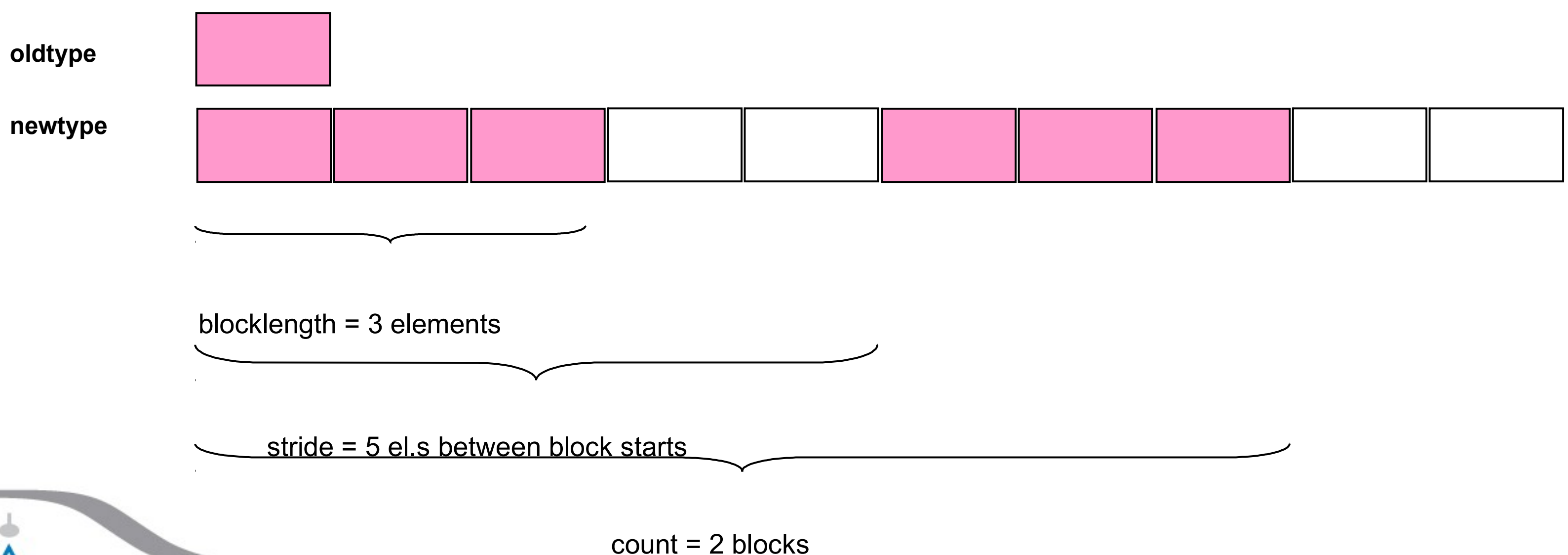
- MPI_TYPE_CONTIGUOUS constructs a typemap consisting of the **replication** of a **datatype** into contiguous locations.
- newtype is the datatype obtained by concatenating count copies of oldtype.



MPI_TYPE_VECTOR

```
MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype)
        IN count: Number of blocks (non-negative integer)
        IN blocklen: Number of elements in each block
                                    (non-negative integer)
        IN stride: Number of elements (NOT bytes) between start of
                             each block (integer)
        IN oldtype: Old datatype (handle)
        OUT newtype: New datatype (handle)
```

- Consists of a number of elements of the same datatype repeated with a certain stride



Example 1 - A rowtype

```
count = 4;
MPI_Type_contiguous(count, MPI_FLOAT, &rowtype);
```

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[2][0], 1, rowtype, dest, tag, comm);
```

9.0	10.0	11.0	12.0

1 element of rowtype

Example 2 - columntype

```
count = 4;   blocklength = 1;   stride = 4;
MPI_Type_vector(count, blocklength, stride, MPI_FLOAT,
                &columntype);
```

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[0][1], 1, columntype, dest, tag, comm);
```

2.0	6.0	10.0	14.0

1 element of columntype

Other tools

- MPI_GET_COUNT, MPI_GET_ELEMENTS
  - Routines which return the number of "copies" of type datatype and the number of basic elements (often used after a MPI_RECV).

int MPI_Get_count(const MPI_Status *status, MPI_Datatype datatype, int *count)int MPI_Get_elements(const MPI_Status *status, MPI_Datatype datatype, int *count)

```
CALL MPI_TYPE_CONTIGUOUS(2, MPI_REAL, Type2, ierr)
CALL MPI_TYPE_COMMIT(Type2, ierr)
...
CALL MPI_COMM_RANK(comm, rank, ierr)
IF(rank.EQ.0) THEN
    CALL MPI_SEND(a, 2, MPI_REAL, 1, 0, comm, ierr)
    CALL MPI_SEND(a, 3, MPI_REAL, 1, 0, comm, ierr)
ELSE
    CALL MPI_RECV(a, 2, Type2, 0, 0, comm, stat, ierr)
    CALL MPI_GET_COUNT(stat, Type2, i, ierr)     ! returns i=1
    CALL MPI_GET_ELEMENTS(stat, Type2, i, ierr)  ! returns i=2
    CALL MPI_RECV(a, 2, Type2, 0, 0, comm, stat, ierr)
    CALL MPI_GET_COUNT(stat, Type2, i, ierr)     ! returns i=MPI_UNDEFINED //  because the number of basic elements received is not a multiple of n=2
    CALL MPI_GET_ELEMENTS(stat, Type2, i, ierr)  ! returns i=3
END IF
```

Derived Datatype Summary

- Provide a portable and elegant way of communicating non-contiguous or mixed types in a message.
- By optimising how data is stored, should improve efficiency during MPI send and receive
- Derived datatypes are built from basic MPI datatypes, according to a template. Can be used for many variables of the same form.
- Remember to commit the datatypes before using them.