



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Message Passing Interface (MPI)

Summer School 2017 – Effective High Performance Computing

Tim Robinson, CSCS

July 19–20, 2017

# Previous course summary

- Point-to-point communication, blocking and non-blocking
- Collective operations

# Course Objectives

- Construct and use MPI derived datatypes

# General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Datatypes

# General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Datatypes
  - Construct datatype
  - Contiguous datatype
  - Indexed datatype
  - Struct datatype



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# MPI derived datatypes

---

# Using MPI derived datatypes

MPI derived datatypes (differently from C or Pseudo) are created (and destroyed) at run-time through calls to MPI library routines.

Implementation steps:

1. Construct the datatype;
2. Allocate the datatype;
3. Use the datatype;
4. Deallocate the datatype.

# Construct a datatype

- `MPI_Type_contiguous`

Produces a new datatype by making count copies of an existing data type.

- `MPI_Type_vector`, `MPI_Type_create_hvector`

Similar to contiguous, but allows for regular gaps (stride) in the displacements. `MPI_Type_create_hvector` is identical to `MPI_Type_vector` except that stride is specified in bytes.

- `MPI_Type_indexed`, `MPI_Type_create_hindexed`

An array of displacements of the input data type is provided as the map for the new data type. `MPI_Type_create_hindexed` is identical to `MPI_Type_indexed` except that offsets are specified in bytes.

- `MPI_Type_create_struct`

The most general of all derived datatypes. The new data type is formed according to completely defined map of the component data types.



# Allocate and destroy the Datatype

A constructed datatype must be committed to the system before it can be used in a communication.

Pseudo-code

```
MPI_Type_commit(datatype)  
MPI_Type_free(datatype)
```

# Contiguous Datatype

`MPI_Type_contiguous` constructs a typemap consisting of the replication of a datatype into contiguous locations.

Pseudo-code

```
MPI_Type_contiguous(count, oldtype, newtype)
```

<b>count</b>	number of BLOCKs to be added
<b>oldtype</b>	oldtype Datatype of each element
<b>newtype</b>	new derived datatype

REMEMBER: BLOCK = contiguous elements of the same type.

## Contiguous Datatype: example

array  $a[][] =$

0.0	0.1	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

Create a new type of 4 floats representing a row in  $a$ .

```
MPI_Type_contiguous(4, MPI_FLOAT, &MyRowType)
```

Use the new type to send one row:

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

Data sent is:

0.8	0.9	0.10	0.11
-----	-----	------	------

# Contiguous Datatype with stride

`MPI_Type_contiguous` constructs a typemap consisting of the replication of a datatype into contiguous locations.

Pseudo-code

```
MPI_Type_vector(count, blocklength, stride,  
               oldtype, newtype)
```

<b>count</b>	number of BLOCKs to be added
<b>blocklength</b>	Number of elements in block
<b>stride</b>	Number of elements (NOT bytes) between start of each block
<b>oldtype</b>	oldtype Datatype of each element
<b>newtype</b>	new derived datatype

The Vector constructor is similar to contiguous, but allows for regular gaps or overlaps (stride) in the displacements.

# Contiguous Datatype with stride: example

array  $a[][] =$

0.0	0.1	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

Create a new type of 4 floats representing a col in  $a$ .

Pseudo-code

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

Use the new type to send one column:

Pseudo-code

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

Data sent is:

0.2	0.6	0.10	0.14
-----	-----	------	------

# Indexed Datatype

`MPI_Type_indexed` constructs a typemap consisting of the replication of a datatype from locations defined by an array of block lengths and an array of displacements.

Pseudo-code

```
MPI_Type_indexed(count, blocklength[], displacement[],  
                 oldtype, newtype)
```

<b>count</b>	number of BLOCKs to be added and number of elements in the following arrays
<b>blocklength</b>	number of instances of oldtype in each block
<b>displacement</b>	displacement of each block in units of extent (oldtype)
<b>oldtype</b>	oldtype Datatype of each element
<b>newtype</b>	new derived datatype

# Indexed Datatype: example

count = 3;

oldtype = MPI\_INT

blocklength= 

2	3	1
---	---	---

displacement= 

0	3	9
---	---	---

Selected blocks are: 

0	1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	---	----

# Struct Datatype

`MPI_Type_create_struct` constructs a typemap consisting of different datatype from locations defined by an array of block lengths and an array of displacements. Displacements are expressed in bytes (since the type can change!!!).

Pseudo-code

```
MPI_Type_create_struct(count, blocklength[], displacement[],  
                      oldtype[], newtype)
```

<b>count</b>	number of BLOCKs to be added and number of elements in the following arrays
<b>blocklength</b>	number of instances of oldtype in each block
<b>displacement</b>	displacement in BYTES of each block
<b>oldtype</b>	oldtype Datatype of each element
<b>newtype</b>	new derived datatype



## Struct Datatype: example

count = 3;

blocklength= 

2	2	1
---	---	---

displacement (in bytes)= 

0	12	36
---	----	----

oldtype= 

MPI_INT	MPI_DOUBLE	MPI_FLOAT
---------	------------	-----------

A block is 4 Bytes long.

Selected blocks are: 

0	1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	---	----

# Other functions

- Manage types:

`MPI_Type_dup` ...

- Getter for types:

`MPI_Type_get_extent`, `MPI_Type_size`,

`MPI_Type_get_contents` ...

# Practicals

## Exercise: 04.MPI\_Type

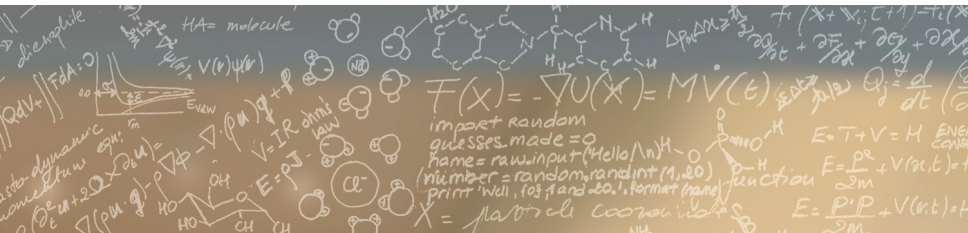
1. Create a derived datatype based on a struct



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



**Thank you for your attention.**