# Derived Data Types

The goals of this lesson are:
- understanding how to define new MPI data types

## 1. Derived Data Types in MPI

In the previous lessons we've dealt with primitive MPI data types. The most often used primitive MPI data types are:

```
MPI_CHAR
MPI_WCHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_LONG_LONG_INT
MPI_LONG_LONG
MPI_SIGNED_CHAR
MPI_UNSIGNED_CHAR
MPI_UNSIGNED_SHORT
MPI_UNSIGNED_LONG
MPI_UNSIGNED
MPI_FLOAT
MPI_DOUBLE
MPI_LONG_DOUBLE
```

By using sequences of the primitive types, MPI allows the definition of custom types. We will refer to these custom types as *derived types*.
Primitive types are contiguous. Derived data types allow us to handle non-contiguous data and treat it as if it was contiguous.
There are 4 categories of derived data types:
- Contiguous
- Vector
- Indexed
- Struct

MPI derived types will be instances of type MPI_Datatype

***Contiguous data types*** are represented as a contiguous sequence of values of the same MPI data type. A contiguous data type can be defined using MPI_Type_contiguous routine:

```
int MPI_Type_contiguous(int count,
                        MPI_Datatype old_type,
                        MPI_Datatype *new_type_p)
```

Parameters:
- count - replication count
- oldtype - old datatype
- newtype - new datatype

***Vector data types*** are similar to the contiguous data types. The main difference is that vector data types allow the use of a stride (gap) in the displacements.
Parameters

```
int MPI_Type_vector(int count,
                   int blocklength,
                   int stride,
                   MPI_Datatype old_type,
                   MPI_Datatype *newtype_p)
```

Parameters:
- count - number of blocks (nonnegative integer)
- blocklength - number of elements in each block (nonnegative integer)
- stride -number of elements between start of each block (integer)
- oldtype - old datatype (handle)
- newtype_p - new datatype (handle)

***Indexed data types*** use as map, two arrays: blocklens and indices. Data is being picked up in blocks. Array *blocklens* contains the length of each block while array *indices* contains the displacements of each block in multiples of the input data type. Block *i* will consist of *blocklens[i]* elements displaced by *indeces[i]* positions.

```
int MPI_Type_indexed(int count,
                    int blocklens[],
                    int indices[],
                    MPI_Datatype old_type,
                    MPI_Datatype *newtype)
```

Parameters:
- count - number of blocks -- also number of entries in indices and blocklens
- blocklens - number of elements in each block (array of nonnegative integers)
- indices - displacement of each block in multiples of old_type (array of integers)
- old_type - old datatype (handle)
- newtype - new datatype (handle)

```
int MPI_Type_struct(int count,
                   int blocklens[],
                   MPI_Aint indices[],
                   MPI_Datatype old_types[],
                   MPI_Datatype *newtype)
```

Parameters:
- count - number of blocks (integer) -- also number of entries in arrays array_of_types , array_of_displacements and array_of_blocklengths
- blocklens - number of elements in each block (array)
- indices - byte displacement of each block (array)
- old_types - type of elements in each block (array of handles to datatype objects)

To get the **type extent**, one can use:

```
int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent)
```

Parameters:
- datatype - datatype (handle)
- extent - datatype extent (address integer)

To **commit** the new data type, one can use:

```
int MPI_Type_commit(MPI_Datatype *datatype)
```

Parameters:

- datatype – the type to be commited

To **remove** a data type, one can use:

int MPI_Type_free(MPI_Datatype *datatype)

Parameters:

- datatype – the type to be commited

## 2. Examples

Example 1: Contiguous types

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4

int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source=0, dest, tag=1, i;
float a[SIZE][SIZE] =
  {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};
float b[SIZE];

MPI_Status stat;
MPI_Datatype rowtype;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

MPI_Type_contiguous(SIZE, MPI_FLOAT, &rowtype);
MPI_Type_commit(&rowtype);

if (numtasks == SIZE) {
  if (rank == 0) {
     for (i=0; i<numtasks; i++)
       MPI_Send(&a[i][0], 1, rowtype, i, tag, MPI_COMM_WORLD);
     }

  MPI_Recv(b, SIZE, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &stat);
  printf("rank= %d  b= %3.1f %3.1f %3.1f %3.1f\n",
         rank,b[0],b[1],b[2],b[3]);
  }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);

MPI_Type_free(&rowtype);
MPI_Finalize();
}
```

Example 2: Vector types

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4

int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source=0, dest, tag=1, i;
float a[SIZE][SIZE] =
  {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};
float b[SIZE];

MPI_Status stat;
MPI_Datatype columntype;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

MPI_Type_vector(SIZE, 1, SIZE, MPI_FLOAT, &columntype);
MPI_Type_commit(&columntype);

if (numtasks == SIZE) {
  if (rank == 0) {
     for (i=0; i<numtasks; i++)
       MPI_Send(&a[0][i], 1, columntype, i, tag, MPI_COMM_WORLD);
        }

  MPI_Recv(b, SIZE, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &stat);
  printf("rank= %d  b= %3.1f %3.1f %3.1f %3.1f\n",
        rank,b[0],b[1],b[2],b[3]);
  }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);

MPI_Type_free(&columntype);
MPI_Finalize();
}
```

Example 3: Indexed types

```
#include "mpi.h"
#include <stdio.h>
#define NELEMENTS 6

int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source=0, dest, tag=1, i;
int blocklengths[2], displacements[2];
float a[16] =
  {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};
float b[NELEMENTS];

MPI_Status stat;
MPI_Datatype indextype;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

blocklengths[0] = 4;
blocklengths[1] = 2;
displacements[0] = 5;
displacements[1] = 12;

MPI_Type_indexed(2, blocklengths, displacements, MPI_FLOAT, &indextype);
MPI_Type_commit(&indextype);

if (rank == 0) {
  for (i=0; i<numtasks; i++)
     MPI_Send(a, 1, indextype, i, tag, MPI_COMM_WORLD);
  }

MPI_Recv(b, NELEMENTS, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &stat);
printf("rank= %d  b= %3.1f %3.1f %3.1f %3.1f %3.1f %3.1f\n",
     rank,b[0],b[1],b[2],b[3],b[4],b[5]);

MPI_Type_free(&indextype);
MPI_Finalize();
}
```

Example 4: Struct types

```
#include "mpi.h"
#include <stdio.h>
#define NELEM 25

int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source=0, dest, tag=1, i;

typedef struct {
  float x, y, z;
  float velocity;
  int  n, type;
  }          Particle;
Particle     p[NELEM], particles[NELEM];
MPI_Datatype particletype, oldtypes[2];
int          blockcounts[2];

/* MPI_Aint type used to be consistent with syntax of */
/* MPI_Type_extent routine */
MPI_Aint    offsets[2], extent;

MPI_Status stat;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

/* Setup description of the 4 MPI_FLOAT fields x, y, z, velocity */
offsets[0] = 0;
oldtypes[0] = MPI_FLOAT;
blockcounts[0] = 4;

/* Setup description of the 2 MPI_INT fields n, type */
/* Need to first figure offset by getting size of MPI_FLOAT */
MPI_Type_extent(MPI_FLOAT, &extent);
offsets[1] = 4 * extent;
oldtypes[1] = MPI_INT;
blockcounts[1] = 2;
```

```
/* Now define structured type and commit it */
MPI_Type_struct(2, blockcounts, offsets, oldtypes, &particletype);
MPI_Type_commit(&particletype);

/* Initialize the particle array and then send it to each task */
if (rank == 0) {
  for (i=0; i<NELEM; i++) {
     particles[i].x = i * 1.0;
     particles[i].y = i * -1.0;
     particles[i].z = i * 1.0;
     particles[i].velocity = 0.25;
     particles[i].n = i;
     particles[i].type = i % 2;
     }
  for (i=0; i<numtasks; i++)
     MPI_Send(particles, NELEM, particletype, i, tag, MPI_COMM_WORLD);
  }

MPI_Recv(p, NELEM, particletype, source, tag, MPI_COMM_WORLD, &stat);

/* Print a sample of what was received */
printf("rank= %d   %3.2f %3.2f %3.2f %3.2f %d %d\n", rank,p[3].x,
     p[3].y,p[3].z,p[3].velocity,p[3].n,p[3].type);

MPI_Type_free(&particletype);
MPI_Finalize();
}
```

# 4. Exercises:

1. Define a type called Student that stores information about students. Create a list of students, and then using n processors, search for one specific student.