Grouping Data for Communication

MPI provides methods for sending messages consisting of more than one scalar element. One can either build a derived datatype, or one can use the two functions MPI_Pack and MPI_Unpack. MPI_Type_contigous can be used to construct a type containing a subset of consecutive entries in an array. MPI_Type_vector can be used to construct a type consisting of array elements that are uniformly spaced in memory. MPI_Type_indexed can be used to construct a type consisting of array elements that are not uniformly spaced in memory. The most general constructor is MPI_Type_struct. If there are a large number of elements that are not in contiguous memory locations, then building a derived type will probably involve less overhead than a large number of calls to MPI_Pack/MPI_Unpack.

1. **MPI_Type_struct**

```
int MPI_Type_struct(
        int             count            /* in  */,
        int             block_lengths[]  /* in  */,
        MPI_Aint        displacements[]  /* in  */,
        MPI_Datatype    typelist[]       /* in  */,
        MPI_Datatype*   new_mpi_t        /* out */)
```

   It can be used to build derived types whose elements have different types and arbitrary locations in memory. **count** is the number of blocks of elements in the derived type. The array **block_lengths** contains the number of entries in each elements type. The array **displacements** contains the displacement of each element from the beginning of the message, and the array **typelist** contains the MPI datatype of each entry. The parameter **new_mpi_t** returns a pointer to the MPI datatype created by the call to MPI_Type_struct.

2. **MPI_Type_contiguous**

```
int MPI_Type_contiguous(
        int              count       /* in  */,
        MPI_Datatype     old_type    /* in  */,
        MPI_Datatype*    new_mpi_t   /* out */)
```

The derived type **new_mpi_t** will consist of **count** contiguous elements, each of which has type **old_type**.

3. **MPI_Type_vector**

```
int MPI_Type_vector(
        int             count           /* in  */,
        int             block_length    /* in  */,
        int             stride          /* in  */,
        MPI_Datatype    element_type    /* in  */,
        MPI_Datatype*   new_mpi_t       /* out */)
```

It can be used to construct a type consisting of array elements that are uniformly spaced in memory. **count** is the number of elements in the type. **block_length** is the number of entries in each element. **stride** is the number of elements of type **element_type** between successive elements of **new_mpi_t**.

4. **MPI_Type_indexed**

```
int MPI_Type_indexed(
        int             count            /* in  */,
        int             block_lengths[]  /* in  */,
        int             displacements[]  /* in  */,
        MPI_Datatype    old_type         /* in  */,
        MPI_Datatype*   new_mpi_t        /* out */)
```

The derived type consists of **count** elements of type **old_type**. the ith element consists of **block_lengths[i]** entries, and it is displaced **displacement[i]** units of **old_type** from the beginning (displacement 0) of the type.

5. **MPI_Type_commit**

```
int MPI_Type_commit(
        MPI_Datatype*   new_mpi_t       /* in/out */)
```

Before a derived type can be used by a communication function, it must be committed with a call to MPI_Type_commit.

6. **MPI_Type_Pack**

```
int MPI_Pack(
        void*           pack_data       /* in     */,
        int             in_count        /* in     */,
        MPI_Datatype    datatype        /* in     */,
        void*           buffer          /* out    */,
        int             buffer_size     /* in     */,
        int*            position        /* in/out */,
        MPI_Comm        comm            /* in     */)
```

This allows one to explicitly store data in a user-defined buffer. **pack_data** references the data to be buffered. It should consist of **in_count** elements, each having type **datatype**. **buffer_size** contains the size in bytes of the memory referenced by **buffer**. **position** keeps track of where data is in buffer, in bytes.

7. **MPI_Type_Unpack**

```
int MPI_Unpack(
        void*           buffer          /* in     */,
        int             size            /* in     */,
        int*            position        /* in/out */,
        void*           unpack_data     /* out    */,
        int             count           /* in     */,
        MPI_Datatype    datatype        /* in     */,
        MPI_Comm        comm            /* in     */)
```

It can be used to extract data from a buffer that was constructed using MPI_Pack. **buffer** references the data to be unpacked. it consists of **size** bytes. MPI_Unpack will copy **count** elements having type **datatype** into **unpack_data**.