MPI Datatypes

- Recall MPI has its own datatypes to wrap around primative types
  - MPI_CHAR
  - MPI_INT
  - MPI_DOUBLE
- MPI also allows the user to define their own datatypes
- Useful for packing data together into a single MPI_Send()
- Also allows sending of non-contiguous memory in a single communication

Type Maps

- Derived datatypes are made up of sets of primitive datatypes
- They form what is called a *type map*
- The map consists of the type and the displacement (or offset) in bytes from the first element
- Say we have a datatype consisting of a double, an int and finally a char then the map is
- `{double,0},{int,8},{char,12}`

Extent and Size

- The extent of a datatype is the distance between subsequent elements of the same datatype
- This can be different to the size of the datatype as certain variable may need to be aligned on 4 or 8 byte boundaries
- In the previous example the size of the data type is 13 bytes but the extent is 16 bytes
- This is because the double in a subsequent element could not start immediately beside the previous char due to memory alignment issues
- Use MPI_Type_extent(type, *extent) to determine

Contiguous Data

- Simplest derived datatype
- Consists of a number of contiguous items of the same datatype
- MPI_Type_contiguous(count, oldtype, *newtype);
- After we create a new datatype we must use MPI_Type_commit(newtype) before we use it for sending data
- When we are done we can use MPI_Type_free(newtype) to delete the datatype

Vector Data

- Vectors are similar to contiguous, but allow for gaps between the objects
- MPI_Type_vector(count, block, stride, oldtype, *newtype);
- There is a hvector version where stride is measured in bytes rather than objects.
- Useful for sending columns of matrix

```
MPI_Type_vector(nrows, 1, ncols, MPI_Double, &coltype);
MPI_Type_commit(coltype);
MPI_Send(&g[0][ncols-1], 1, coltype, dest, tag, comm);
```

Indexed Data

- A more generalized and less structured vector
- Allows sequences of blocks of different lengths with different offsets
- MPI_Type_indexed(count, *blocks, *offsets, oldtype, *newtype);
- Again a hindex version with offsets in bytes
- Not commonly used in scientific computing

MPI Structs

- General purpose wrapper for structured data
- MPI_Type_Struct(count, *blocks, *offsets, *oldtypes, *newtype);
- Remember that pointers within the struct won't be dereferenced.
- Just the value of the pointer will be sent which will most likely be useless on the receiving end.

MPI Structs

```
struct foo {
   double x; int l, m; char c;
};

MPI_Type_extent(MPI_DOUBLE, &exD);
MPI_Type_extent(MPI_INT, &exI);
MPI_Datatype type[3] = {MPI_DOUBLE, MPI_INT, MPI_CHAR};
int blocks[3] = {1, 2, 1};
int offsets[3] = {0, exD, exD+2*exI};

MPI_Type_struct(3, blocks, offsets, type, &newtype);
MPI_Type_commit(&newtype).
```

Packing and Unpacking

- Sometimes the data is not well organized to use vectors or structs
- Want to minimize the number of communications, so bundle data together into a single buffer before sending
- MPI_Pack(input, count, type, output, size, position, comm);
- Position is updated so subsequent calls to MPI_Pack will append data at the appropriate place
- Then use a single MPI_Send using MPI_PACKED datatype
- MPI_Unpack(input, count, type, output, size, position, comm) used to unpack the buffer on the recv'ing side

Summary

- User defined data types are great for sending non-contiguous data or data with a mixture of types in a single operation
- Remember to commit the new datatype before use
- These operation are not collective but must be run on each node where the datatype is to be used