<table>
<thead>
<tr>
<th>C Data Type</th>
<th>MPI Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>MPI_INT</td>
</tr>
<tr>
<td>float</td>
<td>MPI_FLOAT</td>
</tr>
<tr>
<td>double</td>
<td>MPI_DOUBLE</td>
</tr>
<tr>
<td>signed char</td>
<td>MPI_CHAR</td>
</tr>
<tr>
<td>unsigned byte</td>
<td>MPI_BYTE</td>
</tr>
</tbody>
</table>

MPI Data Types in C
MPI Data Types in C

<table>
<thead>
<tr>
<th>Type</th>
<th>MPI Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>MPI_UNSIGNED_SHORT</td>
</tr>
<tr>
<td>unsigned int</td>
<td>MPI_UNSIGNED</td>
</tr>
<tr>
<td>unsigned char</td>
<td>MPI_UNSIGNED_CHAR</td>
</tr>
<tr>
<td>short</td>
<td>MPI_SHORT</td>
</tr>
<tr>
<td>long double</td>
<td>MPI_DOUBLE</td>
</tr>
<tr>
<td>long</td>
<td>MPI_LONG</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td></td>
</tr>
<tr>
<td>MPI_DOUBLE_PRECISION</td>
<td></td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE</td>
<td></td>
</tr>
<tr>
<td>Fortran Datatype</td>
<td>MPI Datatype in Fortran</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>REAL</td>
<td>MPIREAL</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>MPILOGICAL</td>
</tr>
<tr>
<td>INTEGER</td>
<td>MPIINTEGER</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>MPIDOUBLE-PRECISION</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>MPICOMPLEX</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>MPICHAR</td>
</tr>
<tr>
<td></td>
<td>MPIBYTE</td>
</tr>
</tbody>
</table>

The table above illustrates the correspondence between Fortran datatypes and their MPI counterparts.
to be created or used.

**STRTUC**

- The most general that allows structures arbitrarily spaced data, displacement in bytes (useful for dynamic and displacement of the input datatype provided.

**HINDEXED**

- Same as INDEXED but with the

**INDEXED**

- In this datatype, an array of regular gaps in the displacements. creates a new datatype but allows

**VECTOR**

- Making copies of an existing one.

**CONTINUOUS**

- This produces a new datatype by

Some of the Derived Datatypes
Example of CONTIGUOUS DATATYPE

MPI_DATATYPE in C and integer in Fortran

oldtype and newtype should be declared

...may occur due to alignment restrictions.

(essentially) successfully in memory. (Some padding
creates a datatype newtype with two oldtypes placed

MPI-TYPE-SHIFTING(2, oldtype, newtype);

Assume an original datatype oldtype
MPI_Type_free (datatype);
MPI_Send(buffer, 1, datatype, dest, tag, comm);
MPI_Type_commit(datatype);
MPI_Type_continuous(count, datatype, datatype);

the use is a lot like stack manipulation:
To actually send such a data use the sequence of calls
In Fortran we have:

```fortran
type, pointer :: type_ptr
         type_ptr => NULL_PTR
```

In C we have:

```c
typedef MPI_Type vector;
```

Example of VECTOR Datatype

```c
 MPI_Type_vector(count, block_length, stride, &type_ptr);
```
(block to first element of next block)
newtype in units of oldtype (from first element of a
stride – distance between successive blocks of
oldtype in each block)
blocklength – number of (contiguous) entries of
count – total number of blocks
### Example of VECTOR Datatype

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
</tbody>
</table>
MPI

MPI-TYPEx.COMMIT(colType)
MPI-TYPEx.DOUBLE-PRECISION, colType, ierr
MPI-TYPEx-VECTOR(5, 1, 7)
data type

Assume row major ordering of array and define a row
send to each the respective rows and columns.

the north, south, east and west are known in order to

Assume that the ranks of the processes that are to

Portion is column major ordering

Portion array

Consider program to communicate the north and

More Examples of Data Types
C rowtype is 6 reals, located 4 apart.
call MPI_TYPE_VECTOR(6, 1, 4, MPI_REAL, rowtype, ierr)
call MPI_TYPE_COMMIT (rowtype, ierr)

C coltype is 4 contiguous reals.
call MPI_TYPE_CONTIGUOUS(4, MPI_REAL, coltype, ierr)
call MPI_TYPE_COMMIT(coltype, ierr)
call MPI_Type_free(comtype, ierr)
call MPI_Type_free(comtype, ierr)
call MPI_Type_free(comtype, ierr)

other code here that uses types

comm, ierr

call MPI_Sendrecvarray(4,1, comtype, south, 0, comtype, array)

call MPI_Sendrecvarray(1,1, comtype, north, 0, comtype, array)

call MPI_Sendrecvarray(1,1, comtype, east, 0, comtype, array)

call MPI_Sendrecvarray(1,1, comtype, west, 0, comtype, array)

other code here that uses types

***
Beginning of the structure: for each (in oldype)

- displacements is an array of displacements from blockheads
- blockheads is an array of length for each block
- count is the number of blocks

Structure:

- Indexed type allows varying the length of each block
- Indexed type
In C and Fortran respectively,

\begin{verbatim}
CALL MPI_TYPE_INDEXED(count, blocklengths, displacements, oldtype, newtype, ierr)

MPI_TYPE_INDEXED(count, blocklengths, displacements, oldtype, newtype)
\end{verbatim}
\[
\{ \\
\text{dispacements}[i] = (n+i) \times i; \\
\text{blockentries}[i] = n-i; \\
\}
\]

for (i=0; i < n; i++)

\text{MPI-datatype triangle};

int \text{dispacements}[n], \text{blockentries}[n];

float \text{D}[n][n];

float \text{A}[n][n];

Upper triangular part of an array.

Example of Indexed Type
MPI-recv(7, 1, terminate, 0, MPI-COMM-WORLD, &status);
else if

    MPI-send(4, 1, terminate, 1, 0, MPI-COMM-WORLD);
if (my-rank == 0)

    MPI-type-commit(terminate);
MPI-FLAT, terminate;
MPI-type-indexed(n, lock.length, displacements,
MPI-Type facilitates this.

Suppose we create a structure in C and wish to create

`struct Type`
struct rectangle
{
    char display[50];
    int maxiter;
    int width;
    double xmin, ymin;
    double xmax, ymax;
    int height;
    record;
};
Speaking not correct.

address, we have used int before which is strictly

should be done anytime a variable containing an

note we have used Aint(address int) here which

this is used to driver the MP structure

declarations corresponding to different data types

Note there are 4 different intervals in these
\[
\text{MPI-Datatype record-type:}
\]
\[
\text{MPI-Int displs[4];}
\]
\[
\text{MPI-Datatype types[4];}
\]
\[
\text{int blockcounts[4] = \{50,1,4,2\};}
\]
tyPes[2] = MPI-DouBlE;
tyPes[1] = MPI-INT;
tyPes[0] = MPI-CHAr;

MPI-message(ADDRESS, message, address, address, address, address, address, address, address);
Routines

either case can be handled in the send or recv

an arbitrary location: `dist[p][t] = dist[p][0]`

use or we must correct the displacement to handle

rectype so either this is the buffer we are going to

particular instance of the structure of type

note that we have found the addresses for a

MPI Address is a portable version of C
MPI.COMM_WORLD

MPI_Bcast(MPI_BOTTOM, 1, mpi_type, 0);

MPI_Type_commit(&mpi_type);

MPI_Type_struct(4, blockcounts, displs, types,

(addresses)

of the structure we leave the displacements as is (absolute)
To send using Mpi_send as a buffer containing the one instance