MPI - Derived Data Structures

Based on notes from
Science & Technology Support
High Performance Computing

Ohio Supercomputer Center

Derived Datatypes

- MPI Datatypes
- Procedure
- Datatype Construction
- Type Maps
- Contiguous Datatype*
- Vector Datatype*
- Extent of a Datatype
- Structure Datatype*
- Committing a Datatype

*includes sample C and Fortran programs

MPI Datatypes

- Basic types
- Derived types
  - Constructed from existing types (basic and derived)
  - Used in MPI communication routines to transfer high-level, extensive data entities
- Examples:
  - Sub-arrays or “unnatural” array memory striding
  - C structures and Fortran common blocks
  - Large set of general variables
- Alternative to repeated sends of varied basic types
  - Slow, clumsy, and error prone

Procedure

- *Construct* the new datatype using appropriate MPI routines
  `MPI_Type_contiguous, MPI_Type_vector, MPI_Type_struct, MPI_Type_indexed, MPI_Type_hvector, MPI_Type_hindexed`
- *Commit* the new datatype
  `MPI_Type_Commit`
- *Use* the new datatype in sends/receives, etc.

Datatype Construction

- Datatype specified by its *type map*
  - Stencil laid over memory
- Displacements are offsets (in bytes) from the starting memory address of the desired data
  - **`MPI_Type_extent`** function can be used to get size (in bytes) of datatypes

Type Maps

Basic datatype 0	Displacement of datatype 0
Basic datatype 1	Displacement of datatype 1
...	...
Basic datatype n-1	Displacement of datatype n-1

Contiguous Datatype

- The simplest derived datatype consists of a number of contiguous items of the same datatype

C:

```
int MPI_Type_contiguous (int count,
          MPI_Datatype oldtype,MPI_Datatype *newtype)
```

Fortran:

```
INTEGER COUNT, OLDTYPE, NEWTYPE, IERROR
CALL MPI_TYPE_CONTIGUOUS(COUNT,OLDTYPE,NEWTYPE,IERROR)
```


Sample Program #2 - C

```
#include <stdio.h>
#include <mpi.h>
/* Run with four processes */
 void main(int argc, char *argv[]) {
    int rank;
    MPI_Status status;
    struct {
     int x;     int y;     int z;
    } point;
    MPI_Datatype ptype;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Type_contiguous(3,MPI_INT,&ptype);
    MPI_Type_commit(&ptype);
    if(rank==3){
       point.x=15; point.y=23; point.z=6;
       MPI_Send(&point,1,ptype,1,52,MPI_COMM_WORLD);
    } else if(rank==1) {
       MPI_Recv(&point,1,ptype,3,52,MPI_COMM_WORLD,&status);
       printf("P:%d received coords are (%d,%d,%d)
   \n",rank,point.x,point.y,point.z);
    }
P:1 received coords are (15,23,6)
 }
```

Sample Program #2 - Fortran

```
     PROGRAM contiguous
C Run with four processes
      INCLUDE 'mpif.h'
      INTEGER err, rank, size
      integer status(MPI_STATUS_SIZE)
      integer x,y,z
      common/point/x,y,z
      integer ptype
      CALL MPI_INIT(err)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
      call MPI_TYPE_CONTIGUOUS(3,MPI_INTEGER,ptype,err)
      call MPI_TYPE_COMMIT(ptype,err)
      print *,rank,size
      if(rank.eq.3) then
         x=15
         y=23
         z=6
         call MPI_SEND(x,1,ptype,1,30,MPI_COMM_WORLD,err)
        else if(rank.eq.1)then
         call MPI_RECV(x,1,ptype,3,30,MPI_COMM_WORLD,status,err)
         print *,'P:',rank,' coords are ',x,y,z
        end if
P:1 coords are 15  23  6
      END
```


Vector Datatype

- User completely specifies memory locations defining the **vector**

C:

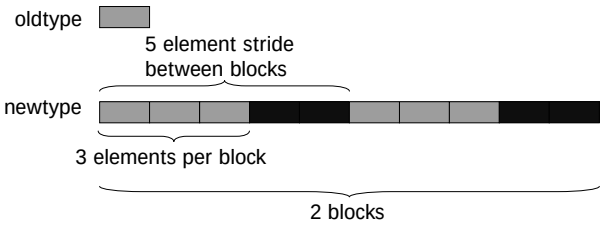
```
int MPI_Type_vector(int count, int blocklength, int
stride,
     MPI_Datatype oldtype, MPI_Datatype *newtype)
```

Fortran:

```
CALL MPI_TYPE_VECTOR(COUNT, BLOCKLENGTH, STRIDE,
                     OLDTYPE, NEWTYPE, IERROR)
```

- *newtype* has *count* blocks each consisting of *blocklength* copies of *oldtype*
- Displacement between blocks is set by *stride*

Vector Datatype Example



- count = 2
- stride = 5
- blocklength = 3

Sample Program #3 - C

```
#include <mpi.h>
#include <math.h>
#include <stdio.h>

 void main(int argc, char *argv[]) {
    int rank,i,j;
    MPI_Status status;
    double x[4][8];
    MPI_Datatype coltype;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Type_vector(4,1,8,MPI_DOUBLE,&coltype);
    MPI_Type_commit(&coltype);
    if(rank==3){
      for(i=0;i<4;++i)
       for(j=0;j<8;++j) x[i][j]=pow(10.0,i+1)+j;
      MPI_Send(&x[0][7],1,coltype,1,52,MPI_COMM_WORLD);
    } else if(rank==1) {
       MPI_Recv(&x[0][2],1,coltype,3,52,MPI_COMM_WORLD,&status);
       for(i=0;i<4;++i)printf("P:%d my x[%d][2]=%1f\n",rank,i,x[i][2]);
    }
    MPI_Finalize();
 }
```

```
P:1 my x[0][2]=17.000000
P:1 my x[1][2]=107.000000
P:1 my x[2][2]=1007.000000
P:1 my x[3][2]=10007.000000
```

Sample Program #3 - Fortran

```
      PROGRAM vector
C Run with four processes
      INCLUDE 'mpif.h'
      INTEGER err, rank, size
      integer status(MPI_STATUS_SIZE)
      real x(4,8)
      integer rowtype
      CALL MPI_INIT(err)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
      call MPI_TYPE_VECTOR(8,1,4,MPI_REAL,rowtype,err)
      call MPI_TYPE_COMMIT(rowtype,err)
      if(rank.eq.3) then
        do i=1,4
         do j=1,8
          x(i,j)=10.0**i+j
         end do
        enddo
         call MPI_SEND(x(2,1),1,rowtype,1,30,MPI_COMM_WORLD,err)
        else if(rank.eq.1)then
         call MPI_RECV(x(4,1),1,rowtype,3,30,MPI_COMM_WORLD,status,err)
         print *,'P:',rank,' the 4th row of x is'
         do i=1,8
          print*,x(4,i)
         end do
        end if
      CALL MPI_FINALIZE(err)
      END
```

```
P:1 the 4th row of x is
101.
102.
103.
104.
105.
106.
107.
108.
```


Extent of a Datatype

- Handy utility function for datatype construction
- Extent defined to be the memory span (in bytes) of a datatype

C:

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

Fortran:

```
INTEGER    DATATYPE, EXTENT, IERROR
CALL MPI_TYPE_EXTENT (DATATYPE, EXTENT, IERROR)
```


Structure Datatype

- Use for variables comprised of heterogeneous datatypes
  - C structures
  - Fortran common blocks
- This is the most general derived data type

C:

```
int MPI_Type_struct (int count, int
  *array_of_blocklengths,
                MPI_Aint *array_of_displacements,
                MPI_Datatype *array_of_types,
                MPI_Datatype *newtype)
```

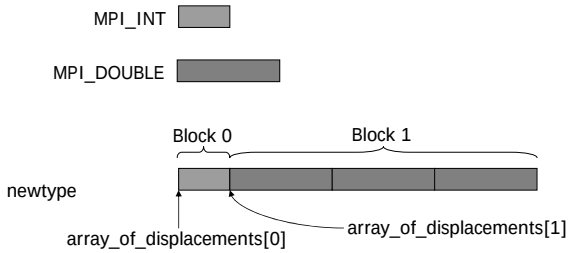
Fortran:

```
CALL MPI_TYPE_STRUCT(COUNT,ARRAY_OF_BLOCKLENGTHS,
                ARRAY_OF_DISPLACEMENTS,ARRAY_OF_TYPES,
                NEWTYPE,IERROR)
```


Structure Datatype (cont)

newtype consists of count blocks where the i^{th} block is array_of_blocklengths[i] copies of the type array_of_types[i]. The displacement of the i^{th} block (in bytes) is given by **array_of_displacements[i]**.

Struct Datatype Example



- count = 2
- array_of_blocklengths = {1,3}
- array_of_types = {MPI_INT, MPI_DOUBLE}
- array_of_displacements = {0, extent(MPI_INT)}

Sample Program #4 - C

```
#include <stdio.h>
#include<mpi.h>
void main(int argc, char *argv[]) {
 int rank,i;
 MPI_Status status;
 struct {
    int num;
    float x;
    double data[4];
  } a;
  int blocklengths[3]={1,1,4};
  MPI_Datatype types[3]={MPI_INT,MPI_FLOAT,MPI_DOUBLE};
  MPI_Aint displacements[3];
  MPI_Datatype restype;
  MPI_Aint intex,floatex;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  MPI_Type_extent(MPI_INT,&intex);MPI_Type_extent(MPI_FLOAT,&floatex
  );
  displacements[0]= (MPI_Aint) 0; displacemtnts[1]=intex;
  displacements[2]=intex+floatex;
  MPI_Type_struct(3,blocklengths,displacements,types,&restype);
```


Sample Program #4 - C (cont.)

```
...
   MPI_Type_commit(&restype);
   if (rank==3){
      a.num=6; a.x=3.14; for(i=0;i 4;++i) a.data[i]=(double) i;
      MPI_Send(&a,1,restype,1,52,MPI_COMM_WORLD);
   } else if(rank==1) {
      MPI_Recv(&a,1,restype,3,52,MPI_COMM_WORLD,&status);
      printf("P:%d my a is %d %f %lf %lf %lf %lf\n",
             rank,a.num,a.x,a.data[0],a.data[1],a.data[2],a.data[3]);
    }
    MPI_Finalize();
}
```

```
P:1 my a is 6 3.140000 0.000000 1.000000 2.000000 3.000002
```


Sample Program #4 - Fortran

```
      PROGRAM structure
      INCLUDE 'mpif.h'
      INTEGER err, rank, size
      integer status(MPI_STATUS_SIZE)
      integer num
      real x
      complex data(4)
      common /result/num,x,data
      integer blocklengths(3)
      data blocklengths/1,1,4/
      integer displacements(3)
      integer types(3),restype
      data types/MPI_INTEGER,MPI_REAL,MPI_COMPLEX/
      integer intex,realex

      CALL MPI_INIT(err)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
      call MPI_TYPE_EXTENT(MPI_INTEGER,intex,err)
      call MPI_TYPE_EXTENT(MPI_REAL,realex,err)
      displacements(1)=0
      displacements(2)=intex
      displacements(3)=intex+realex
```

Sample Program #4 - Fortran (cont.)

```
      call MPI_TYPE_STRUCT(3,blocklengths, displacements,types,
     &                     restype,err)
      call MPI_TYPE_COMMIT(restype,err)
      if(rank.eq.3) then
        num=6
        x=3.14
        do i=1,4
         data(i)=cmplx(i,i)
        end do
        call MPI_SEND(num,1,restype,1,30,MPI_COMM_WORLD,err)
       else if(rank.eq.1)then
        call MPI_RECV(num,1,restype,3,30,MPI_COMM_WORLD,status,err)
        print*,'P:',rank,' I got'
        print*,num
        print*,x
        print*,data
       end if
      CALL MPI_FINALIZE(err)
      END
```

```
P:1 I got
6
3.1400001
(1.,1.), (2.,2.), (3.,3.), (4.,4.)
```


Committing a Datatype

- Once a datatype has been constructed, it needs to be committed before it is used.
- This is done using `MPI_TYPE_COMMIT`

C:

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

Fortran:

```
CALL MPI_TYPE_COMMIT (DATATYPE, IERROR)
```


Examples of Datatype Use

- Building a derived data type
- Sending Row
- Sending Column
- Pack and Unpack

Problem Set

1) Write a program in which four processors search an array in parallel (each gets a fourth of the elements to search). All the processors are searching the integer array for the element whose value is 11. There is only one 11 in the entire array of 400 integers.

   By using the non-blocking MPI commands you have learned, have each processor continue searching until one of them has found the 11. Then they all should stop and print out the index they stopped their own search at.

   You have been given a file called data which contains the integer array (ASCII, one element per line). Before the searching begins have ONLY P0 read in the array elements from the data file and distribute one fourth to each of the other processors and keep one fourth for its own search.

Problem Set

2) Write a program for three processors in the boss-worker programming style. Specifically, P0 will be the boss. It will first create a 40-element real 1-D array. The value for each array element will be the its index raised to the 2.5 power.

Processors 1 and 2 will act as workers. P1 will calculate the average of all the array elements which have an odd index. P2 will calculate the average of all the array elements which have an even index. When the workers are done they will output their results.

P0 will transfer ONLY the odd-indexed elements of the array to P1. Similarly for P2 and the even-indexed elements. Your program should do these transfers by making a new derived vector data type that consists of every other element in the 1-D array. The boss and each worker should use only one send/receive pair, using the new data type.