MPI Datatypes

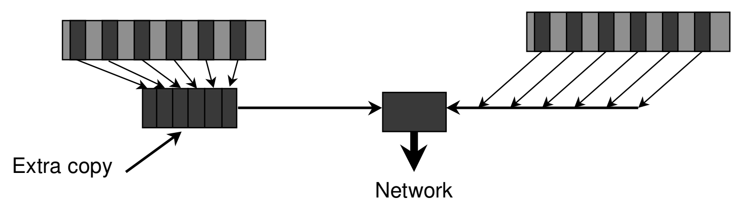
- The data in a message to sent or received is described by a triple (address, count, datatype), where
- An MPI *datatype* is recursively defined as:
  - » predefined, corresponding to a data type from the language (e.g., MPI_INT, MPI_DOUBLE_PRECISION)
  - » a contiguous array of MPI datatypes
  - » a strided block of datatypes
  - » an indexed array of blocks of datatypes
  - » an arbitrary structure of datatypes
- There are MPI functions to construct custom datatypes, such an array of (int, float) pairs, or a row of a matrix stored columnwise.

Why Datatypes?

- Since all data is labeled by type, an MPI implementation can support communication between processes on machines with very different memory representations and lengths of elementary datatypes (heterogeneous communication).
- Specifying application-oriented layout of data in memory
  - » can reduce memory-to-memory copies in the implementation
  - » allows the use of special hardware (scatter/gather) when available

Non-contiguous Datatypes

- Provided to *allow* MPI implementations to avoid copy



  » Not widely implemented *yet*
- Handling of important special cases
  » Constant stride
  » Contiguous structures

3

Potential Performance Advantage in MPI Datatypes

- Handling non-contiguous data
- Assume must pack/unpack on each end
  » $cn + (s + r\,n) + cn = s + (2c + r)n$
- Can move directly
  » $s + r'\,n$
  » r' probably > r but < $(2c+r)$
- MPI implementation must copy data anyway (into network buffer or shared memory); having the datatype permits removing 2 copies

4

Performance of MPI Datatypes

- Test of 1000 element vector of doubles with stride of 24 doubles (MB/sec).
  - » MPI_Type_vector
  - » MPI_Type_struct (MPI_UB for stride)
  - » User packs and unpacks by hand

	SP2	SP3	SGI	Origin	T3D
Vector	4.0	5.4	6.7	8.0	**12.7**
Struct	3.0	4.3	3.0	3.2	0.4
User	**15**	**7.9**	**40**	**47**	10.8

- Performance very dependent on implementation; should improve with time

Performance of MPI Datatypes II

- Performance of MPI_Type_Vector
  - » 1000 doubles with stride 24
  - » Performance in MB/sec

Equivalent Type_Struct

Platform	Vendor MPI	MPICH	User
IBM SP	3.79	6.71	9.52
Blue Pacific	2.94	1.02	
SGI O2000	7.63	20.2	43.8
Blue Mountain	3.09	2.6	
Intel Tflops	32.3	MPICH *is*	34.7
Red	3.29	Vendor MPI	

Datatype Abstractions

- Standard Unix abstraction is “block of contiguous bytes” (e.g., readv, writev)
- MPI specifies datatypes recursively as
  - » count of (type,offset)
    where offset may be relative or absolute
  - » MPICH uses a simplified form of this for MPI_Type_vector (and so outperforms vendor MPIs)
- More general form would provide superior performance for most user-defined datatypes
  - » MPI implementations can improve

7

Working With MPI Datatypes

- An MPI datatype defines a *type signature*:
  - » sequence of pairs: (basic type,offset)
  - » An integer at offset 0, followed by another integer at offset 8, followed by a double at offset 16 is
    - – (integer,0), (integer,4), (double,16)
  - » Offsets need not be increasing:
    - – (integer,64),(double,0)
- An MPI datatype has an extent and a size
  - » *size* is the number of bytes of the datatype
  - » *extent* controls how a datatype is used with the *count* field in a send and similar MPI operations
  - » extent is a misleading name

8

What does extent do?

- Consider
  MPI_Send(buf, count, datatype, …)
- What actually gets sent?
- MPI defines this as

```
do i=0,count-1
   MPI_Send(buf(1+i*extent(datatype)),1,
                 datatype,…)
```

- *extent* is used to decide where to send from (or where to receive to in MPI_Recv) for count > 1
- Normally, this is right after the last byte used for (i-1)

9

Changing the extent

- MPI-1 provides two special types, MPI_LB and MPI_UB, for changing the extent of a datatype
  - » This doesn't change the *size*, just how MPI decides what addresses in memory to use in offseting one datatype from another.
- Use MPI_Type_struct to create a new datatype from an old one with a different extent
  - » Use MPI_Type_create_resized in MPI-2

10

Sending Rows of a Matrix

- From Fortran, assume you want to send a row of the matrix
  A(n,m),
  that is, A(row,j), for j=1,…, m
- A(row,j) is not adjacent in memory to A(row,j+1)
- One solution: send each element separately:

```
Do j=1,m
   Call MPI_Send( A(row,j), 1, MPI_DOUBLE_PRECSION, …)
```

- Why not?

11

MPI Type vector

- Create a single datatype representing elements separated by a constant distance (*stride*) in memory
  - » m items, separated by a stride of n:
  - » MPI_Type_vector(m, 1, n,
    MPI_DOUBLE_PRECISION, newtype, ierr)
    MPI_Type_commit(newtype, ierr)
  - » Type_commit required before using a type in an MPI communication operation.
- Then send one instance of this type
  MPI_Send(a(row,1), 1, newtype, ….)

12

Test your understanding of Extent

- How do you send 2 rows of the matrix? Can you do this:
  MPI_Send(a(row,j),2,newtype,…)
- Hint: Extent(newtype) is distance from the first to last byte of the type
  - » Last byte is a(row,m)
- Hint: What is the first location of A that is sent after the first row?

13

Sending with MPI_Vector

- Extent(newtype) is ((m-1)*n+1)*sizeof(double)
  - » Last element sent is A(row,m)
- do i=0,1
  MPI_Send(buf(1+i*extent(datatype)),1,
  datatype,…)
  becomes
- MPI_Send(A(row,1:m),…) (i=0)
  MPI_Send(A(row+1,m),…) (i=1)
- ! The second step is not
  MPI_Send(A(row+1,1),…)

14

Solutions for Vectors

- MPI_Type_vector is for very specific uses
  - » rarely makes sense to use count other than 1
- Two send two rows, simply change the blockcount:
  MPI_Type_vector(m, 2, n,
   MPI_DOUBLE_PRECISION, newtype, ierr)
- Stride is still relative to basic type

15

Sending Vectors of Different Sizes

- How would you send A(i,2:m) and A(i+1,3:m) with a single MPI datatype?
  - » Use “count” to select the number, as in
    MPI_Send(A(i,2),m-1,type,…)
    MPI_Send(A(i+1,3),m-2,type,…)
- Hint: Use an extent of n elements

16

Striding Type

- Create a type with an extent of a column of the array:
  »
  ```
  types(1) = MPI_DOUBLE_PRECISION
  types(2) = MPI_UB
  displs(1) = 0
  displs(2) = n * 8   ! Bytes!
  blkcnt(1) = 1
  blkcnt(2) = 1
  call MPI_Type_struct( 2, blkcnt, displs, types,
                        newtype, ierr )
  ```
- MPI_Send(A(i,2),m-1,newtype,…) sends the elements A(i,2:m)

17

Exercises

- Write a program that sends rows of a matrix from one processor to another. Use both MPI_Type_vector and MPI_Type_struct methods
  - » Which is most efficient?
  - » Which is easier to use?
- Write a program that sends a matrix from one processor to another. Arrange the datatypes so that the matrix is received in transposed order
  - » A(i,j) on sender arrives in A(j,i) on receiver

18