Introduction to Parallel Processing

Lecture 7 : Advanced Datatypes

Professor Amanda Bienz

First, MPI Refresher

- Setup
  - MPI_Init called first by each process
  - MPI_Finalize called last by each process
- Datatypes:
  - Basic datatypes MPI_INT, MPI_CHAR, MPI_DOUBLE, etc
  - Can derive complex datatypes such as arrays, structs, striped types **(today)**
- Communicators: (i.e. MPI_COMM_WORLD)
  - All communication happens inside a communicator
  - Each process in communicator has rank in that communicator
  - Will talk about more complicated communicators later (i.e. subset of processes in a group)

First, MPI Refresher

- Basic point-to-point MPI communication is between pairs of processes in a communicator
- Communication is two-sided — sender must call send, receiver must call receive
- Sender specifies destination and a tag (*integer value*) associated with message
- Receiver specifics source and tag
    - MPI_Status can be used to receive message of any size or from any source **(today)**
- Must be careful to avoid deadlocks

MPI Datatypes

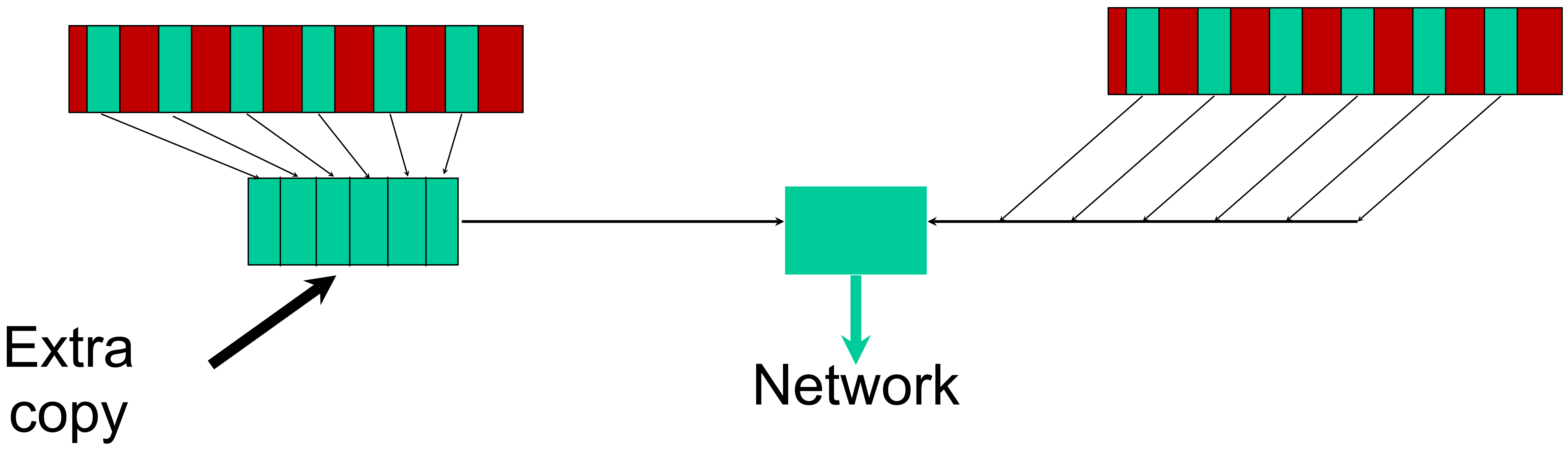
- The data in a message to be sent or received is described by:
  - Address of data
  - Count (size of data)
  - Datatype
- Predefined datatypes : MPI_INT, MPI_DOUBLE, MPI_CHAR, etc
- Can also have:
  - Contiguous array of multiple MPI_Datatypes
  - Strided block of datatypes
  - Arbitrary structure of datatypes
- Can also construct custom datatypes such as array of pairs, columns of a matrix stored row-wise

Why Datatypes?

- MPI implementation can support heterogeneous communication between machines
  - Different memory representations
  - Different lengths of elementary datatypes
- Specifies application-oriented layout of data in memory
  - Can reduce memory-to-memory copies
  - Allows 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)

Sending Columns of a Matrix

- Assume you want to send a column of the matrix A[m,n]
- A[i, column] is not adjacent in memory to A[i+1, column]
- **One solution: Send each element separately:**
  for i = 0 to m:
  MPI_Send(&(A[i, column], 1, MPI_DOUBLE, …)
- Why don't you want to do this?

Sending Columns of a Matrix

- Assume you want to send a column of the matrix A[m,n]
- A[i, column] is not adjacent in memory to A[i+1, column]
- **Another solution: Pack into contiguous buffer:**
  for i = 0 to m:
  buffer[i] = (A[i, column], 1, MPI_DOUBLE, …)
  MPI_Send(buffer, m, MPI_DOUBLE, …)
- Why don’t you want to do this?

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_Datatype newtype
  - MPI_Type_vector(m, 1, n, MPI_DOUBLE, &newtype)
  - MPI_Type_commit(&newtype)
  - Type_commit required before using the new type in an MPI communication operation
  - MPI_Type_free(&newtype)
- Now, can send the entire column in one instance :
  MPI_Send(&A[0, column], 1, newtype,…)

MPI Derived Datatype Calls

- MPI_Type_contiguous : allows you to create a datatype for a contiguous array (i.e. a row in C, column in Fortran)
- MPI_Type_struct : allows for heterogeneous datatypes
  - More complicated, structs usually have padding determined by compiler
  - Have to pass an array of types and an array of offsets
  - C primitive offsetof(struct, member) is useful for computing these offsets
- MPI_Type_indexed : allows for non-contiguous, unevenly spaced data (2D blocks of a matrix)

Packing Data

- MPI_Pack : pack data of any type into a contiguous char* buffer
- Predefined datatype : MPI_PACKED
- Useful for sending sparse matrices:
    - int* rowptr
    - int* col_indices
    - double* data
- Want to send full matrix as a single message

Packing Data

- ```
  MPI_Pack(const void* inbuf,
           int incount,
           MPI_Datatype datatype,
           void* outbuf,
           int outsize,
           int* position,
           MPI_Comm comm)
  ```

- Packs datatype into contiguous memory

- Can pack single value at a time or group of values at one time
  ```

Sending Packed Data

- MPI_Send(char *packed_buffer, packed_size, MPI_PACKED,
  int destination, int tag, MPI_Comm comm)
- Sends contiguous packed buffer

Unpacking Data

- MPI_Unpack(const void* inbuf,
  int insize,
  int* position,
  void* outbuf,
  int outcount,
  MPI_Datatype datatype,
  MPI_Comm comm)

- Unpacks received char* to original datatype

- Can unpack single value at a time, or in group of values at once

Helpful Packing Method

- MPI_Pack_size(int count,
  MPI_Datatype,
  MPI_Comm comm,
  int* pack_size)

- Know the original size of data, this method will return the size of the data once it is packed

How do we actually send / recv this?

- Now we can send all data at once time!
- This is great, because there is a large overhead associated with sending each message, regardless of size
- Fewer messages = cheaper
- However, how do we receive a sparse matrix without knowing the size we will be receiving?

Receiving a Message of any size

- MPI_Probe(int source,
  int tag,
  MPI_Comm comm,
  MPI_Status* status)

  - Waits until the process finds a message from ‘source’ of any size

- MPI_Get_count(MPI_Status* status,
  MPI_Datatype datatype,
  int* count)

  - Returns the size of the message found by probe

- Then, call MPI_Recv with this count!

Receiving a message from any process

- MPI_ANY_SOURCE receives a message from any source
- MPI_STATUS.MPI_SOURCE : the sending process attached to a message
- Can use this in a few ways
  - MPI_Recv(MPI_ANY_SOURCE, count, …) recv a message of size ‘count’ from any process
    - Only works if all messages will have size count
  - MPI_Probe(MPI_ANY_SOURCE, int tag, MPI_Comm comm, MPI_Status* status)
    - Finds a message sent to my process, from any source, of any size
    - Can then find the source (status.MPI_SOURCE) and size (MPI_Get_count(…))