Lecture 31: Introduction to the Message Passing Interface (MPI) cont.

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu
Worksheet #30: MPI send and receive

In the space below, use the minimum amount of non-blocking communication to reach the print statement in line 10 (assume that the program is executed with two MPI processes).

1. int a[], b[];
2. ...
3. if (MPI.COMM_WORLD.rank() == 0) {
   4.   MPI.COMM_WORLD.Isend(a, 0, 10, MPI.INT, 1, 1);
   5.   MPI.COMM_WORLD.Send(b, 0, 10, MPI.INT, 1, 2);
   6. }
7. else {
   8.   Status s2 = MPI.COMM_WORLD.Recv(b, 0, 10, MPI.INT, 0, 2);
   9.   Status s1 = MPI.COMM_WORLD.Recv(a, 0, 10, MPI_INT, 0, 1);
   10.  System.out.println("a = " + a + " ; b = " + b);
   11. }
12. ...
if (myrank == 0) {
    // I’m the main process.
    a = new double[size * size];  // a = new double[size][size]
    b = new double[size * size];  // b = new double[size][size]
    c = new double[size * size];  // c = new double[size][size]

    // Initialize matrices.
    init( size );

    // Construct message components.
    averows = size / nprocs;
    extra = size % nprocs;
    offset[0] = 0;

    // Transfer matrices to each worker. Assume matching receives for Worker.
    for (int rank = 0; rank < nprocs; rank++) {
        rows[0] = ( rank < extra ) ? averows + 1 : averows;
        System.out.println( "Main process transmitting " + rows[0] + " rows to rank " + rank );
        if (rank != 0) {
            final double[] copy = new double[rows[0] * size];
            System.arraycopy(a, offset[0] * size, copy, 0, rows[0] * size);
            // TODO: send data to process rank with tag tagFromMain
            // using the temporary copy created above. Note that the length of a single row = size.
            // TODO: MPI.COMM_WORLD.send( . . .);
            // TODO: send correct elements in array b to process rank with tag tagFromMain
            // TODO MPI.COMM_WORLD.send( . . .);
        }
        offset[0] += rows[0];
    }
if (myrank == 0) {
  // I'm the main process.
  a = new double[size * size]; // a = new double[size][size]
  b = new double[size * size]; // b = new double[size][size]
  c = new double[size * size]; // c = new double[size][size]

  // Initialize matrices.
  init(size);

  // Construct message components.
  averows = size / nprocs;
  extra = size % nprocs;
  offset[0] = 0;

  // Transfer matrices to each worker. Assume matching receives for Worker.
  for (int rank = 0; rank < nprocs; rank++) {
    rows[0] = (rank < extra) ? averows + 1 : averows;
    System.out.println("Main process transmitting " + rows[0] + " rows to rank " + rank);
    if (rank != 0) {
      MPI.COMM_WORLD.send(offset, 0, 1, MPI.INT, rank, tagFromMain);
      MPI.COMM_WORLD.send(rows, 0, 1, MPI.INT, rank, tagFromMain);

      final double[] copy = new double[rows[0] * size];
      System.arraycopy(a, offset[0] * size, copy, 0, rows[0] * size);
      MPI.COMM_WORLD.send(copy, rows[0] * size, MPI.DOUBLE, rank, tagFromMain);
      MPI.COMM_WORLD.send(b, 0, size * size, MPI.DOUBLE, rank, tagFromMain);
    }
    offset[0] += rows[0];
  }...
if (myrank == 0) {
  ...
  //previous slide

  // Collect results from each worker. Assume matching sends from worker.
  for (int source = 1; source < nprocs; source++) {
    // TODO: receive date from process source with tag tagFromWorker
    // using the temporary copy created below (assume matching send from worker).
    final double[] copy = new double[...];
    System.arraycopy(copy, 0, c, ..., ...);
  }
}

if (myrank == 0) {
    ... //previous slide

    // Collect results from each worker. Assume matching sends from worker.
    for (int source = 1; source < nprocs; source++) {
        MPI.COMM_WORLD.recv(offset, 0, 1, MPI.INT, source, tagFromWorker);
        MPI.COMM_WORLD.recv(rows, 0, 1, MPI.INT, source, tagFromWorker);
        final double[] copy = new double[rows[0] * size];
        MPI.COMM_WORLD.recv(copy, 0, rows[0] * size, MPI.DOUBLE, source, tagFromWorker);
        System.arraycopy(copy, 0, c, offset[0] * size, rows[0] * size);
    }
}...
Collective Communications

• A popular feature of MPI is its family of collective communication operations.

• Each collective operation is defined over a communicator (most often, MPI.COMM_WORLD)
  — Each collective operation contains an implicit barrier. The operation completes and execution
    continues when all processes in the communicator perform the same collective operation.
  — A mismatch in operations results in deadlock e.g.,
    Process 0: .... MPI.Bcast(...) ....
    Process 1: .... MPI.Bcast(...) ....
    Process 2: .... MPI.Gather(...) ....

• A simple example is the broadcast operation: all processes invoke the operation, all agreeing on one
  root process. Data is broadcast from that root.

  void Bcast(Object buf, int offset, int count, Datatype type, int root)
MPI Bcast

buf = new int[1]; if (rank==0) buf[0] = 29;
void Bcast(buf, 0, 1, MPI.INT, 0); // Executed by all processes

A root process sends same message to all

Broadcast can be implemented as a tree by MPI runtime

29 represents an array of values
More Examples of Collective Operations

void Gather(Object sendbuf, int sendoffset, int sendcount, Datatype sendtype, Object recvbuf, int recvoffset, int recvcount, Datatype recvtype, int root)

- Each process sends the contents of its send buffer to the root process.

void Scatter(Object sendbuf, int sendoffset, int sendcount, Datatype sendtype, Object recvbuf, int recvoffset, int recvcount, Datatype recvtype, int root)

- Inverse of the operation Gather.

void Reduce(Object sendbuf, int sendoffset, Object recvbuf, int recvoffset, int count, Datatype datatype, Op op, int root)

- Combine elements in send buffer of each process using the reduce operation, and return the combined value in the receive buffer of the root process.
MPI Gather

- Use to copy an array of data from each process into a single array on a single process.
- Graphically:

  ![Graphical representation of MPI Gather]

- Note: only process 0 (P0) needs to supply storage for the output

```c
void Gather(Object sendbuf, int sendoffset, int sendcount, Datatype sendtype, Object recvbuf, int recvoffset, int recvcount, Datatype recvtype, int root)
```

  - Each process sends the contents of its send buffer to the root process.
# Predefined Reduction Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Meaning</th>
<th>Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>Maximum</td>
<td>int, long, float, double</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>Minimum</td>
<td>int, long, float, double</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>Sum</td>
<td>int, long, float, double</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>Product</td>
<td>int, long, float, double</td>
</tr>
<tr>
<td>MPI_LAND</td>
<td>Logical AND</td>
<td>int, long</td>
</tr>
<tr>
<td>MPI_BAND</td>
<td>Bit-wise AND</td>
<td>byte, int, long</td>
</tr>
<tr>
<td>MPI_LOR</td>
<td>Logical OR</td>
<td>int, long</td>
</tr>
<tr>
<td>MPI_BOR</td>
<td>Bit-wise OR</td>
<td>byte, int, long</td>
</tr>
<tr>
<td>MPI_LXOR</td>
<td>Logical XOR</td>
<td>int, long</td>
</tr>
<tr>
<td>MPI_BXOR</td>
<td>Bit-wise XOR</td>
<td>byte, int, long</td>
</tr>
<tr>
<td>MPI_MAXLOC</td>
<td>max-min value-location</td>
<td>Data-pairs</td>
</tr>
<tr>
<td>MPI_MINLOC</td>
<td>min-min value-location</td>
<td>Data-pairs</td>
</tr>
</tbody>
</table>
MPI Reduce

void MPI.COMM_WORLD.Reduce(
    Object sendbuf /* in */,
    int sendoffset /* in */,
    Object recvbuf /* out */,
    int recvoffset /* in */,
    int count /* in */,
    MPI.Datatype datatype /* in */,
    MPI.Op operator /* in */,
    int root /* in */)

How would you write this using MPI Reduce with sendbuf size of 1?
MPI Reduce

```c
void MPI.COMM_WORLD.Reduce(
    Object sendbuf /* in */,
    int sendoffset /* in */,
    Object recvbuf /* out */,
    int recvoffset /* in */,
    int count /* in */,
    MPI.Datatype datatype /* in */,
    MPI.Op operator /* in */,
    int root /* in */ )
```

MPI.COMM_WORLD.Reduce(msg, 0, result, 0, 1, MPI.INT, MPI.SUM, 2);
Announcements & Reminders

• Quiz for Unit 7 is today at 11:59pm

• HW 4 Checkpoint #1 is due by Monday April 19th at 11:59pm
Worksheet #31: MPI_Gather

In the space below, indicate what value should be provided instead of ??? in line 6, and how it should depend on myrank.

2. MPI.Init(args) ;
3. int myrank = MPI.COMM_WORLD.Rank() ;
4. int numProcs = MPI.COMM_WORLD.Size() ;
5. int size = ...;
6. int[] sendbuf = new int[size];
7. int[] recvbuf = new int[???];
8. ... // Each process initializes sendbuf
9. MPI.COMM_WORLD.Gather(sendbuf, 0, size, MPI.INT,
10. recvbuf, 0, size, MPI.INT,
11. 0/*root*/);
12. ...
13. MPI.Finalize();