Blocking and non-blocking MPI communications

Blocking and non-blocking sends

- Blocking: return only when the buffer is ready to be reused.
- Non-blocking: return immediately.
- Buffering: data is kept until it is received.
- Synchronization: when a send is completed.
- Collective communications in MPI are always blocking.

Examples:

MPI_Send (): Blocking send. Will not return until you can use the send buffer.

MPI_Isend (): Nonblocking send. But not necessarily asynchronous. You CANNOT reuse the send buffer until either a successful wait/test or you certainly KNOW that the message has been received. An immediate send must return to the user without requiring a matching receive at the destination.

Example of a deadlock:

Both ranks wait for the other one to receive the message.

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

int main(int argc, char** argv) {
  MPI_Status status;
  int num;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&num);

  double d = 100.0;
  int tag = 1;

  if(num==0) {
    /* synchronous send: returns when the destination
       has started to receive the message */
    MPI_Ssend(&d,1,MPI_DOUBLE,1,tag,MPI_COMM_WORLD);
    MPI_Recv (&d,1,MPI_DOUBLE,1,tag,MPI_COMM_WORLD,&status);
  }
  else {
    MPI_Ssend(&d,1,MPI_DOUBLE,0,tag,MPI_COMM_WORLD);
    MPI_Recv (&d,1,MPI_DOUBLE,0,tag,MPI_COMM_WORLD,&status);
  }
  MPI_Finalize();
return 0;
}
```

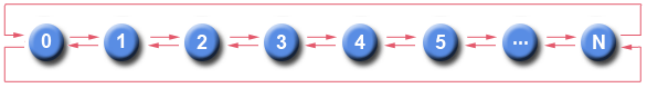
A remedy: use standard send & receive

```
if(num==0) {
  /* Standard send: can be synchronous or buffered,
     depending on message size */
  MPI_Send(&d,1,MPI_DOUBLE,1,tag,MPI_COMM_WORLD);
  MPI_Recv (&d,1,MPI_DOUBLE,1,tag,MPI_COMM_WORLD,&status);
}
else {
  MPI_Send(&d,1,MPI_DOUBLE,0,tag,MPI_COMM_WORLD);
  MPI_Recv (&d,1,MPI_DOUBLE,0,tag,MPI_COMM_WORLD,&status);
}
```

For such a small message MPI will always buffer it when using a standard send. However, the deadlock may still occur depending on the size of the buffer.

Use non-blocking MPI send & receive

Example: Nearest neighbor exchange in a ring topology




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

   main(int argc, char *argv[])  {
   int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
   MPI_Request reqs[4];   // required variable for non-blocking calls
   MPI_Status stats[4];   // required variable for Waitall routine

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   // determine left and right neighbors
   prev = rank-1;
   next = rank+1;
   if (rank == 0)  prev = numtasks - 1;
   if (rank == (numtasks - 1))  next = 0;

   // post non-blocking receives and sends for neighbors
   MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);
   MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, &reqs[1]);

   MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs[2]);
   MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD, &reqs[3]);

      // do some work while sends/receives progress in background

   // wait for all non-blocking operations to complete
   MPI_Waitall(4, reqs, stats);

      // continue - do more work

   MPI_Finalize();
   }
```

Homework

- Use the MPI non-blocking message passing and rewrite the heat_mpi.c example with a non-blocking neighbor communication.