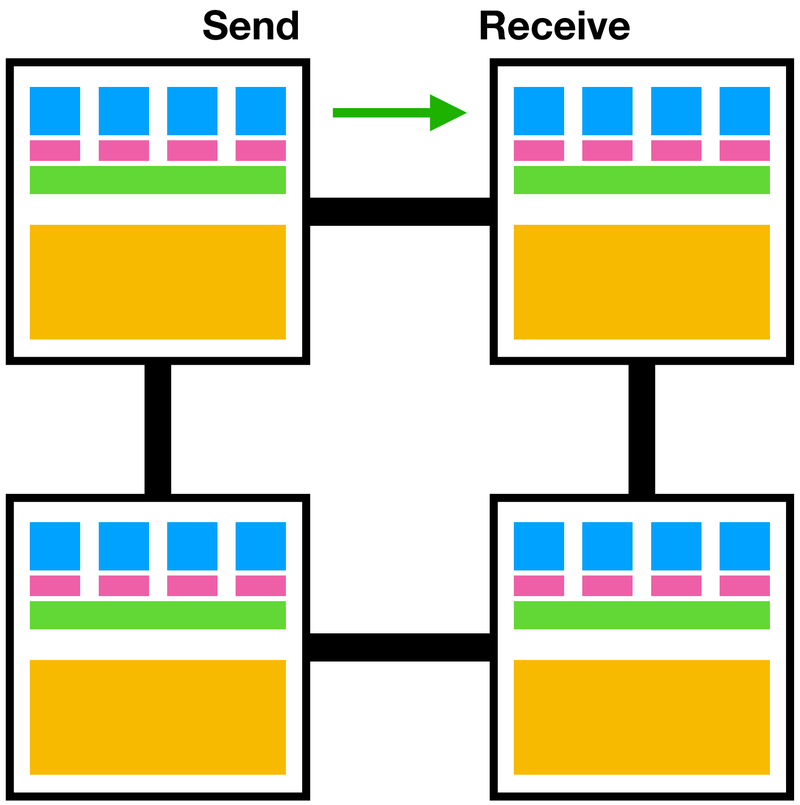
Introduction to Parallel Processing

Lecture 3 : Point-to-Point Communication

Professor Amanda Bienz

What is Point-to-Point Communication?



- Let's assume process 0 needs to send a message to process 1
- Also, would need process 1 to receive that message from process 0
- How do we do this?

MPI_Send / MPI_Recv

- MPI_Send(const void* send_buffer,
  int message_size,
  MPI_Datatype message_datatype,
  int destination_process,
  int message_tag,
  MPI_Comm communicator)

- Say we want process 0 to send a single integer called ‘size’ to process 1.

- What would the code look like?

MPI_Send / MPI_Recv

- **MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);**
- Say we want process 0 to send a single integer called ‘size’ to process 1.
- What would the code look like?

MPI_Send / MPI_Recv

- **But, we don't want every process to send this message to process 1. So, the actual code will look like this:**

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();

    if (rank == 0) MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);

    MPI_Finalize();
    return 0;
}
```

MPI_Send / MPI_Recv

- **What happens if you try to run this program? Does it work?**

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();

    if (rank == 0) MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);

    MPI_Finalize();
    return 0;
}
```

MPI_Send / MPI_Recv

- **No, it will hang, because no process is ever receiving this message!**
- MPI_Send is not guaranteed to return until the associated receive has been posted.
- Which brings us to :
- MPI_Recv(const void* **recv_buffer**
  int message_size,
  MPI_Datatype message_datatype,
  int **process_of_origin**
  int message_tag,
  MPI_Comm communicator,
  MPI_Status* status)

MPI_Send / MPI_Recv

- **Need to have process 0 send to process 1, and process 1 recv from process 0**

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();

    MPI_Status recv_status;
    if (rank == 0) MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);
    else if (rank == 1) MPI_Recv(&size, 1, MPI_INT, 0, 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

MPI_Send / MPI_Recv

- **Need to have process 0 send to process 1, and process 1 recv from process 0**

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();

    MPI_Status recv_status;
    if (rank == 0) MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);
    else if (rank == 1) MPI_Recv(&size, 1, MPI_INT, 0, 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

Do these variables need to be the same?

MPI_Send / MPI_Recv

- **Need to have process 0 send to process 1, and process 1 recv from process 0**

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    if (rank == 0) MPI_Send(&size, 1, MPI_INT, 1, 1234, MPI_COMM_WORLD);
    else if (rank == 1) MPI_Recv(&size_other, 1, MPI_INT, 0 , 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

Sending and Receiving

- What if we want every process to both send and recv?

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    int proc = 1 - rank%2;
    MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
    MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

Sending and Receiving

- What if we want every process to both send and recv?

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    int proc = 1 - rank%2;

    MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
    MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status)

    MPI_Finalize();
    return 0;
}
```

Each process is sending and then receiving.

But MPI_Send may not return until MPI_Recv has been executed.

Similarly, MPI_Recv won't return until it has received it's message.

So what do we do?

Sending and Receiving

- What if we want every process to both send and recv?

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    int proc = 1 - rank%2;
    if (rank % 2 == 0)
    {
        MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
        MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status);
    }
    else
    {
        MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status);
        MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

Many (or variable) sends and receives

- What if every process sends a large number of messages?
  Or if different processes send different numbers of messages?

- **MPI_Isend(const void* buf, int count, MPI_Datatype datatype,**
  int dest, int tag, MPI_Comm comm, MPI_Request* request);

- **MPI_Irecv(const void* buf, int count, MPI_Datatype datatype,**
  int dest, int tag, MPI_Comm comm, MPI_Request* request);

- **MPI_Wait(MPI_Request* request, MPI_Status* status);**

Many (or variable) sends and receives

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status send_status, recv_status;
    MPI_Request send_request, recv_request;
    int proc = 1 - rank%2;

    MPI_Isend(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &send_request);
    MPI_Irecv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_request);

    MPI_Wait(&send_request, &send_status);
    MPI_Wait(&recv_request, &recv_status);

    MPI_Finalize();
    return 0;
}
```

Many (or variable) sends and receives

- Why can you have all processes execute MPI_Isend before any execute MPI_Irecv?
- **MPI_Isend and MPI_Irecv are non-blocking, *meaning they return before the operation completes***
- MPI_Wait is the blocking call that waits for an operation to complete, but MPI_Isend and MPI_Irecv are both called before MPI_Wait, so everything can complete here

Some other points about this…

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status send_status, recv_status;
    MPI_Request send_request, recv_request;
    int proc = 1 - rank%2;

    MPI_Isend(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &send_request);
    MPI_Irecv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_request);

    MPI_Wait(&send_request, &send_status);
    MPI_Wait(&recv_request, &recv_status);

    MPI_Finalize();
    return 0;
}
```

MPI_Send Routines

- MPI_Send : will not return until you can re-use the send_buffer (the data that you are sending).
  - This is not the same thing as blocking until there is a matching receive posted
  - This means that if each process sends, and then each process receives, this may or may not work, depending on the MPI implementation

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    int proc = 1 - rank%2;
    MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
    MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

This code may or may not cause a deadlock

Other MPI_Send Routines

- MPI_Bsend : Buffer send, returns immediately and you can use the send buffer. However, performance may suffer.
- MPI_Ssend : Will not return until the matching receive has been posted
- MPI_Rsend : May be used only if matching receive is already posted.

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // Declare the variable that I want to send
    int size = rand();
    int size_other;

    MPI_Status recv_status;
    int proc = 1 - rank%2;
    MPI_Send(&size, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD);
    MPI_Recv(&size_other, 1, MPI_INT, proc, 1234, MPI_COMM_WORLD, &recv_status);

    MPI_Finalize();
    return 0;
}
```

Other MPI_Send Routines

- MPI_Isend : Non-blocking send, but not necessarily asynchronous. You can NOT use the send buffer until after MPI_Wait (or similar routine)
- MPI_Ibsend : Buffer non-blocking send
- MPI_Issend : Synchronous non-blocking send
- MPI_Irsend : Non-blocking version of MPI_Rsend