What is MPI?

**Messaging Passing Interface**
A language-independent communication protocol for parallel computers

- Run the same code on a number of **nodes** (different hardware threads, servers)
- Explicit message passing
- Dominant model for high performance computing (the de-facto standard)
High Level MPI

- MPI is a type of SPMD (single process, multiple data)

- Idea: to have multiple instances of the same program all working on different data

- The program could be running on the same, or cluster of machines

- Allow simple communication of data between processes
MPI Functions

// Initialize MPI
int MPI_Init(int *argc, char **argv)

// Determine number of processes within a communicator
int MPI_Comm_size(MPI_Comm comm, int *size)

// Determine processor rank within a communicator
int MPI_Comm_rank(MPI_Comm comm, int *rank)

// Exit MPI (must be called last by all processors)
int MPI_Finalize()

// Send a message
int MPI_Send(void *buf, int count, MPI_Datatype datatype,
             int dest, int tag, MPI_Comm comm)

// Receive a message
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
             int source, int tag, MPI_Comm comm,
             MPI_Status *status)
MPI Function Notes

- MPI_Datatype is just an enum, MPI_Comm is commonly MPI_COMM_WORLD for the global communication channel

- dest/source are the “rank” of the process to send the message to/receive the message from
  - You may use MPI_ANY_SOURCE in MPI_Recv

- Both MPI_Send and MPI_Recv are blocking calls

- You can use man MPI_Send or man MPI_Recv for good documentation

- The tag allows you to organize your messages, so you can receive only a specific tag
Example

Here’s a common example:

- Have the “master” (rank 0) process create some strings and send them to the worker processes
- The worker processes modify the string and send it back to the master
Example Code (1)

```c
/*
   "Hello World" MPI Test Program
*/
#include <mpi.h>
#include <stdio.h>
#include <string.h>

#define BUFSIZE 128
#define TAG 0

int main(int argc, char *argv[]) {
    char idstr[32];
    char buff[BUFSIZE];
    int numprocs;
    int myid;
    int i;
    MPI_Status stat;
```
Example Code (2)

```c
/* all MPI programs start with MPI_Init; all 'N'
 * processes exist thereafter */
MPI_Init(&argc,&argv);

/* find out how big the SPMD world is */
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

/* and this processes' rank is */
MPI_Comm_rank(MPI_COMM_WORLD,&myid);

/* At this point, all programs are running equivalently, */
/* the rank distinguishes the roles of the programs in */
/* the SPMD model, with rank 0 often used specially... */
```
if (myid == 0) {
    printf("%d: We have %d processors\n", myid, numprocs);
    for (i=1; i<numprocs; i++) {
        sprintf(buff, "Hello %d! ", i);
        MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
    }
    for (i=1; i<numprocs; i++) {
        MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &stat);
        printf("%d: %s\n", myid, buff);
    }
}
```c
else
{
    /* receive from rank 0: */
    MPI_Recv(buf, BUFSIZE, MPI_CHAR, 0, TAG,
              MPI_COMM_WORLD, &stat);
    printf(idstr, "Processor %d", myid);
    strncat(buf, idstr, BUFSIZE-1);
    strncat(buf, "reporting for duty", BUFSIZE-1);
    /* send to rank 0: */
    MPI_Send(buf, BUFSIZE, MPI_CHAR, 0, TAG,
             MPI_COMM_WORLD);
}

/* MPI Programs end with MPI Finalize; this is a weak * synchronization point */
MPI_Finalize();
return 0;
}
// Wrappers for gcc (C/C++)
mpicc
mpicxx

// Compiler Flags
OMPI_MPICC_CFLAGS
OMPI_MPICXX_CXXFLAGS

// Linker Flags
OMPI_MPICC_LDFLAGS
OMPI_MPICXX_LDFLAGS

OpenMPI does not recommend you to set the flags yourself, to see them try:

# Show the flags necessary to compile MPI C applications
shell$ mpicc --showme:compile

# Show the flags necessary to link MPI C applications
shell$ mpicc --showme:link
Compiling and Running

mpirun -np <num_processors> <program>
mpiexec -np <num_processors> <program>

- Starts num_processors instances of the program using MPI

```
jon@riker examples master % mpicc hello_mpi.c
jon@riker examples master % mpirun -np 8 a.out
0: We have 8 processors
0: Hello 1! Processor 1 reporting for duty
0: Hello 2! Processor 2 reporting for duty
0: Hello 3! Processor 3 reporting for duty
0: Hello 4! Processor 4 reporting for duty
0: Hello 5! Processor 5 reporting for duty
0: Hello 6! Processor 6 reporting for duty
0: Hello 7! Processor 7 reporting for duty
```

- By default, MPI uses the lowest-latency resource available (shared memory in this case)
Other Things MPI Can Do

- We can use nodes on a network (by using a hostfile)
- We can even use MPMD
  - multiple processes, multiple data

```bash
% mpirun -np 2 a.out : -np 2 b.out
```

This launches a single parallel application
- All in the same MPI_COMM_WORLD
- Ranks 0 and 1 are instances of a.out
- Ranks 2 and 3 are instances of b.out

You could also use the -app flag with an appfile instead of typing out everything
Performance Considerations

- Your bottleneck for performance here is messages

- Keep the communication to a minimum

- The more machines, the slower the communication in general
Summary

- MPI is a powerful tool for highly parallel computing across multiple machines

- Programming is similar to a more powerful version of fork/join

- If you don’t have a partner and would like one for Assignment 3, email me

- If you don’t want a partner, email me so it can be added to the site

- If you have a partner, email me so it can be added to the site