15-440: Distributed Systems

Recitation 9 - MPI

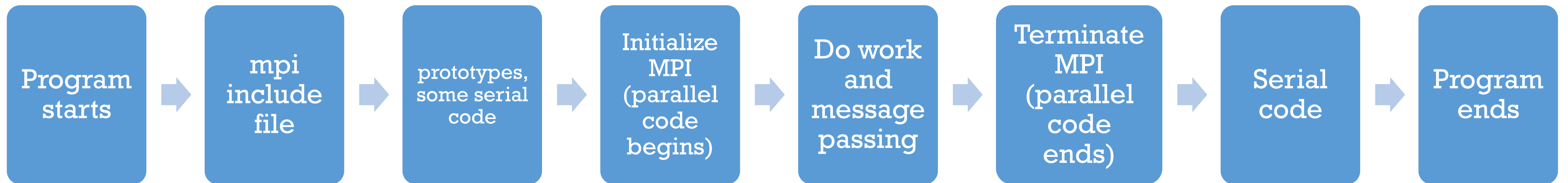
Zeinab Khalifa

Sept 24, 2019

What is MPI?

- It's a library of routines that can be used to create parallel programs
- Applications can be written in C, C++ and calls to MPI can be added where required

Program structure



Communicators & Groups

- They define which collection of processes may communicate with each other
- Processes can be collected into groups and/or communicators.
- Groups are objects that represent groups of processes
- Communicator is a set of processes that may communicate with each other and may consist of processes from a single group or multiple groups.
- When an MPI application starts, the group of all processes is initially given a predefined name called **MPI_COMM_WORLD**

Ranks

- Within a communicator, each process has its own and unique ID or rank
- These IDs are commonly used conditionally to control program execution
- Ranks start from 0

MPI Routines

- **MPI_INIT –** initialize the MPI library (must be the first routine called)
- **MPI_COMM_SIZE -** determines the number of processes in the group associated with the comm communicator
- **MPI_COMM_RANK –** get the rank of the calling process in the communicator
- **MPI_SEND –** send a message to another process
- **MPI_RECV –** send a message to another process
- **MPI_FINALIZE –** clean up all MPI state (must be the last MPI function called by a process)
- **MPI_Wtime –** determines elapsed wall clock time in seconds on the calling processor. We'll use this to measure the runtime of an MPI program

Example (1)

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

int main(int argc, char ** argv) {
      int rank, size;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &size);
      printf("I am %d of %d\n", rank, size);
      MPI_Finalize();
      return 0;
}
```

Example (1)

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

int main(int argc, char ** argv) {
      int rank, size;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      MPI_Comm_size(MPI_COMM_WORLD, &size);
      printf("I am %d of %d\n", rank, size);
      MPI_Finalize();
      return 0;
}
```

Communication model (send/receive)

Each process sends/receives data to/from other processes.

MPI Send

- **MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
- This is a basic blocking send operation. It returns only after the application has sent the data to the recipient(s)
- MPI **Datatype** is very similar to a C datatype: MPI_INT, MPI_CHAR
- The **count** refers to how many datatype elements should be communicated
- **tag** is a user-defined "type" for the message
- **dest** is the rank of the target process in the communicator specified by **comm**.

MPI Recv

- **MPI_Recv(void *buf, int count, MPI_Datatype datatype, int src, int tag, MPI_Comm comm, MPI_Status *status)**
- This receives a message and blocks until the requested data is available in the application buffer
- source is rank in communicator comm
- status contains further information on who sent the message, how much data was actually received,..

Example (2)

```
int main(int argc, char ** argv) {
      int rank, data[100];
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      if (rank == 0)
            MPI_Send(data, 100, MPI_INT, 1, 0,  MPI_COMM_WORLD);
      else if (rank == 1)
            MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,
            MPI_STATUS_IGNORE);
      MPI_Finalize();
      return 0;
}
```

Master code

Slave with rank == 1 code

Example (3)

Distributed Sum Program

(code and handout are attached)

Example (3) master code walkthrough

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Example (3) master code walkthrough

Calculating the time on the calling processor to measure the runtime of the program.

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Example (3) master code walkthrough

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Initializing the buffer

Example (3) master code walkthrough

Starting from process with rank 1 (j=1, slaves) give each process a starting index based on the number of elements per process. Then, send the buffer elements to the respective process.

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Example (3) master code walkthrough

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Summing master's elements

Example (3) master code walkthrough

```
double startTime = MPI_Wtime();
int i;
for(i = 0; i < NUM_ELEMS; i++)
        sendBuff[i] = i+1;
int j;
for(j=1; j<NUM_PROCS; j++){
        int startIndex = j * NUM_ELEM_PER_PROC;
        int endIndex = startIndex + NUM_ELEM_PER_PROC;
        MPI_Send(&sendBuff[startIndex], NUM_ELEM_PER_PROC, MPI_INT, j, TAG_X, MPI_COMM_WORLD);
}
int l;
for(l=0; l < NUM_ELEM_PER_PROC; l++)
        sum += sendBuff[l];
int partial_sum = 0;
MPI_Recv(&partial_sum, 1, MPI_INT, 1, TAG_Y, MPI_COMM_WORLD, &status);
sum += partial_sum;
double endTime = MPI_Wtime();
printf("Grand Sum = %d and it took = %f\n", sum, (endTime - startTime));
```

Receiving partial sums from other processes and adding them to my own sum

Example (3) slave code walkthrough

```
MPI_Recv(&recvBuff, NUM_ELEM_PER_PROC, MPI_INT, 0, TAG_X, MPI_COMM_WORLD, &status);
int partial_sum = 0;
int k;
for(k=0; k < NUM_ELEM_PER_PROC; k++)
        partial_sum += recvBuff[k];
MPI_Send(&partial_sum, 1, MPI_INT, 0, TAG_Y, MPI_COMM_WORLD);
```

Receive the buffer from the master

Example (3) slave code walkthrough

```
MPI_Recv(&recvBuff, NUM_ELEM_PER_PROC, MPI_INT, 0, TAG_X, MPI_COMM_WORLD, &status);
int partial_sum = 0;
int k;
for(k=0; k < NUM_ELEM_PER_PROC; k++)
        partial_sum += recvBuff[k];
MPI_Send(&partial_sum, 1, MPI_INT, 0, TAG_Y, MPI_COMM_WORLD);
```

Add the buffer elements

Example (3) slave code walkthrough

```
MPI_Recv(&recvBuff, NUM_ELEM_PER_PROC, MPI_INT, 0, TAG_X, MPI_COMM_WORLD, &status);
int partial_sum = 0;
int k;
for(k=0; k < NUM_ELEM_PER_PROC; k++)
        partial_sum += recvBuff[k];
MPI_Send(&partial_sum, 1, MPI_INT, 0, TAG_Y, MPI_COMM_WORLD);
```

Send the partial sum to the master

Example (4) Sorting an array

Process (1)

Process (1)

Process (2)

Process (1)

Example (4) Sorting an array

```
int rank;
int a[1000], b[500];
MPI_Init(&argc, &argv); MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
      MPI_Send(&a[500], 500, MPI_INT, 1, 0, MPI_COMM_WORLD);
      sort(a, 500);
      MPI_Recv(b, 500, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
      /* Serial: Merge array b and sorted part of array a */
} else if (rank == 1) {
      MPI_Recv(b, 500, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
      sort(b, 500);
      MPI_Send(b, 500, MPI_INT, 0, 0, MPI_COMM_WORLD);
} MPI_Finalize();
```

Running MPI

- machinefile

  contains a list of the possible machines on which you want your MPI program to run

- Compiling:

  mpicc HelloWorld.c -o HelloWorld

- Copying object file (to all machines you want to use)

  scp -p "HelloWorld" andrewid-n02.qatar.cmu.local:/home/hadoop/

- Running the program:

  mpiexec -f machinefile -n 2 ./HelloWorld

VMs

- 4 VMs/nodes provisioned (will be shared with each student)
- Coding in C
- Using n01 as your primary (master) VM