Lab Guide 7

Introduction to MPI

Objectives:

- basic message passing concepts
- concept of a single program on multiple data (SPMD)
- pipeline parallelism pattern

Introduction

This lab session aims to introduce the basic concepts of MPI programming, starting from a basic program with two processes, where one process sends a message to another process. The basic program will be extended to support any number of processes (a pipeline of N processes) and any number of messages among processes.

The program should be compiled in the cluster frontend using the `mpicc -O2 prog.c` command. To run the program, use the `sbatch mpi.sh`. The `mpi.sh` file should specify the required resources and should run the MPI program. The following example requests two PUs during 1 second and spawns two MPI processes:

```
[search7edu]$ cat mpi.sh
#!/bin/bash
#SBATCH --time=1:00
#SBATCH --ntasks=2
#SBATCH --partition=cpar

mpirun -np 2 ./a.out
```

Note that the number of requested resources (`--ntasks`) must be the same as the number of resources used in the `mpirun` command.

Exercise 1 - Pipeline of processes

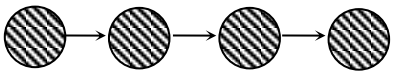
Compile and run the following MPI program, where the process with rank 0 sends a message (integer value 123456) to the process with rank 1:

```
#include <mpi.h>
#include <stdio.h>
int main( int argc, char *argv[]) {
   int rank, msg;
   MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank( MPI_COMM_WORLD, &rank ); // gets this process rank

   /* Process 0 sends and Process 1 receives */
   if (rank == 0) {
         msg = 123456;
         MPI_Send( &msg, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
   }
   else if (rank == 1) {
         MPI_Recv( &msg, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
         printf( "Received %d\n", msg);
   }
   MPI_Finalize();
   return 0;
}
```

Modify the program to implement a pipeline of processes (using the SPMD model):

a) Start by modifying the program to support a pipeline with four processes: process with rank 0 sends the message that is successively processed (e.g., printed) by each process in the pipeline.



b) Modify the program developed in **a)** to implement a pipeline with an arbitrary number of processes specified as a parameter in the command `mpirun -np xx`.
Note that MPI is based on the SPMD style of parallel programming. The same process will be spawned `xx` times. The number of processes spawned by the `mpirun` command can be retrieved with the `MPI_Comm_size` call.

c) Modify the program developed in **b)** to process 10 messages: the process with rank 0 should send 10 messages to the next in the pipeline; each other process should receive a message, process it (e.g., print) and send it to the next one in the pipeline.

Exercise 2 (optional) - Farm of processes and collective operations

Modify the original program to implement a directive-like behaving as "work sharing". A master process has a set of tasks to process, where each task will perform a given operation and produce as result an integer.
Each worker receives the required data to process its task — which is a message with the argument (an integer) — and returns the processed task to the master.
Implement the following variations:

a) <u>Static scheduling</u>: set the number of tasks to process equal to the number of MPI worker processes (one task per worker).

b) <u>Dynamic scheduling</u>: set the number of tasks as 10x the number of MPI worker processes; faster processes should get more tasks.

c) <u>Collective operations</u>: a message is broadcasted to all workers and then a reduce with the sum operation joins the results from all workers.