Parallel Programming 2014

COMMUNICATION IN A RING PROCESSES

October 3, 2014

Stephane KI
Abo Akademy University
I.T - Software Engineering
stephane.ki@abo.fi

1 Introduction

This exercise is about creating an MPI program that sends a message around a number of processes connected in a ring , and measures the time it takes for the message to travel around the ring. The program should work for any number of processes greater than or equal to two. In this report you will find a description of the implementation and the measured message transfer times. Join to this report you will find 3 files : *ring_mpi.c*, *ring_mpi_light.c*, and *test.sh*. You can also find them on *Github*: *https://github.com/kiStephane/Ring_MPI.git*

2 Description of the implementation

2.1 The message buffer

A buffer is initialize using a dynamic memory allocation with *malloc* and *memset* to fill it with one. This buffer will be used by all the porcesses to store the message received from its neighbor node. The maximum size allowed in this program is 1 GB.

2.2 Send the message size

In our program we choose to give the option to set the message type in two ways.

1. Directly as an argument of the program. *Example: srun -n 12 ./ring 1000*

2. During the execution : If the message size is not give before the execution , then during the execution of the process 0, the size will be asked to the user.

As said before the message size shall not more than 1 GB.

2.3 Send the message size to the world

The process 0 send the message size to the other nodes, so they can knox in advance what is the size of the message they will receive. To perform this the best way is to use the *MPI_Bcast* to send a message to all the processes including the source. To receive the message size the other processes also call *MPI_Bcast*.

2.4 Initialize and send the message

Knowing the message size, process 0 create a message of this specific size. The message is initialize with a specific number : we choosed 7. After the initialization the process send the message to the next node in the ring : node 1. After this send it wait for a message from the last node. Here we used *MPI_Sendrecv* routine.

2.5 Receive and forward the message to the next node

After receiving the message size , the other processes just wait for a message from the node before them in the node. When they receive the message they read it and then forward it to the next node. To know the next node we just calculate its rank : *int next=(id + 1)%np* ; where *np* represent the number of processes and *id* the rank of the current process.

3 Message transfer times

3.1 Results

Table 1: Measured message transfer times

Nr. of processors	Message size(bytes)	Times(s)
2	1	0.000073
2	1000	0.000084
2	1000000	0.000998
2	1000000000	0.630725
12	1	0.000162
12	1000	0.01734
12	1000000	0.05273
12	1000000000	3.8273
13	1	0.000967
13	1000	0.000624
13	1000000	0.007874
13	1000000000	4.556595
24	1	0.0015
24	1000	0.074640
24	1000000	0.021693
24	1000000000	8.664523
48	1	0.106654
48	1000	0.164730
48	1000000	0.150660
48	1000000000	16.676318

3.2 Comments

Our first observation is that the results obtained are close to those given as reference in the assignment. The second observation is that the transfer time seems to be linear related to the number of processes. That is normal because all the processes (except

process 0) are doing the same task so the time spent double if you double the number of processes.