Assignment 3

Marco Franzon

August 2020

1 Introduction

The aim of this assignment is to solve two exercises concerning MPI programming. The first of the two exercises requires to implement a cyclic sum between processors. The processors should communicate in a ring-like fashion. The second exercise required to initialise a distributed identity matrix of size N N. The matrix should then be printed (in the correct order) on the standard output if N ¡ 10; otherwise, it should be printed on a binary file.

2 Exercise 4

The processors exchange vectors of integers of size SIZE, set as a macro to 10^7, and each processor contains three vectors of size SIZE: send_buffer, recv_buffer, and sum. At first, send_buffer is initialised to the rank of the processor, and sum, to zero. At each step (there are nproc steps to complete the ring, where nproc is the number of processors) a processor sends its content send_buffer to the processor on its left, which will store it in recv_buffer. Each processor then updates its sum and overwrites send_buffer with recv_buffer to propagate the process.

The provided implementation carries out the communication using a non-blocking approach: MPI_Isend and MPI_Irecv allow a processor to update its sum without waiting for the send and receive operations to be completed. Two MPI_Wait calls then wait for the send and receive calls to be completed before updating the content of the processor to propagate the sum. The execution time is measured using MPI_Wtime. Its call is preceded by an MPI_Barrier to make sure that all the processes are ready to start when the start time is measured, and that all the processes are done communicating/computing when the total_time is measured.

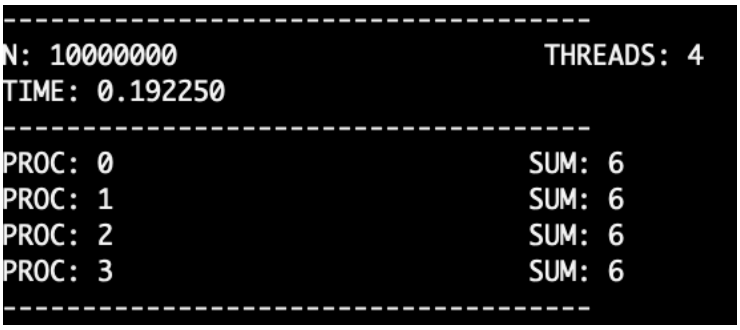

Figure 1: Ring communication output

3 Exercise 5

The executable must be fed with the matrix size, passed as an argument. The script, on the other hand, must be provided with the size of the matrix and the number of processors as arguments. In the following example the program will deal with a 5x5 matrix and with 4 processors.

The initialisation is performed by striping the matrix over the row index, assigning a portion of the matrix to each processor. If the number of processors is not a divisor of the matrix size, the rest is distributed to all the processors which have rank ¡ modulus, to guarantee balance. Each processor allocates its portion of the matrix and fills it. Each processor then sends its portion to the root process (process 0). Depending on N, process 0 prints to the standard output or to file.

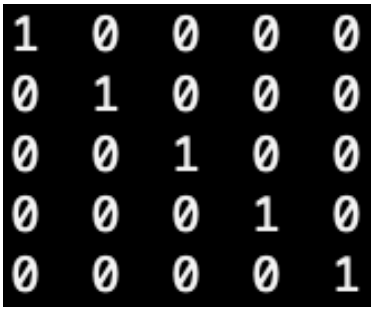

Figure 2: Identity matrix: 5x5 with 4 threads