CENG 478 – Introduction to Parallel Computing
SPRING 2014-2015 Homework-1
Due to: 22.03.2014 23:55

PARALLEL SUMMATION

In this assignment, you will implement a parallel algorithm that finds the minimum number in an array of numbers using MPI processes and compare the results with the MPI_Reduce method.

Algorithm:

Let n denote the length of array of double precision numbers, p the number of processes and p_i the i^{th} process. Assume that p is even and the array is distributed among the processes and each have almost the same amount of elements ($\cong n/p$).

In the first part of the algorithm, each processor finds the minimum of its own segment of the array independently.

In the second part which involves inter-process communication, the algorithm initially divides the processes into pairs and one process from each pair sends its subtotal to the other, and the other compares the recieved one with its own subtotal and decides which is the minimum. The same procedure is applied repeatedly in several stages until the whole sum is obtained in one process. This part of the algorithm is illustrated in Figure 1.

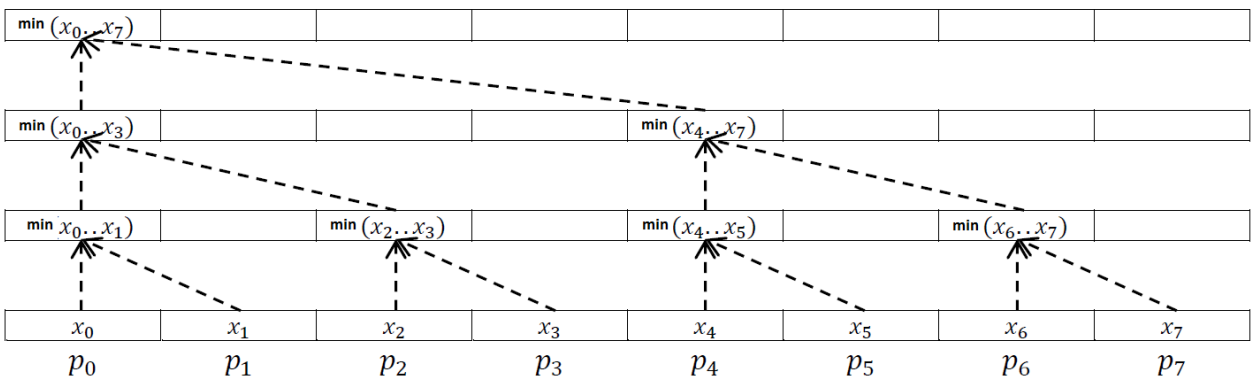

Figure 1: Overwiev of the algorithm. x_i denotes the minimum found by p_i.

Tasks:

1. Comment on the complexity of the above method. What is the improvement over an unefficient method that simply compares all the elements in a process sequentially?

2. Implement the above algorithm using MPI_Send and MPI_Recv

3. Write a code that uses MPI_Reduce method to determine the minimum of the results of all processes.

4. Your programs should print the following two lines:

   The minimum value

   The total (wall clock) time in seconds (double precision) spent for the computation and communication using MPI_Wtime (excluding the time for initialisation of the array, and distributing it among the processes).

5. Run the two implementations above with the same nonzero double precision array using number of MPI processes 1, 2, 4, 8 and 16 (running at most two nodes of NAR cluster). You should choose the array length to be large enough in order to have a meaningful timing.

6. Graph the timings of the two implementations as a function of number of MPI processes in the same plot. The horizontal axis corresponds to the number of processes, and the vertical axis corresponds to the time measurements. Add legends, axis-titles etc. to make the plot easily readable.

7. Based on the plots you draw, compare the scalibility of both methods.

8. Write a report based on these tasks. Note that **most** of the credit you earn for this and the following homeworks will be based on your **reports**. So, give importance to the reports. If you submit your homework without a report it is possible that you may **not** get any credits.

9. Submit hw1.tar including the **PDF** of your report, “reduce.c” and “sendrecieve.c” files. If you build the array outside your codes, and take those as inputs for your code, send your test array as well.

- Please note that zombie processes may pile up. So, try to control frequently if there are any, with *qstat* command, and kill them with *qdel* command to ensure that HPC will serve all the users properly.

- Do not leave your work to the deadline. Since all of you work on same HPC, on the last day it may work very slow and you may not be able to test your code efficiently.

- Note that implementing MPI macros does not mean your code is parallel. You have to design your code so that it works really in parallel. For those who submit a code that does not work in parallel will get no credits from this homework.

- Cheating policy of our department applies also for this course. Please submit your own work, unless you want to get zero points from homework and being subjected to a disciplinary action.