Parallel Processing

Programming Assignment 1

1 Goals

This assignment will introduce you to run very simple MPI (MPI-2, MPI-3, or MPICH) programs (use C) on the EXTREME cluster. You will familiarize yourself with simple MPI calls, and how to execute programs on clusters. Your goal is to create a large array A of n numbers at one node (let us call it the *source node*), distribute blocks of the array across the processors, and find the global sum of the numbers in A.

Your goal is to solve this FIND_SUM problem using at least these two different logical topologies:

1. a ring of k processors
2. a hypercube of $k = 2^d$ processors

You can also try 2-D mesh with wraparound, and a tree.

Try different ways of distributing A across the processors, and different ways of collecting the results, for each of the topologies experimented with. You are allowed to use MPI group communication primitives (for example, for scatter and gather). It is suggested that the elements of A be chosen from a small domain such as $\{+1, 0, -1\}$.

Please observe the following points.

1. Remember to start early, because EXTREME is a shared machine and there are users from all over campus. You submit jobs in batch mode, and may not get the result immediately. (During bad times, it takes a day or more for completion of even small jobs).
2. First get the basic programs correctly running. Then you can experiment with varying parameters, and more interesting variants of decomposing the workload and of MPI primitives.

2 Experiment

1. For each logical topology, write running code for each of the parallel formulations you design. You may use divide and conquer, or any other strategy for splitting your input array of numbers. Similarly for collecting and aggregating the partial results into a single result.

   You are encouraged to experiment with multiple approaches for each topology. For example, for the ring, you can collect the locally computed sums directly at the source. Or while collecting the local sums, each intermediate node can calculate the sum of the local sums it receives and its own local sum. Using different approaches will allow you to use different MPI primitives.

2. Measure the *timings* for each of your approaches.
   If time permits, you can experiment with different values of n and k. The actual parameter ranges you vary will depend on various factors such as the workload on and the availability of the EXTREME nodes, OS limits etc..

3 Input and Testing

1. The program will take as inputs: n, k, p. Here, p is the number of physical processors (nodes) you use on EXTREME. Whereas k is the number of logical processors (cores). (In practice, you may

be constrained by the *sysadmin policy* which does not allocate more than 7 nodes of 16 cores each.) (When simulating more logical processors than the available cores ($7 * 16$ in the above example), multiple logical processors need to be mapped to each available logical processor.) The array A is created by one process only. You can use a random number generator or any other mechanism.

2. While you debug your code, please work with small data sets, small number of logical nodes, and few physical processors i.e., small n, small k, small p.

4 Output

1. Prepare a **detailed report (PDF) describing your experiments**. The report must have the following sections/information.

   (a) **Formulations:** Describe each of the formulations of your parallel FIND_SUM in about 1 page each. List clearly the MPI calls you used in the implementation of each formulation.

   (b) **Parameter ranges:** For each of the formulations, describe the ranges of the parameters you experimented with.

   (c) **Results:** Give the tables and plot the graphs showing the *timing* variations for each of the formulations, for the ranges of parameters you used. Remember to use appropriate scales for the graphs. For each setting of parameters, take several readings and plot the average (and standard deviation if possible).

   (d) **Analysis:** Analyse the results. Give your interpretation and reading of the results for each formulation. You should address questions such as the following:

   - Which formulation is better? under what (or all) circumstances? Why?
   - Can you measure the computation time versus communication time? How do the formulations compare with each other with respect to this breakup of the time overhead?

   Explain why you observe what you observe. In particular, any anomalous observations should be explained.

   (e) **Lessons:** What insights you learned from this assignment.

5 Grading

The problem is reasonably well-formulated but the experimental approach is open-ended.

A non-running program may get zero credit. Your assignment will be judged on how *comprehensively* and *methodically* you have designed and run the experiment, and reported on the results. Your insights into the analysis of the results will also be considered in judging the assignment.

6 Reference Material/Chapters

Chapters 4 and 6 from the text. The sample MPI programs linked via the class web site, and any other source you can find on the web.