|  |
| --- |
| **Objective** |
| The objectives of this assignments are:   * Parallelize a given problem using MPI * Gain further familiarity with MPI’s blocking communication functions |

1. This program uses exactly 2 processes to compute the number of factors for a given number.

2. A set of data files numbers\_100.txt, numbers\_1000.txt, and numbers\_10000.txt files to be used for testing and performance verification.

## Compiling:

Use the standard mpicxx command-line to compile this program as shown below:

|  |
| --- |
| $ mpicxx –std=c++11 –g –Wall program\_name.cpp –o program\_name |

# Parallelize the application

## Description:

The objective of this project is to parallelize the prime number verification program. The objective is to parallelize the program to use 2 or more processes (recollect that in the Exercise #13 only 2 processes were used, one for manager and another one for worker) while adhering to the following requirements (**pay attention to all of these requirements**):

1. Your final solution source code must be called program\_name.cpp.
2. Given *n* parallel processes to use, your program must use one of them to serve as a manager and the remaining *n*-1 processes as workers. The operations to be performed by a worker and a manager are described further below for your immediate reference.
3. The worker processes must use the existing getFactorCount() method (from the supplied Exercise13.cpp) and associated logic to determine if a given number is a prime number.
4. All interactions between worker and manager must use only MPI’s blocking communication functions. Non-blocking communication calls must not be used in this homework exercise.
5. The manager and worker may exchange only one number (int or long long int) per send or receive call. This feature is already setup in the program and you are expected to retain this setup in your final solution as well.
6. Only the manager process is permitted to perform any I/O (such as: file reading and displaying results on the console) operations.
7. All testing must be performed with the supplied numbers\_100.txt, numbers\_1000.txt, and numbers\_10000.txt files.

## Required behaviors of Manager and Worker:

The following operations are the minimal set of operations that a Manager and a Worker process are expected to perform. The processes may perform additional tasks in order to streamline the program to extract additional performance.

* + The manager process must perform the following tasks:
    - Repeatedly reads numbers (long long int) from a text file (name of the file would change depending on the test).
    - For each number read, the manager process:
    1. Sends the number to one of the worker process using MPI\_Send
    2. Obtains the number of factors from the worker using MPI\_Recv (possibly as and when responses are available).
    3. Displays number of factors on the screen.
    4. Finally it sends number -1 to all the workers indicating work is done.
  + A worker process must perform the following tasks:
    - Repeatedly reads only a single number from the manager and stops when manager sends -1 as the number. For each number:
    1. It computes the number of factors for the number using getFactorCount()
    2. Sends the number of factors back to the manager

## Reporting:

In order to do this task, the following procedure must be adopted:

1. Ensure your implementation operates correctly.
2. Download and save the attached Report.doc to your local computer. You should save/rename this document using the naming convention Report.doc.
3. Run your programs using the supplied numbers\_100.txt, numbers\_1000.txt, and numbers\_10000.txt files and measure the runtime of your implementation to update Table 1 in the report document. Use the supplied Exercise13.cpp as the serial implementation and measure its runtime
4. Now based on the data in Table 1, compute the efficiency values (see Table 5.1 on Page 211 of Textbook for reference) and update Table 2 in the report document.
5. Using the data from Table 1 and Table 2 draw inferences about the scalability and performance of your parallel implementation and record them in Table 3.
6. Submit your report document along with your program\_name.cpp.

## Tips & Suggestions:

* Try to keep the workers as busy as possible.
* In order to keep workers busy one possible approach is to:
  + First distribute one number to all the workers.
  + Whenever a worker responds with a result assign another number to the worker.
* As a general rule of thumb, always compile your code with –Wall (report all warnings) flag in the g++ command line. It will minimize runtime errors in your code and save you a lot of aggravation on the long run.
* Use valgrind to verify operation of your program. When running in parallel, you may use valgrind along with mpiexec command as shown below to troubleshoot memory errors.
  + In case you don’t have valgrind already setup, first setup valgrind to be automatically loaded each time you login using the following command:

|  |
| --- |
| $ module initadd valgrind |

* + Log out and log back into the cluster for the above command to take effect
  + Run mpiexec along with valgrind using the following command:

|  |
| --- |
| $ mpiexec –n 3 valgrind ./program\_name |

Don’t forget to compile your programs with –g option when using valgrind. The output from valgrind will be the corresponding output files if you are running batch mode.