CS 362

Programming Languages
Team Project 2

1. (25%) Benchmark the second circuit satisfiability program for 1, …, 10 processors with printing disabled. For each number of processors, determine the average execution time after five runs and compare it with the theoretical execution time (time for one processor / *p*). Plot the results using a line graph.

   Use program SATI.c on Canvas. Do not submit the code. Submit the summary and interpretation of the observed results.

2. (25%) A small college wishes to assign unique identification numbers to all of its present and future students. The administration is thinking of using a identifier but is not sure that there will be enough combinations, given various constraints that have been placed on what is an "acceptable" identifier. Write a parallel MPI program to count the number of different six-digit combinations of the numerals 0-9, given these constraints:

   - The first may not be a 0.
   - Two consecutive digits may not be the same.
   - The sum of the digits may not be 7, 11, or 13.

   Benchmark the program for 1, …, 10 processors with printing disabled. For each number of processors, determine the average execution time after five runs and compare it with the theoretical execution time (time for one processor / *p*). Plot the results using a line graph.

   Submit: a) the code; b) the summary and interpretation of the observed results.

3. (25%) Implement a new version of the Sieve of Eratosthenes program in MPI, replacing the broadcast step with a pipeline of sends and receives. In other words, instead of process 0 broadcasting the next prime to all the other processes, it sends the next prime to process 1, which receives the value, and it sends it to process 2, and so on. Process 0 does not perform a receive, and the process of highest rank does not perform a send. For each prime found, the maximum number of communication steps per process is this way reduced from log *p* to 2, where *p* is the number of processors.

   Benchmark the original program (Sieve.c on Canvas) and the new program for 1, …, 10 processors with printing disabled. The theoretical execution time is not needed here. For each number of processors, determine the average execution time after five runs. Plot the results using a line graph.

   Submit: a) the code; b) the summary and interpretation of the observed results.

4. (25%) Write a parallel MPI program that has a hard-coded matrix containing the initial state of Conway's game. It should play the Game of Life for *j* iterations, printing the state of the game after every iteration. The parameters *p* (the number of processes) and *j* are command-line arguments. Use a 2D Checkerboard Block Decomposition.

   For a fixed value of *j*, benchmark the program for 1, …, 10 processors with printing disabled. For each number of processors, determine the average execution time after five runs and compare it with the theoretical execution time (time for one processor / *p*). Plot the results using a line graph.

   Submit: a) the code; b) the summary and interpretation of the observed results.

   About Conway's Game of Life
   Conway's Game of Life is a cellular automaton that is played on a 2D square grid. Each square (or "cell") on the grid can be either alive or dead, and they evolve according to the following rules:

   - Any live cell with fewer than two live neighbors dies (referred to as underpopulation).
   - Any live cell with more than three live neighbors dies (referred to as overpopulation).
   - Any live cell with two or three live neighbors lives, unchanged, to the next generation.
   - Any dead cell with exactly three live neighbors comes to life.

   The initial configuration of cells can be created by a human, but all generations thereafter are completely determined by the above rules. The goal of the game is to find patterns that evolve in interesting ways. Conway's Game of Life is Turing-complete (computationally universal).
   Details: https://conwaylife.com/ .

How to collaborate: Do not simply distribute the problems among the team members. Each of you should try to solve all problems on your own. Then, meet and discuss the solutions and only submit the best solution for each problem. This is how you will learn best.