Parallel 15 Puzzle Solver

The 15-puzzle is a well-known brainteaser. The goal is to unjumble a set of squares by moving tiles into the available gap. As computer scientists, we realize that solving the 15-puzzle by hand is time- consuming and tedious. As such, many search strategies have been applied to this problem. In this problem, we give you a 15-puzzle solver based on the well-known A* search algorithm. Your task is to parallelize the program so it can run on multiple processors.

It is helpful to understand a few details of A* search. The search itself is brute force: we try all moves, then all successors to those moves, etc. A* uses a strategy called *iterative deepening*. At each iteration, we consider all solutions of size N. If no solution is found, then we move on to consider solutions of size N+1. This sounds wasteful, and it is, but not in a way that affects the asymptotic runtime of the algorithm. More details are available online.

Step 1: Multi-threaded Puzzle Solver

Your first task is implement a multi-threaded version of the 15-puzzle solver. The number of threads to spawn is defined by an argument to the program:

```
./FifteenPuzzleSolver.cpp n
```

Where n is the number of threads. At present, any argument other than 1 causes the program to crash. (no argument defaults to 1 thread). Obviously, it is not enough to simply spawn threads: the threads should do the work of solving the 15-puzzle. In the benchmark step (described below), you will verify whether the threads had any effect.

Note that the generation of board is random. This is good, because it allows for repeated testing of the code.You may assume that a `FifteenPuzzleSolver` instance is only used once. Also, it is permissible to have extra "helper" threads, as long as they aren't doing any useful work towards solving the puzzle.

Step 2: Benchmark

Your next task is to benchmark the 15-puzzle solver using a variable number of threads. Your writeup should include a graph with threads on the x-axis and latency on the y-axis. The number of threads should range from 1 to roughly 10.

In addition, your writeup should provide an explanation of this graph. In particular, you should explain your results.

If you are running on a shared machine, make sure your results are not skewed by someone else's experiment. One way to guard against this is to run the UNIX `top` application, which shows which processes are consuming CPU resources.

Step 3: Implement another solution

You can implement a different approach to parallelizing the 15-puzzle. You should compare the performance of your second approach with the first, and provide an explanation of the differences.

To get the most points, your second technique should be both interesting and different from your first approach. Changing data structures is not very interesting. Also, we are primarily concerned with parallelizing the existing algorithm.

The write-up is critical for this. A brilliant idea with a bad write-up will not score well.