CSE 332: Data Structures and Parallelism — Winter 2019

P3 Write-up

P3 Due Date: Wednesday, March 13

Only ONE person from your group should submit the write-up to Gradescope. Please make sure that you select corresponding pages for each question when you are submitting your write-up to Gradescope. You should also add your partner to your group on Gradescope after you submitted your write-up.

Project Feedback

(1) Project Experience

Answer the following questions about your experience doing the project.

- What was your favorite part of the project? Why?
- What was your least favorite part of the project? Why?
- How could the project be improved? Why?
- Did you enjoy the project? Why or why not?

(2) Chess Server

Answer the following questions about your experience using the chess server.

- When you faced Clamps, what did the code you used do? Was it just your jamboree? Did you do something fancier?
- Did you enjoy watching your bot play on the server? Is your bot better at chess than you are?
- Did your bot compete with anyone else in the class? Did you win?
- Did you do any Above and Beyond? Describe exactly what you implemented.

Experiments

NOTE: Feel free to tweak your divide cutoff variable to beat clamps, but keep it constant for your experiments.

(3) Chess Game

Hypotheses: Suppose your bot goes 3-ply deep. **How many game tree nodes do you think it explores** (we're looking for an order of magnitude)

- ... if you're using minimax?
- ... if you're using alphabeta?

Results: Run an experiment to determine the actual answers for the above. To run the experiment, do the following:

- Run SimpleSearcher against AlphaBetaSearcher and capture the board states (fens) during the game. To do this, you'll want to use code similar to the code in the testing folder.
- Now that you have a list of fens, you can run each bot on each of them sequentially. You'll want to slightly edit your algorithm to record the number of nodes you visit along the way.

- Run the same experiment for 1, 2, 3, 4, and 5 ply. And with all four implementations (use ply/2 for the cut-off for the parallel implementations). **Make a pretty graph of your results and fill in the table in the write-up template as well.** NOTE: Your result should be the average number of nodes visited across all fens for each bot & depth.

Conclusions: How close were your estimates to the actual values? Did you find any entry in the table surprising? Based ONLY on this table, do you feel like there is a substantial difference between the four algorithms?

Optimizing Experiments

THE EXPERIMENTS IN THIS SECTION WILL TAKE A LONG TIME TO RUN. To make this better, you should use Google Compute Engine. **PLEASE REMEMBER TO TURN OFF YOUR GCE INSTANCES WHEN YOU ARE NOT USING THEM.**

- Google Compute Engine lets you spin up multiple instances. You should do this to run multiple experiments in parallel.
- DO NOT run multiple experiments in parallel on the same machine. This will lead to strange results.
- It's not strictly required to run experiments on multiple machines, but the write-up will take a lot longer if you don't do this.

NOTE: Because chess games are very different at the beginning, middle, and end, you should choose the starting board, a board around the middle of a game, and a board about 5 moves from the end of the game. The exact boards you choose don't matter (although, you shouldn't choose a board already in checkmate), but they should be different.

(4) Sequential Cutoffs

Experimentally determine the best sequential cut-off for both of your parallel searchers. You should test this at depth 5. If you want it to go more quickly, now is a good time to figure out Google Compute Engine.

Plot your results and discuss which cut-offs work the best on each of your three boards.

(5) Number of Processors

Now that you have found an optimal cut-off, you should find the optimal number of processors. You MUST use Google Compute Engine for this experiment. For the same three boards that you used in the previous experiment, at the same depth 5, using your optimal cut-offs, test Parallel Minimax and Jamboree on a varying number of processors. You shouldn't need to test all 32 options; instead, you may want to try something like a ternary search to help you find the best option without trying all possible options (e.g. try 2 processors, 16 processors, and 32 processors. Then try the middle point of the better performing range, etc.). You can tell the ForkJoin framework to only use k processors by giving an argument when constructing the pool, like so: `ForkJoinPool POOL = new ForkJoinPool(k);`

Plot your results and discuss which number of processors works the best on each of the three boards.

(6) Comparing the Algorithms

Now that you have found an optimal cut-off and an optimal number of processors, you should compare the actual run times of your four implementations. You MUST use Google Compute Engine for this experiment (Remember: when calculating runtimes using timing, the machine matters). At depth 5, using your optimal cut-offs and the optimal number of processors, time all four of your algorithms for each of the three boards.

Plot your results and discuss anything surprising about your results here.

Beating Traffic

(7) Traffic

In the last part of the project, you made a very small modification to your bot to solve a new problem. We'd like you to think a bit more about the formalization of the traffic problem as a graph in this question.

- To use Minimax to solve this problem, we had to represent it as a game. In particular, the "states" of the game were "stretches of road" and the valid moves were choices of other adjacent "stretches of road". The traffic and distance were factored in using the evaluation function. **If you wanted to use Dijkstra's Algorithm to solve this problem instead of Minimax, how would you formulate it as a graph?**
- These two algorithms DO NOT optimize for the same thing. (If they did, Dijkstra's is always faster; so, there would be no reason to ever use Minimax.) **Describe the pros and cons of each of the algorithms. When will they output different paths?**