Spencer Gabhart

Samuel Bland

Distributed System Project

Parallel Chess AI

The application that uses parallel computing that we have chosen to examine and attempt to implement is IBM’s Deep Blue project. This project was IBM trying to explore using parallel processing to solve complex problems. The complex problem they decided to handle was Chess. Chess is a very difficult game and requires players to think many moves ahead in an attempt at grasping victory, so they used Deep Blue to analyze the many possible paths and moves to win a chess game by computing the possible outcomes of its moves and choosing the best that would lead to victory. When facing the world chess champion in 6 games in 1997 where it won two games, lost one, and the other three ended in draws declaring the computer winner of the match. The previous year Deep Blue had lost against the chess grandmaster and it in 1997 it came back with even more computing power and its adaptability to what led to its defeat the prior year which was the chess grandmaster changing strategies mid-game. The program was capable of exploring up to 200 million chess positions per second and 60 billion moves within 3 minutes which is the time allotted to make a move in classical chess. Deep Blue’s software was written in C and ran under the AIX operating system while utilizing the IBM SP Parallel System called MPI. Deep Blue also had a total of 256 processors working together. The search system it uses is called selective extensions and it allows the computer to selectively choose certain paths to follow in the search process which eliminates paths that are obviously not worth following. The computer itself will look at its current position and begin to search moves based on that position and will distribute its search for the most valuable move amongst different computers at the same time.

Given that Deep Blue was an insane project by itself we decided to try to just do a parallelized chess AI without going too in depth to go as far as beat a chess grandmaster. To accomplish this, we decided to use java and java’s ExecutorService to handle working with a pool of threads. In the chess MinMax algorithm we follow the methodology of making the move that will cause the opponent to have a pool of moves that are scored less than the pool of moves the AI will have if it makes the move. We aim to minimize the score of the opponent while maximizing the AI’s score. Evaluating all of the possible moves to determine this takes time so this is the main part of where we use the ExecutorService to spit off threads evaluating moves and their values. We compared the running time of the AI using threads against one that just uses a single thread and saw a major discrepancy in performance between the two with the threaded one performing a lot better.

Creating a chess AI is a difficult process and improving a chess AI’s running time is even more difficult. We had a lot of problems throughout the assignment and given that we switched from another project to do this one, we had a lot to do with very little time to do it. We had originally wanted to use some other parallel extensions such as trying to use MPI or OpenMP as well as some others. Unfortunately, due to time we had to focus on what we started with and completed, but we evaluated how we could accomplish using other parallel libraries and determined while possible we just didn’t have the time. To add to our time constraints, we both had very busy schedules with tests and then finals week as well as general life requirements. We are proud of what we have accomplished and may look to continue working on the project to see how we can improve it further and how we can fully implement other parallel libraries.

References

*Deep Blue Overview*. (n.d.). Retrieved from IBM: https://www.research.ibm.com/deepblue/meet/html/d.3.html

*How Deep Blue Works*. (n.d.). Retrieved from IBM: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

*Meet Deep Blue*. (n.d.). Retrieved from IBM: https://www.research.ibm.com/deepblue/meet/html/d.3.2.html