Parallel Othello AI

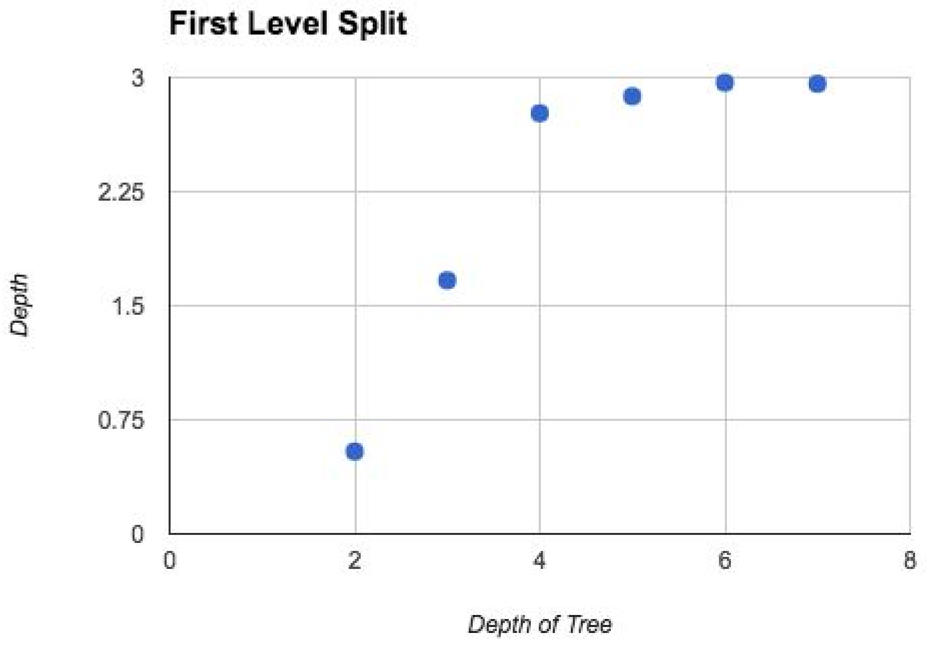
We created an artificial intelligence for the game Othello using the minimax and alpha-beta algorithms. Our goal was to find parallelization within the AI for both minimax and alpha-beta and try to achieve a speedup of as close to 4x as possible on a 4-core computer.

Mini-Max

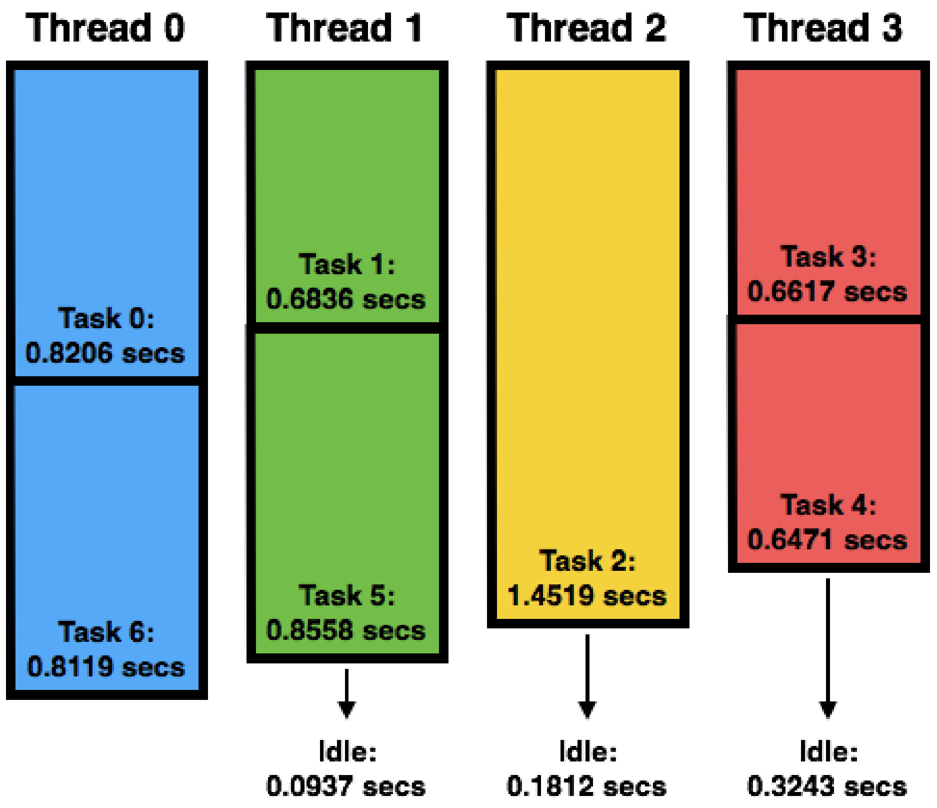
Level 1 Split

The first method of parallelization for minimax was splitting it across the first level of the minimax tree, the legal moves possible given the initial board. This parallelization was accomplished using an OpenMP for loop, with 4 threads for the 4 cores on the GHC machines.

The speedup, as shown in the graph, was up to 3x the sequential minimax algorithm running on the Gates machines. We achieved closer to this speedup the more work there was to be done, which is when the depth of minimax was greater.



To see why we weren't achieving the optimal speedup of 4x we timed out the different branches of a few of the runs. This graphic shows the work imbalance between the tasks, as the work for a given branch would be determined by how many legal moves could be found down that path, which was variable. Here, thread 3 is idle for 20% of the time.

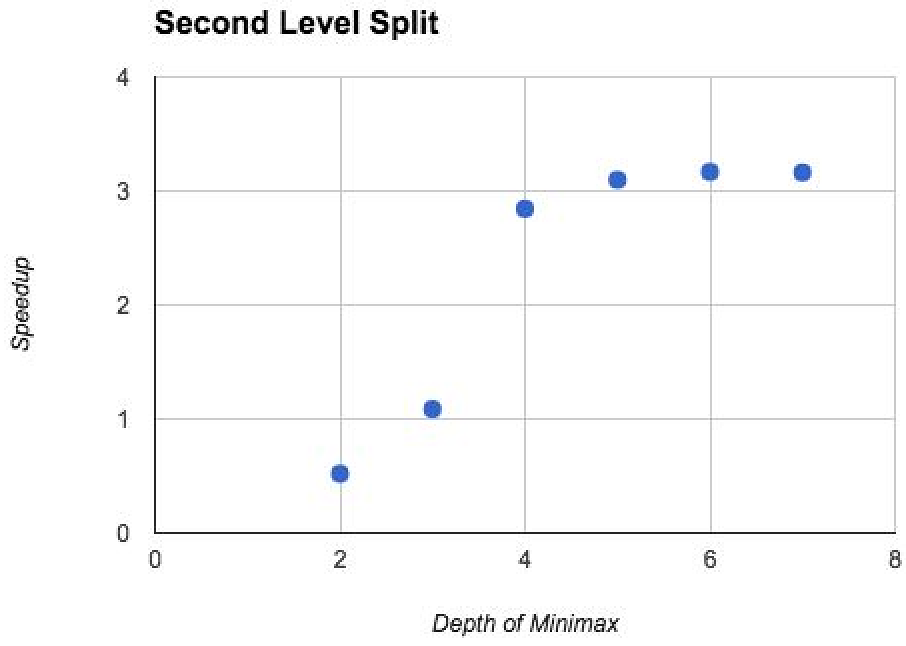


Level 2 Split

To combat the workload imbalance, we wanted to split the work into smaller chunks, by partitioning it at the second level of the minimax tree. This chart shows the time taken by the split up tasks, using the same previously imbalanced example from before. It is clear that there are many more tasks in smaller chunks.

The first way we did the second level split was by having each initial legal moves' children run in parallel and then synchronize. This led to a worse speedup than the level 1 split, with only up to a 2.8x speedup, as shown in the graph below.

Thus, we decided to launch all of the smaller level 2 tasks in parallel together, once again using dynamic scheduling with 4 threads for the 4 cores of the GHC machines. This lead to a greater speedup of about 3.2x, as shown in the graph below.

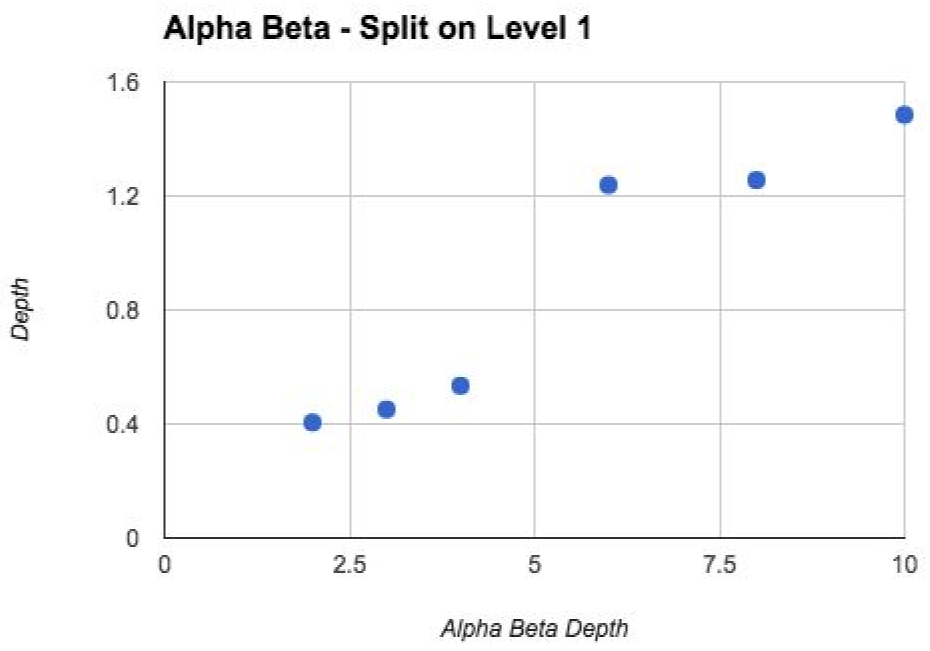


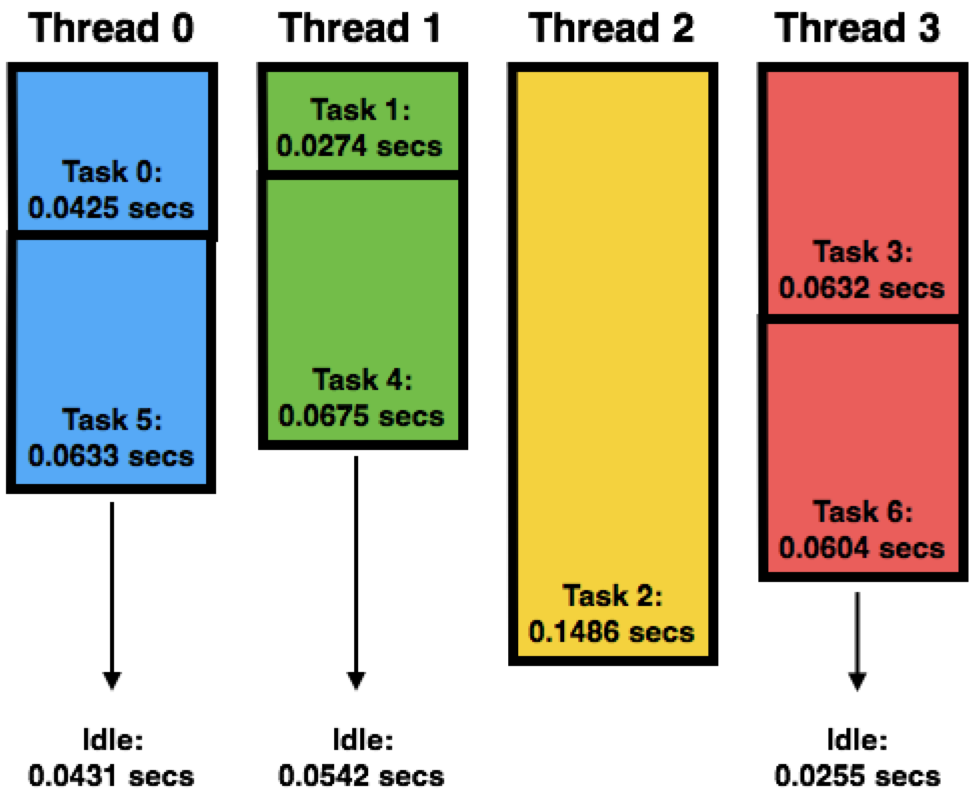
Alpha Beta Pruning

The second way of creating a game AI based on a decision tree is by using alpha beta pruning, which determines which branches of a minimax tree to stop calculating.

Level 1

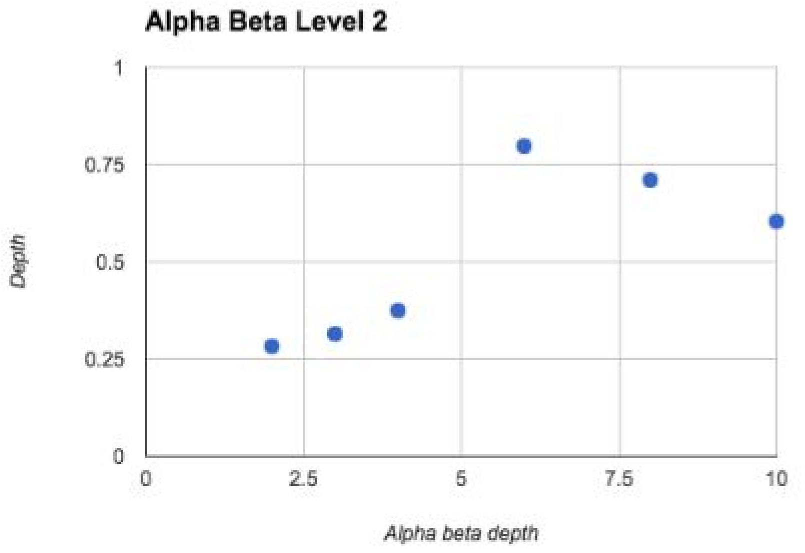
Our first naive implementation of parallel alpha beta was simply to use alpha beta pruning instead of minimax on the first level of legal moves. This led to a 1.5x speedup, but it created even more work imbalance from before. Using the same initial board from before as an example, here is a diagram which depicts the workload imbalance. As you can see both thread 0 and thread 1 are idle ⅓ of the time.


Alpha Beta - Split on Level 1
Depth
1.6
1.2
0.8
0.4
0
0
2.5
5
7.5
10
Alpha Beta Depth

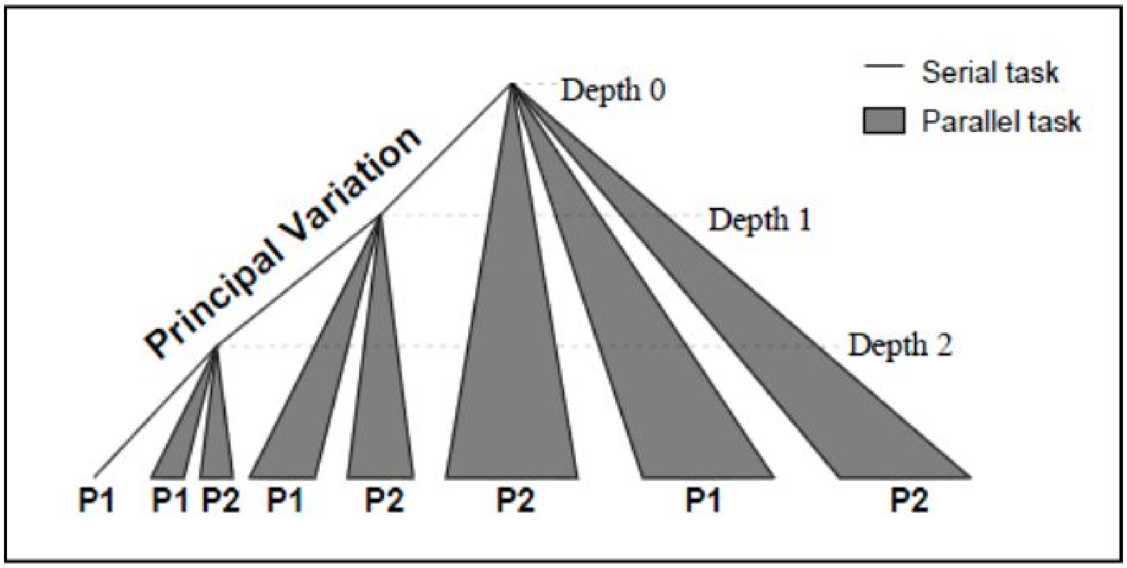

Thread 0
Thread 1
Thread 2
Thread 3
Task 0: 0.0425 secs
Task 5: 0.0633 secs
Task 1: 0.0274 secs
Task 4: 0.0675 secs
Task 2: 0.1486 secs
Task 3: 0.0632 secs
Task 6: 0.0604 secs
Idle: 0.0431 secs
Idle: 0.0542 secs
Idle: 0.0255 secs

Level 2

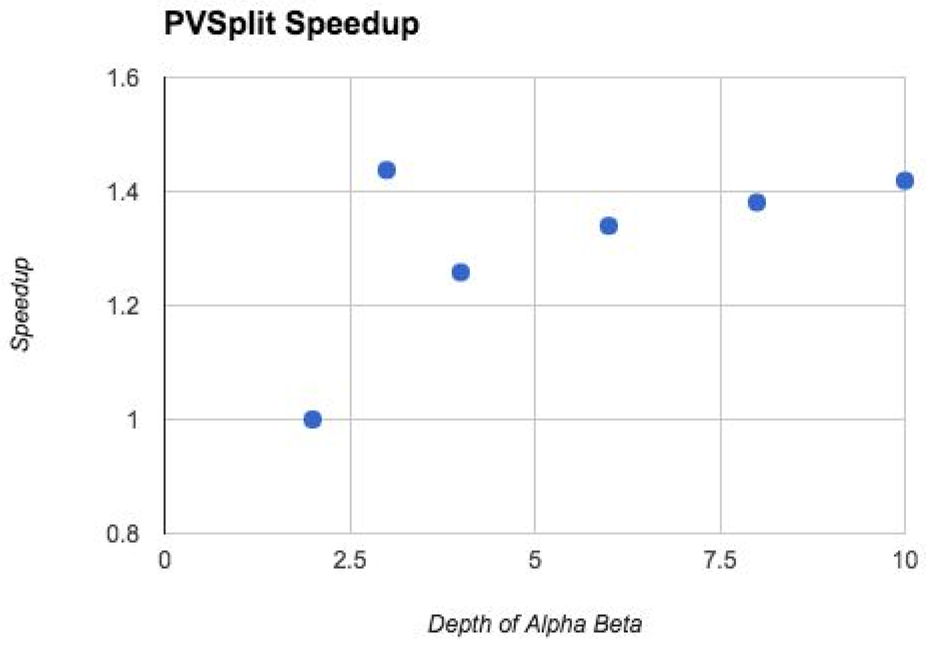
We then tried to combat the imbalance in the same way which benefited the minimax theorem, by splitting at the second level of legal moves, and this led to even worse times than the sequential algorithm. This is because the work was split into so many independent branches and these branches did not have as much information about the rest of the tree so they were not able to do as much as pruning.



PVSplit



We then decided to find a better way to implement the parallel version of alpha-beta and so we turned to a paper which described the PVSplit algorithm. This algorithm essentially allows for parallelization with compromises in total pruning as well as sequential execution. We implemented this and only received a 1.5x speedup. We hypothesize that this is because the branching factor for othello is large and so many of the branches were done in parallel without knowing what to be pruned and so there was a lot of sacrifice made in terms of actually pruning.



Next Steps

After discussing with Professor Railing in our project presentation, the next step to improve speedup on this project would be to dynamically create more work. There was a static splitting of work in our parallelization methods, and a more effective use of the cores would be to assess if a particular task would run for longer, and in that case, dynamically split it into multiple tasks.