Me

I am Yiwen Song, a 4th year undergraduate studying Computer Science. I play online poker in my free time and I will be working in quantitative finance next year.
From this class I hope to get some experience with parallel HPC usage and see cool things that go really fast. I was once told that CS267 is the computer science equivalent of *Fast and Furious* (great movie, by the way), whatever that means.
I am an undergrad with zero research experience, but I'd want to research something related to AI or data science.

Application of Parallel Computing: Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS) is a decision-making method, most famously applied to the game Go. The goal of this algorithm is to maximize the utility of an agent at the end of some series of decision-making events. An iteration in MCTS is as follows:

1. We select a leaf in our previously expanded search tree.
2. Expand that leaf and select a new leaf from the expansion (*C*).
3. Simulate the remainder of the game from *C*.
4. Update the average payout of all ancestors of *C*.

In step 1, it is important that we balance exploring the new nodes with exploiting the known winningest nodes. Being too explorative creates a model of an opponent which is completely random, while being too exploitative limits the scope of our search and we may completely fail to explore better options.

One of the downfalls of a brute-force-like algorithms like Alpha-Beta search is that it heavily relies on its evaluation functions for high branching-factor games like Go and Arimaa; because of the large branching factor, the time or memory restrictions of exhaustive search agents limits the depth of the search to be rather shallow, which then means that they require an evaluation heuristic to determine which moves are good and which are not. For MCTS, however, an evaluation function is not required as the evaluation of each move are simply simulations to the end of the game. While some possibilities may remain unexplored, MCTS is allowed to be greedier with parts which have historically positive results. This means that the MCTS will have a better predictive model for a few moves, rather than the Alpha-Beta's worse predictive model for all moves.

An MCTS simulation is purely random, and thus there is no dependencies to any existing data structures until the end of the simulation. Because of the independent nature of the MCTS simulations, the obvious parallelization of the process is at the simulation, however, the problem lies in what we should do for the tree, which is not a parallel process.

The first way of solving this is to have one thread select and expand a leaf, but have multiple threads performing the simulations. At the end of all of these simulations, the data is collected and propagated back to the root. The issue with this approach is the fact that each iteration of simulation must be equal to the tree traversal time plus the maximum of the simulation times, which means that a lot of the time the threads are idling and doing nothing.

The second method of parallelization is to simply create many trees, with each tree traversed by a single thread. At the end of the simulations, all threads add up their results and create a tree and make a decision based on the joint results of all the simulations.

The third method is to parallelize selection and expansion along with the simulation. Because the selection process requires the reading of the search tree, this method requires a mutex for the tree. At the end of each simulation, the threads then acquire the tree lock and update it with the results of the simulation. Because of the mutex required for this arrangement, the speedup with *n* threads will actually be much less than *n* times, since a significant amount of time spent by a thread will be inside the tree rather than in simulation.

This particular problem is very difficult to scale for a variety of reasons. First, if threads are allowed to communicate, there is a significant amount of time for each thread that must be spent in sequential land. A rough estimate of a MCTS thread's time traversing the tree is 25%, which means that the maximum speedup is still only 4 times as fast as the sequential process. The second aspect of this is that, if threads are not allowed to communicate, it is very likely that work will be repeated, especially for the earlier nodes. If we were to try to scale here, we will encounter diminishing returns on each additional thread, since so many of them would be running the same simulations as each other.

References

Guillaume M.J-B. Chaslot, Mark H.M. Winands, and H. Jaap van den Herik, "Parallel Monte-Carlo Tree Search" *Proc. Comput. Games, pp. 60-71, 2008.*