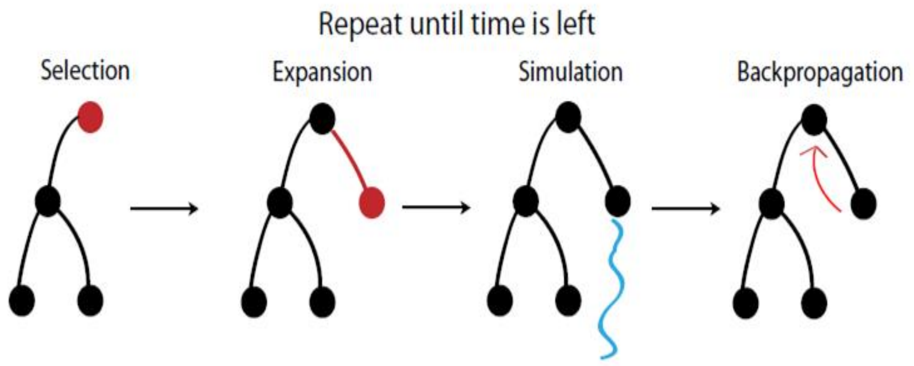
Monte Carlo Tree Search Parallelization

By Jacques Zhang, Kartik Misra, Patrick Ghazal and Anna Bieber

Problem & Context

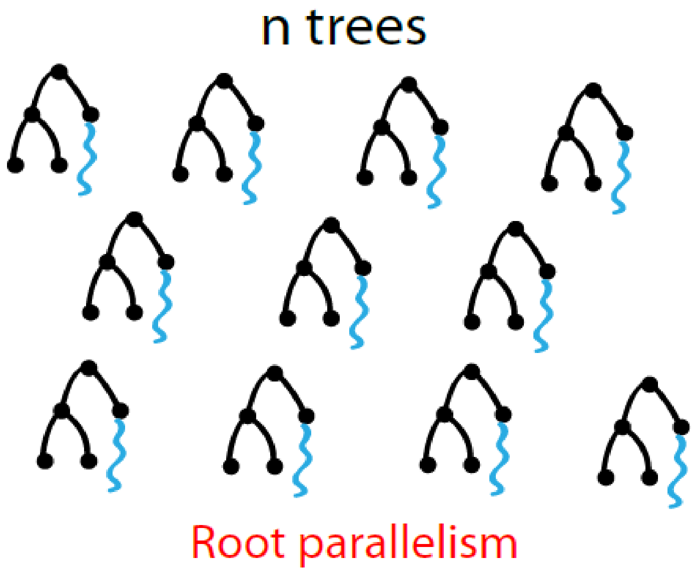
- Proof of concept of parallelizing MCTS.
- Research in MCTS has sharply risen due to great success with Go.
- MCTS uses a tree data structure to search and calculate outcomes of random simulations to find the most optimal option.

Monte Carlo Tree Search

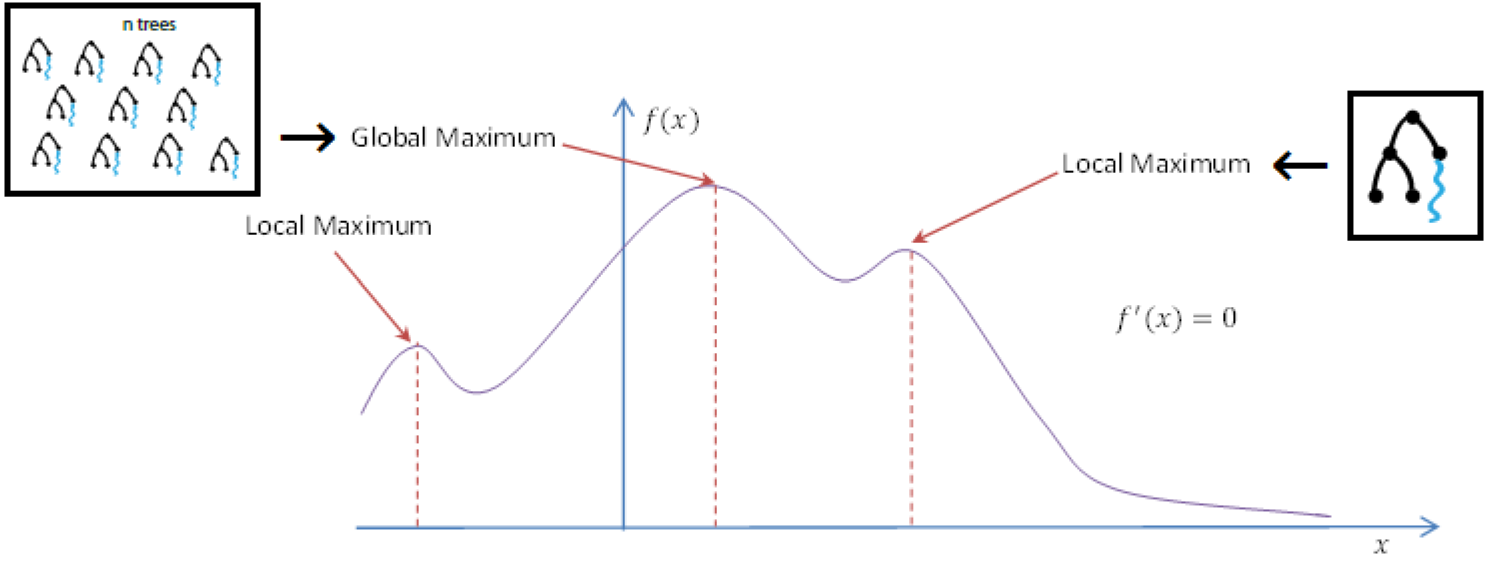


How are we parallelizing ?

- The concept of “Root Parallelism”.
- n Monte Carlo Trees that will each perform many iterations of the 4 phases in parallel.
- Increasing the chances of finding the true global max

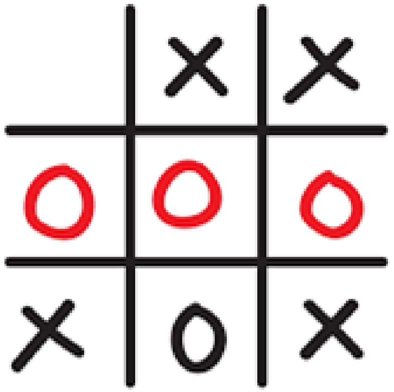


A simple overview

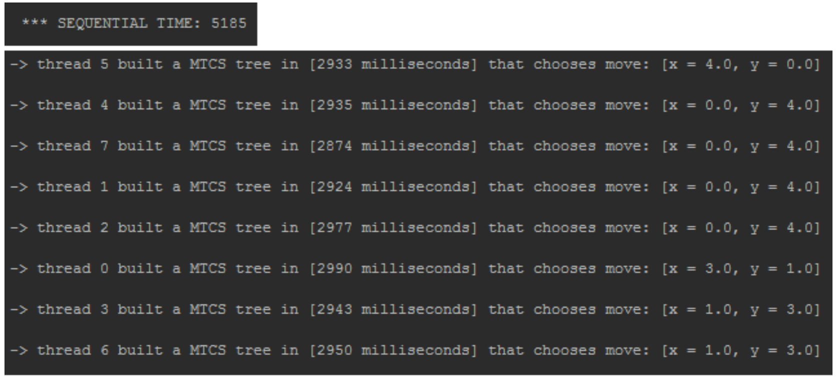


Design & Approach

- AI agent for TicTacToe Game
- CUDA parallelization attempt - GPU limitations
- Proof of Concept in Java with a java thread version of the parallel MC

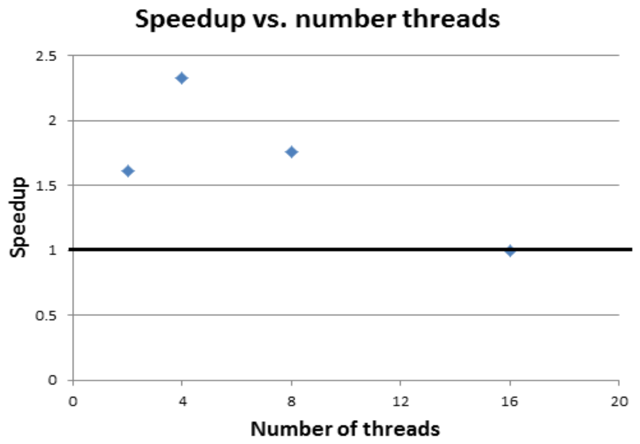


Results - Performance



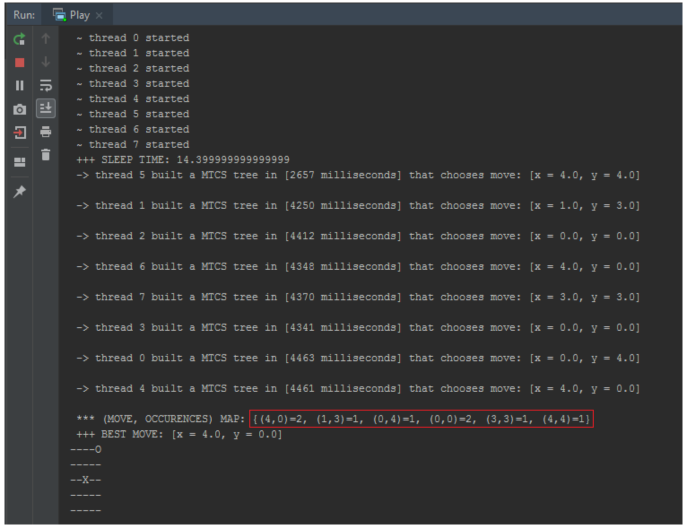
```
*** SEQUENTIAL TIME: 5185
```

```
-> thread 5 built a MTCS tree in [2933 milliseconds] that chooses move: [x = 4.0, y = 0.0]

-> thread 4 built a MTCS tree in [2935 milliseconds] that chooses move: [x = 0.0, y = 4.0]

-> thread 7 built a MTCS tree in [2874 milliseconds] that chooses move: [x = 0.0, y = 4.0]

-> thread 1 built a MTCS tree in [2924 milliseconds] that chooses move: [x = 0.0, y = 4.0]

-> thread 2 built a MTCS tree in [2977 milliseconds] that chooses move: [x = 0.0, y = 4.0]

-> thread 0 built a MTCS tree in [2990 milliseconds] that chooses move: [x = 3.0, y = 1.0]

-> thread 3 built a MTCS tree in [2943 milliseconds] that chooses move: [x = 1.0, y = 3.0]

-> thread 6 built a MTCS tree in [2950 milliseconds] that chooses move: [x = 1.0, y = 3.0]
```



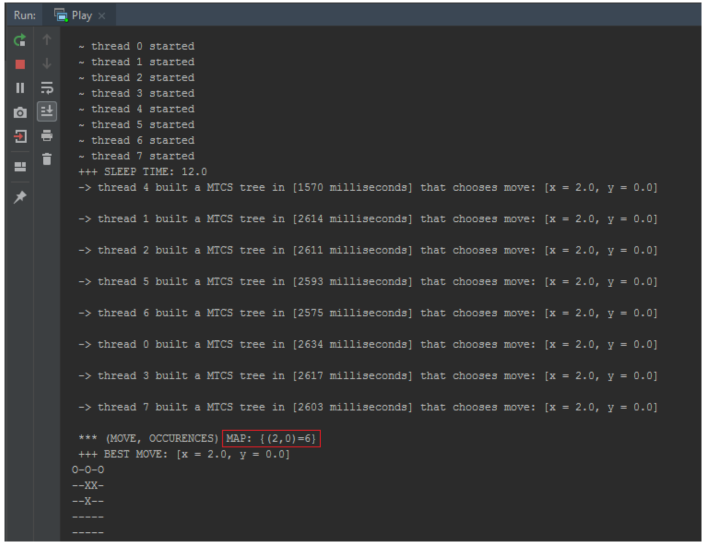
Constant variables:
- Require very first move from MCTS
- First move chosen by human is center cell
- 5*5 board

Results - Optimization



Run: Play

```
 ~ thread 0 started
 ~ thread 1 started
 ~ thread 2 started
 ~ thread 3 started
 ~ thread 4 started
 ~ thread 5 started
 ~ thread 6 started
 ~ thread 7 started
 +++ SLEEP TIME: 14.399999999999999
 -> thread 5 built a MTCS tree in [2657 milliseconds] that chooses move: [x = 4.0, y = 4.0]

 -> thread 1 built a MTCS tree in [4250 milliseconds] that chooses move: [x = 1.0, y = 3.0]

 -> thread 2 built a MTCS tree in [4412 milliseconds] that chooses move: [x = 0.0, y = 0.0]

 -> thread 6 built a MTCS tree in [4348 milliseconds] that chooses move: [x = 4.0, y = 0.0]

 -> thread 7 built a MTCS tree in [4370 milliseconds] that chooses move: [x = 3.0, y = 3.0]

 -> thread 3 built a MTCS tree in [4341 milliseconds] that chooses move: [x = 0.0, y = 0.0]

 -> thread 0 built a MTCS tree in [4463 milliseconds] that chooses move: [x = 0.0, y = 4.0]

 -> thread 4 built a MTCS tree in [4461 milliseconds] that chooses move: [x = 4.0, y = 0.0]

 *** (MOVE, OCCURENCES) MAP: {(4,0)=2, (1,3)=1, (0,4)=1, (0,0)=2, (3,3)=1, (4,4)=1}
 +++ BEST MOVE: [x = 4.0, y = 0.0]
----O
-----
--X--
-----
-----
```



Run: Play

```
 ~ thread 0 started
 ~ thread 1 started
 ~ thread 2 started
 ~ thread 3 started
 ~ thread 4 started
 ~ thread 5 started
 ~ thread 6 started
 ~ thread 7 started
 +++ SLEEP TIME: 12.0
 -> thread 4 built a MTCS tree in [1570 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 1 built a MTCS tree in [2614 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 2 built a MTCS tree in [2611 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 5 built a MTCS tree in [2593 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 6 built a MTCS tree in [2575 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 0 built a MTCS tree in [2634 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 3 built a MTCS tree in [2617 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 -> thread 7 built a MTCS tree in [2603 milliseconds] that chooses move: [x = 2.0, y = 0.0]

 *** (MOVE, OCCURENCES) MAP: {(2,0)=6}
 +++ BEST MOVE: [x = 2.0, y = 0.0]
O-O-O
--XX-
--X--
-----
-----
```

Thanks!

Any questions?