MCTS Parallelisation

Francois van Niekerk
Supervisors: Dr. Gert-Jan van Rooyen, Dr. Steve Kroon, Dr. Cornelia Inggs

Computer Go

The dominant algorithm of Computer Go is currently Monte-Carlo Tree Search (MCTS). One of the advantages of MCTS is that it can be parallelised. This project focused on the parallelisation of MCTS.

Objectives

The objectives of this project were to extend an existing MCTS implementation with:

- Pondering (thinking during the opponent's time)
- Multi-core parallelisation
- Cluster parallelisation

Design and Implementation

An existing MCTS implementation, Oakfoam, was extended with: pondering, tree parallelisation for multi-core systems, and root parallelisation for cluster systems. Boost C++ Libraries were used for threading and Open MPI was used for cluster communication.

Conclusions

Pondering worked as expected on 9x9 and 19x19. Multi-core parallelisation scaled up to eight cores on 9x9 and 19x19. Cluster parallelisation failed to scale well on 9x9, but scaled on 19x19 up to eight cores, where it achieved a strength increase equivalent to four ideal cores.

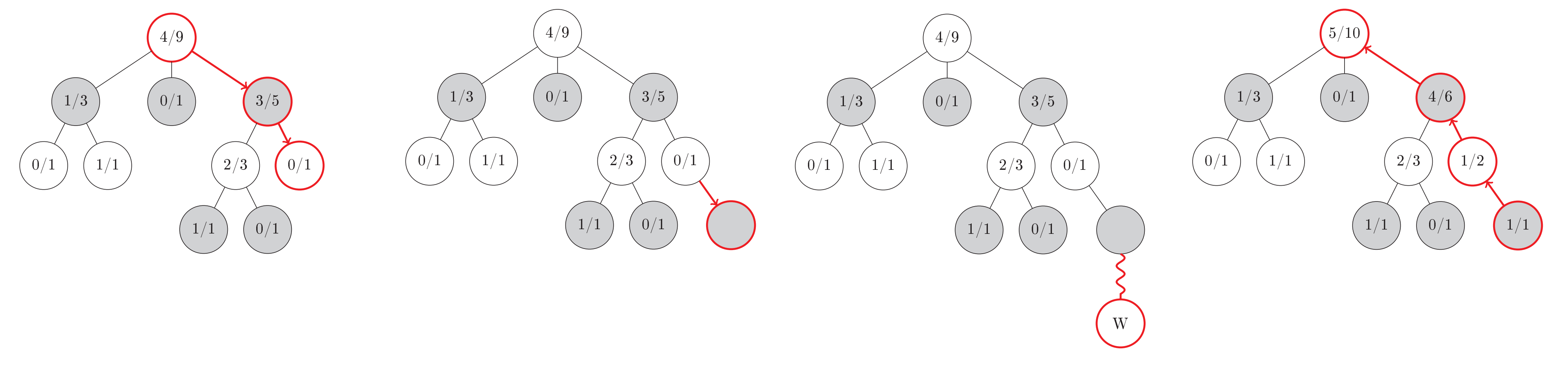
Further work can test multi-core parallelisation on more cores and consider a hybrid of multi-core and cluster parallelisation.

Source Code

Source code available at:
`http://bitbucket.org/francoisvn/oakfoam`

Monte-Carlo Tree Search

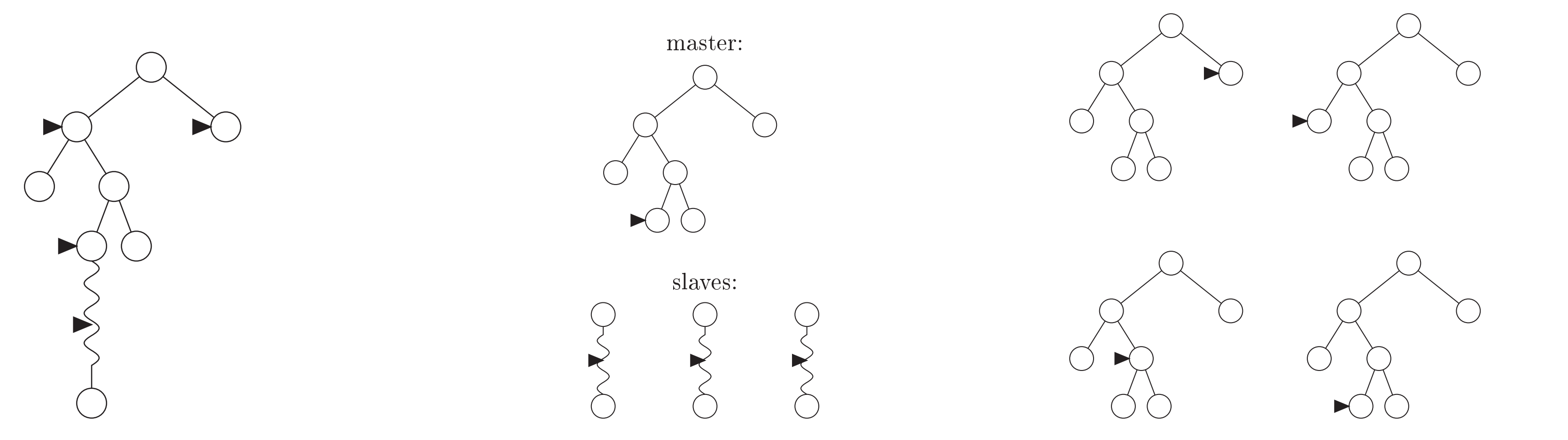
Monte-Carlo (MC) methods evaluate a Go position by doing a number of simulations (playouts) starting from that position. These playouts select a random move for each player and continue until the end of the game. If these playouts are combined with a game tree we arrive at MCTS. One iteration of an MCTS search is shown to the right.



1. Selection 2. Expansion 3. Simulation 4. Backpropagation

Parallelisation

It has been shown that an increase in the number of playouts gives an increase in playing strength. Parallelisation attempts to provide this increase by using parallel hardware to increase the rate of playouts. The three major parallelisation methods for MCTS are shown to the right (execution threads are indicated with ▶).



Tree Parallelisation Leaf Parallelisation Root Parallelisation

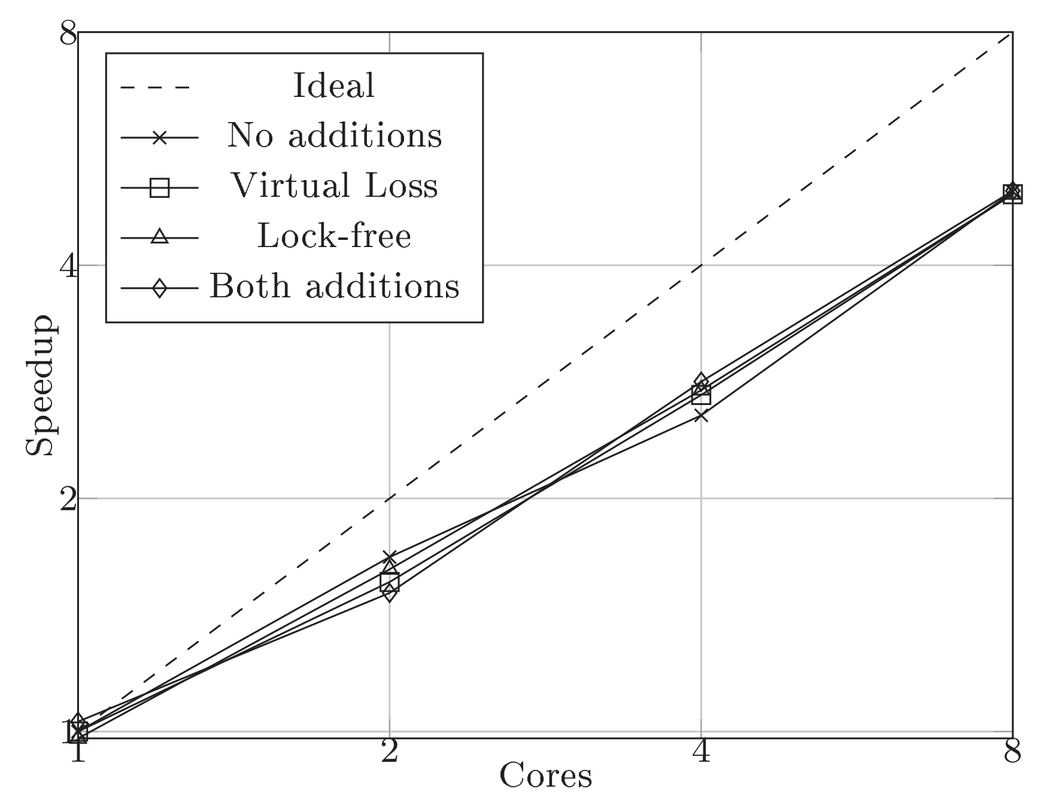
Results

Time/Move	Games	Winrate [%]
2s	100	57
10s	100	68

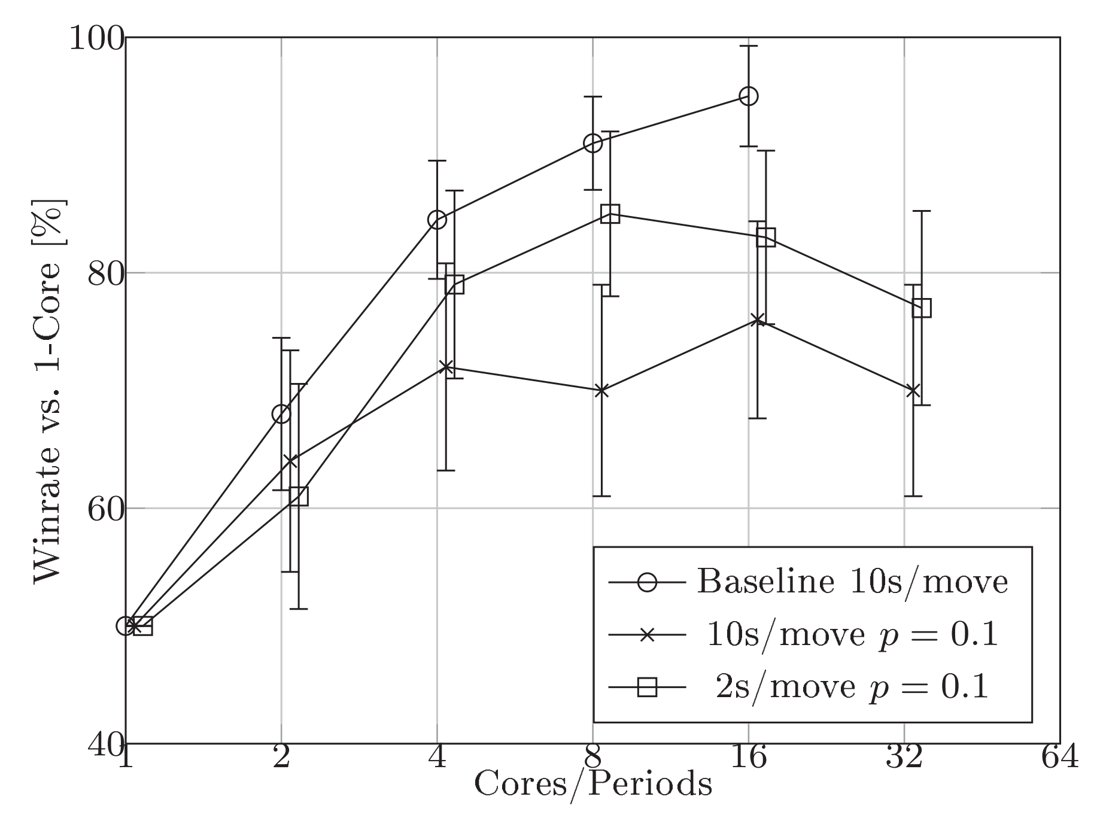
Performance with Pondering on 9x9

Time/Move	Games	Winrate [%]
2s	100	57
10s	100	56

Performance with Pondering on 19x19



Multi-Core Speedup on 19x19



Cluster Strength on 19x19

Pondering was tested by comparing a version of the program with pondering and one without. Results show that pondering performs as expected. Multi-core parallelisation was tested by measuring the speedup in the rate of playouts. Results showed a linear speedup to eight cores. Cluster parallelisation was tested by comparing against a single-core version of the program given more thinking time. Results on 19x19 showed that cluster parallelisation scaled up to eight cores, where a strength increase equivalent to four ideal cores was achieved.