UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

MCTS Parallelisation

Author:
Francois van Niekerk

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

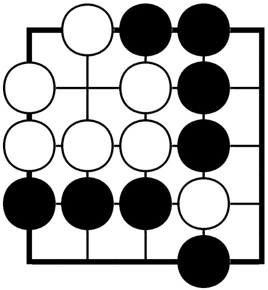
November 2011

Outline

Introduction

- Physical processor constraints have lead to parallel hardware
- Parallelisation of algorithms is increasingly important
- Parallelisation of the Monte-Carlo Tree Search (MCTS) algorithm for Computer Go was the focus of this project

Go and Computer Go



- Go is an ancient board game
- Rules are simple
- Emergent complexity
- Computer Go is the field of software that plays Go
- Dominant algorithm is MCTS
- MCTS can be parallelised

Objectives

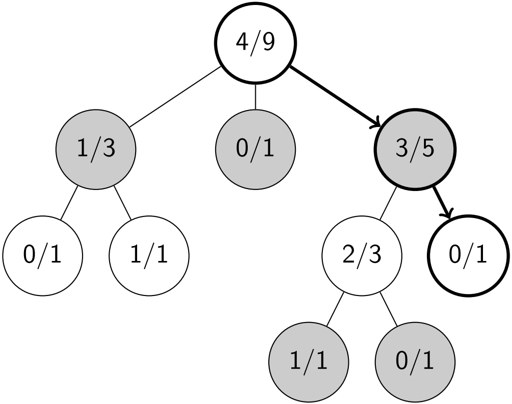
- Pondering (thinking during the opponent's time)
- Multi-core parallelisation
- Cluster parallelisation

Monte-Carlo Tree Search

- Difficult to determine good evaluation function for Go
- Monte-Carlo (MC) methods simulate the outcome (playout)
- MCTS uses a game tree with node values based on MC simulations
- MCTS performs better than alternatives

Monte-Carlo Tree Search

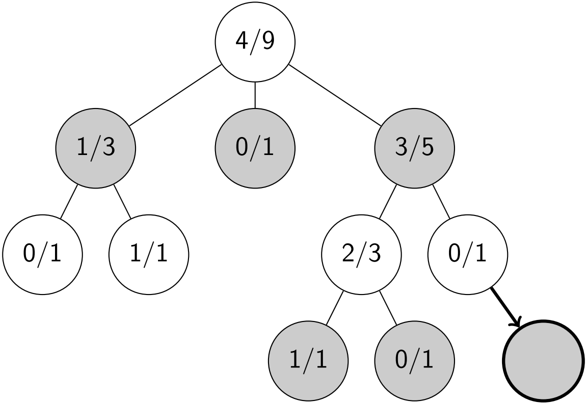
Example



Selection

Monte-Carlo Tree Search

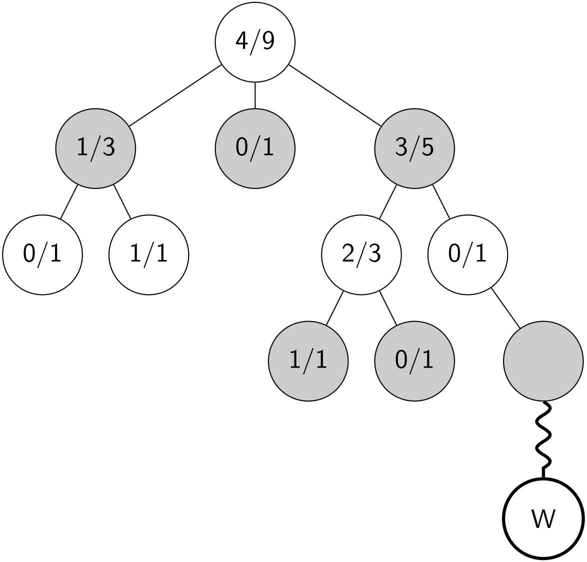
Example



Expansion

Monte-Carlo Tree Search

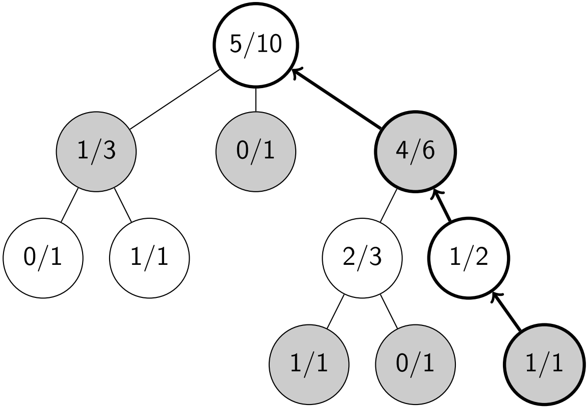
Example



Simulation (playout)

Monte-Carlo Tree Search

Example

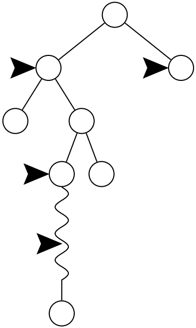


Backpropagation

Parallelisation

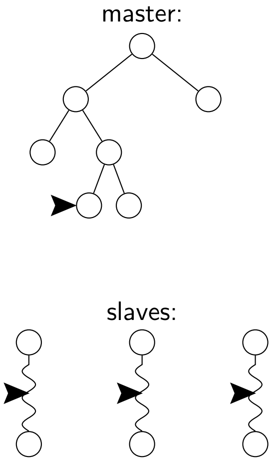
- Increase in number of playouts gives an increase in playing strength
  - Thinking time
  - Rate of playouts
- Parallelisation: use parallel hardware to increase rate of playouts
- Three major parallelisation methods for MCTS:
  - Tree
  - Leaf
  - Root

Tree Parallelisation



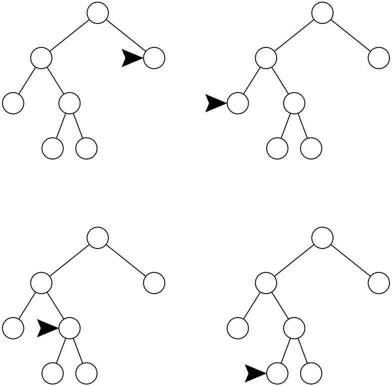
- Shared tree
- Suitable for shared-memory systems only

Leaf Parallelisation



- Master and slave nodes
- Only one tree, on the master
- Slaves are playout workers

Root Parallelisation



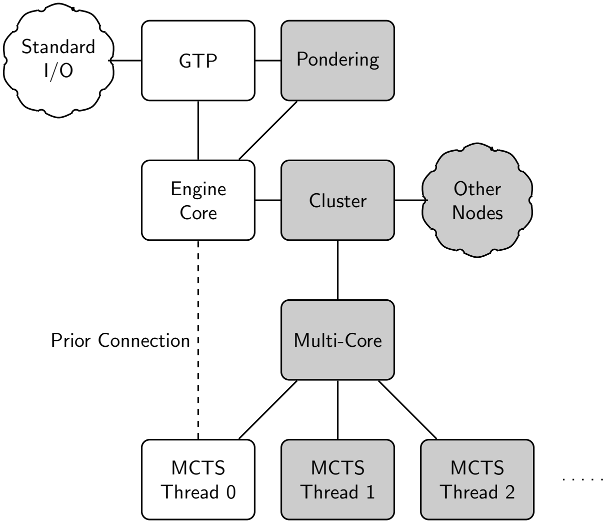
- Each compute node maintains a tree
- Periodic sharing of information

Design and Implementation

- Extend existing MCTS implementation (Oakfoam)
- Tree parallelisation for multi-core systems
  - Boost C++ Threads
- Root parallelisation for cluster systems
  - MPI standard, Open MPI

Design and Implementation

System Diagram



Pondering Results

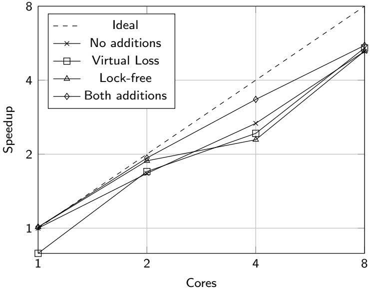
Time/Move	Games	Winrate [%]
2s	100	57 ± 9.70
10s	100	68 ± 9.14

Performance with Pondering on 9x9

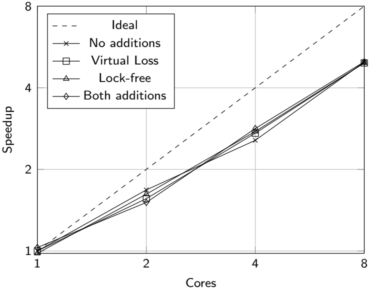
Time/Move	Games	Winrate [%]
2s	100	57 ± 9.70
10s	100	56 ± 9.73

Performance with Pondering on 19x19

Multi-Core Parallelisation Results

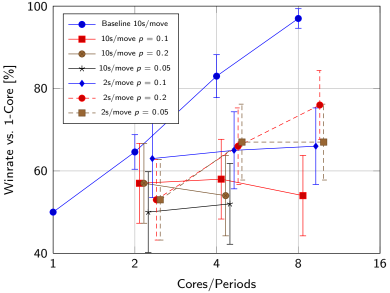


Speedup on 9x9

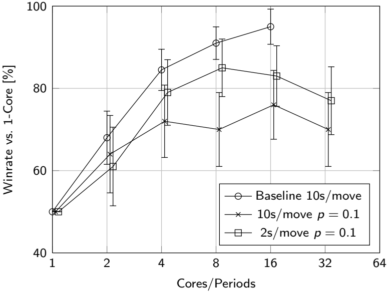


Speedup on 19x19

Cluster Parallelisation Results



Strength Comparison on 9x9



Strength Comparison on 19x19

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 of four ideal cores

Thanks

Thank you to everyone that made this project possible and thank you for listening to this talk.

Oakfoam source code:
`http://bitbucket.org/francoisvn/oakfoam`

Stellenbosch Go Club meetings are Wednesdays from about 19:00 in the Neelsie near Jeff's Place. Beginners are welcome.

This Wednesday there will be an extended talk from 18:00 to 19:00 on Computer Go. Petr Baudiš, author of Pachi, will be speaking.

Any questions?

Prepared with LaTeX and BEAMER