The Parallelization of Monte-Carlo Planning

S.Gelly, J.B.Hoock, Y.Kalemkarian, A.Rimmel and O.Teytaud

ICINCO 2008
Introduction

MultiCore Parallelization

Cluster Parallelization

Conclusion
Introduction

Context :
- Parallelization of Bandit Based Monte-Carlo (BBMC)

Application :
- The Game of Go
the Game of Go

- two players board game
- complete information
- “Task Par Excellence for AI” (Hans Berliner)
BBMC

- Construction of an unbalanced Tree of possible futures.
- Evaluation of each node with random simulations.
- Trade off between exploration and exploitation.

⇒ very different from alpha-beta
⇒ much better at least for the game of Go
⇒ can also be used in planification
Introduction
MultiCore Parallelization
Cluster Parallelization
Conclusion

The Parallelization of Monte-Carlo Planning
The Parallelization of Monte-Carlo Planning
The Parallelization of Monte-Carlo Planning
The Parallelization of Monte-Carlo Planning
Introduction
MultiCore Parallelization
Cluster Parallelization
Conclusion

S.Gelly, J.B.Hoock, Y.Kalemkarian, A.Rimmel and O.Teytaud
The Parallelization of Monte-Carlo Planning
Policies

- Tree Policy:
  We chose the move $i$ with the highest
  \[ \hat{X}_i + p \sqrt{\frac{\log T}{T_i}} \]
  \( \hat{X}_i \): Empirical average reward for move $i$
  \( T_i \): Number of trials for move $i$
  \( T \): Total number of trials

- Default Policy:
  Random
  Some handcrafted simple rules
Trade-off between...

\[ \hat{X}_i + p \sqrt{\frac{\log T}{T_i}} \]
S.Gelly, J.B.Hoock, Y.Kalemkarian, A.Rimmel and O.Teytaud

The Parallelization of Monte-Carlo Planning

Exploitation

\[
\hat{X}_i + p \sqrt{\frac{\log T}{T_i}}
\]
**Exploration**

\[ \hat{X}_i + p \sqrt{\frac{\log T}{T_i}} \]
Introduction

MultiCore Parallelization

Cluster Parallelization

Conclusion
Idea

- Each Thread makes an independent simulation
- Each Thread updates the memory
Principle
Principle
Principle
Principle
Principle
The Parallelization of Monte-Carlo Planning

Principle
Principle
Principle
Principle
Principle
Delay Effect

Not strictly equivalent to the sequential code:

- During a simulation, one thread does not have the updated statistics from threads below it.
- During the update of the memory, there could be some time lost because the threads can’t access to the same part of the memory at the same time (concurrency in memory access).
### Results

<table>
<thead>
<tr>
<th>Nb Threads *</th>
<th>10 sec.procs</th>
<th>20 sec.procs</th>
<th>40 sec.procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Thread</td>
<td>51.1 ± 1.8</td>
<td>62.0 ± 2.2</td>
<td>74.4 ± 2.4</td>
</tr>
<tr>
<td>2 Thread</td>
<td></td>
<td>62.9 ± 1.8</td>
<td></td>
</tr>
<tr>
<td>4 Thread</td>
<td></td>
<td></td>
<td>66.4 ± 2.1</td>
</tr>
</tbody>
</table>
Introduction

MultiCore Parallelization

Cluster Parallelization

Conclusion
1. Each computation node makes an independent simulation
2. Each node updates his tree
3. Each node sends informations to the other nodes
4. Each node receives informations from the other nodes
5. Each node updates his tree with the new informations
Limited Speed up

Hypothesis:
- no communication time
  - 3 and 4 become negligible
- infinite number of computation nodes
  - 1 becomes negligible

Conclusion:
- only the update time remains.
- let $\alpha$ be the proportion of time taken by the update of the tree in the sequential algorithm.
- the maximum speed up is $1/\alpha$. In our program, roughly 20.
Alternate Solution

Each computation node builds in his own tree. Every $T_0$ second:

- average statistics in the tree for all nodes
  - with a depth $\leq K$
  - with at least $N_{min}$ simulations
The Parallelization of Monte-Carlo Planning
The Parallelization of Monte-Carlo Planning

Introduction
MultiCore Parallelization
Cluster Parallelization
Conclusion

N_{min}=4

K=3
Introduction
MultiCore Parallelization
Cluster Parallelization
Conclusion

The Parallelization of Monte-Carlo Planning
## Results

Experiments with 32 computation nodes:

<table>
<thead>
<tr>
<th>$N$ nodes</th>
<th>19*19 success rate</th>
<th>9*9 success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>48.0 ± 9.9</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>63.2 ± 5.5</td>
<td>50.5 ± 2.1</td>
</tr>
<tr>
<td>8</td>
<td>63.1 ± 4.2</td>
<td>53.5 ± 2.1</td>
</tr>
<tr>
<td>4</td>
<td>73.5 ± 3.4</td>
<td>61.2 ± 2.9</td>
</tr>
<tr>
<td>2</td>
<td>82.4 ± 2.7</td>
<td>63.1 ± 2.9</td>
</tr>
<tr>
<td>1</td>
<td>93.1 ± 1.4</td>
<td>72.5 ± 2.6</td>
</tr>
</tbody>
</table>
Conclusion

- Straightforward cluster parallelization doesn't work in practice.
- The simple proposed algorithm achieves much better results.
- The two parallelizations are orthogonal.
- First approved victory of a go program against a professional player on a 9*9 board (one game on three).