NeuGenGo

Kann unser neuronales Netz besser Go spielen als wir?

Lennart Braun, Armin Schaare, Theresa Eimer

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, Arbeitsbereich WR
Praktikum Parallele Programmierung SS 15

9. September 2015

Problemstellung

- Unser Ziel ist es, neuronale Netzwerke zu trainieren, so dass diese uns im Go schlagen können.
- Zwischenziel / Alternative: Können wir neuronale Netze so trainieren, so dass sie besser als zufällig erzeugte Netze spielen?

TODO: Ziele besser verkaufen

Go

- Asiatisches Brettspiel
- Wird auf Brettern mit 19×19 Knoten gespielt.
- Ziel: Gebiet einkreisen und gegnerische Steine schlagen
- Spielende: wenn beide Spieler passen

TODO: Graphik (möglichst unter CC / selbst erstellt)

Abbildung: Stellung eines Go Spiels

Neuronale Netzwerke

- TODO: kurze Beschreibung von neuronalen Netzwerken

TODO: Graphik (möglichst unter CC / selbst erstellt)

Abbildung: Schema eines neuronalen Netzwerks

Lösungsansatz

- Beschränkung auf 9×9 Bretter
- Feedforward Netze (TODO: Layout)
- Genetische Algorithmen (TODO: Parameter)

Lösungsansatz

Algorithmus 1 sequentielle Lösung

1: $N_0 \leftarrow \{n$ zufällig generierte neuronale Netzwerke $\}$
2: **for** $net \in N_0$ **do**
3: trainiere net auf regelgerechtes Spielen
4: **end for**
5: **for** Generation $i = 0$ bis ... **do**
6: **for** $\forall net_a \neq net_b \in N_i$ **do**
7: lass net_a, net_b gegeneinander spielen
8: zähle die Anzahl der Siege
9: **end for**
10: generiere N_{i+1} mittels genetischen Algorithmus abhängig von N_i und den Spielergebnissen
11: **end for**
12: Speichere N_n

UML

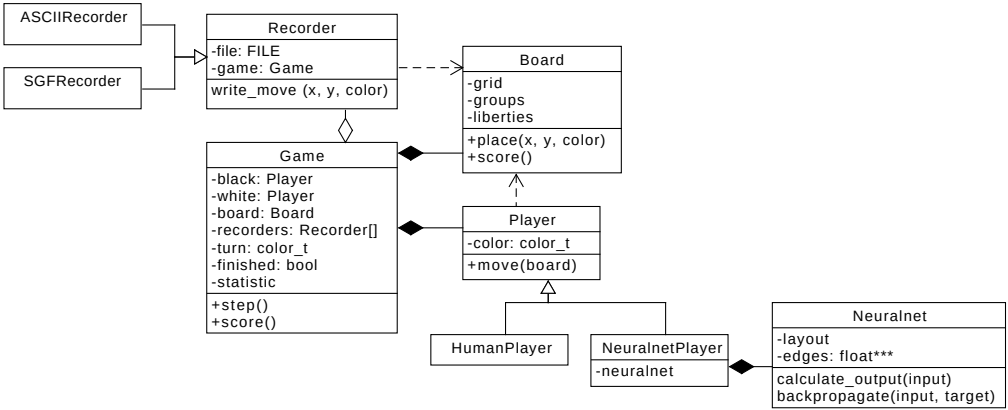

Abbildung: Klassendiagramm

Parallelisierungsschema

Was ist parallelisierbar?

- – Die Generationen sind inherent sequentiell
- \+ die einzelnen Spiele sind unabhängig voneinander (**for** $net_a, net_b \in N_i$ **do**)
- \+ die Ausgabeberechnung in den Neuronalen Netzwerken (dreifache Schleife)

Parallelisierung der Spielphase

message passing

Ziel: n^2 Spiele bei n Netzen auf p Prozesse zu verteilen
notwendige Kommunikation:

- die Netze $n \cdot 1.59\,\mathrm{KiB}$
- Anzahl der Siege $n \cdot 8\,\mathrm{B}$ (uint8_t verwenden?)

Probleme:

- neue Netzwerke in jeder Runde
- unterschiedliche Laufzeit der Spiele
  $\Rightarrow$ Wartezeiten bei Ringstruktur

Parallelisierung der Spielphase

message passing

Algorithmus 2 parallele Spielphase

```
 1: for Generation i = 0 bis ... do
 2:     broadcast(N_i, 0)
 3:     for ∀net_a ∈ N_i pardo
 4:         for ∀net_b ≠ net_a ∈ N_i do
 5:             lass net_a, net_b spielen
 6:             zähle die Siege (wins)
 7:         end for
 8:     end pardo
 9:     reduce(wins)
10:     if rank = 0 then generiere N_{i+1} end if
11: end for
```

- Jeder Prozess erhält alle Netze.
- Gleichmäßige Verteilung der äußeren Schleife (Zeile 3).

Parallelisierung der Spielphase

Spurdatenanalyse

TODO: Visualisierung durch Vampir

Abbildung: Vampir

Parallelisierung der neuronalen Netzwerke

shared memory

TODO:

Strong Scaling

TODO:

Weak Scaling

TODO: