Universiteit van Amsterdam

Technisch Rapport

Doolhoven met Go

25 maart 2018

Tutor:
R. Klusman

Practicumgroep:
C1

Cursus:
Programmeertalen

Student:
Sam van Kampen
11874716

1 Introductie

In dit technisch rapport wordt gekeken naar een manier om een pad van een beginpunt naar de uitgang van een doolhof te vinden op een concurrente manier. Ook zal een implementatie worden besproken in de programmeertaal Go. Aangezien de implementatie concurrent moet zijn, was mijn onderzoeksvraag of er een meetbaar verschil is tussen het uitvoeren van het programma op één of op meerdere processorkernen te draaïen.

2 Probleemstructuur

Iedereen kent het wel van de basisschool: het oplossen van een doolhof. Een doolhof bestaat uit een aantal vakjes met mogelijke muren om die vakjes. Er moet een pad gevonden worden van een beginvakje (op coördinaat `(0, 0)`) naar een uitgang van het doolhof, waarbij niet door muren gelopen mag worden.

Voor een mens kan een doolhof natuurlijk visueel worden gerepresenteerd, maar voor een computer zijn andere representaties simpeler. De representatie die voor deze opdracht werd voorgeschreven is als volgt:

- Een doolhof kan worden gerepresenteerd als tekst...
- Waarbij elke regel een rij in het doolhof is en elke kolom een kolom...
- En elk karakter een getal is van nul tot drie, waarvoor geldt
  - **0** → Geen muren
  - **1** → Een muur aan de zuidkant
  - **2** → Een muur aan de oostkant
  - **3** → Een muur aan beide kanten

Alleen de muren aan de zuid- en oostkant zijn belangrijk, aangezien muren aan de noord- en westkant afgeleid kunnen worden door te kijken of het vakje ten noorden of ten westen een muur aan de zuid- of oostkant hebben, respectievelijk.

3 Methode

De geschreven implementatie bestaat uit drie delen: een deel dat de representatie inleest, een boekhoudfunctie die bijhoudt hoeveel oplossingsroutines er draaien, en een oplossingsroutine. Deze delen worden aan elkaar gelijmd door de hoofdfunctie, die het zo mogelijk opgeloste doolhof print.

De implementatie is concurrent door middel van het concept van **goroutines** - aparte functies die onafhankelijk van elkaar in parallel kunnen draaien. Door deze op verschillende processorkernen te draaien kan men parallellisme creëren.

3.1 Het inlezen van de representatie

De interne representatie van de implementatie is simpelweg een 2D-array, die bestaat uit de regels van het doolhofbestand. Het inlezen is dan ook vrij simpel - elke regel wordt in een array gelezen. Het was ook mogelijk geweest om de representatie in te lezen als een binaire boom, maar deze methode is simpeler te implementeren en is bruikbaar voor ons algoritme.

3.2 De boekhoudfunctie

De boekhoudfunctie houdt bij hoeveel oplossingsroutines er draaien, en zorgt ervoor dat er nieuwe oplossingsroutines gestart worden als er nieuwe mogelijke paden teruggegeven worden door de draaiende routines.

Ook houdt de boekhoudfunctie bij of er al een route gevonden is. Als dit het geval is worden er geen volgende routines meer aangemaakt, en sluit het programma af nadat alle goroutines beëindigd zijn.

3.3 De oplossingsroutine

De oplossingsroutine communiceert met de boekhoudfunctie. De routine geeft aan welke mogelijke volgende paden er doorzocht kunnen worden, wacht op een ontvangstbericht en sluit dan af. Als de routine een uitgang heeft gevonden wordt dit ook gecommuniceeerd, en wordt het pad naar de uitgang teruggestuurd naar het hoofdproces.

Het is theoretisch mogelijk om ervoor te zorgen dat de oplossingsroutine een van de mogelijke volgende paden zelf volgt in plaats van af te sluiten. Dit zou efficiënter zijn, maar ook moeilijker te implementeren, en heb ik dus achterwege gelaten.

4 Resultaten

De volgende resultaten zijn verkregen door `maze` meermaals te draaien onder het programma `multitime`, die automatisch deze statistieken berekent.

Tabel 1: Tijd om `maze1.txt` op te lossen met één of meerdere processorkernen, in seconden. Data is berekent over 1000 runs. De 2/4 refereert naar twee cores met hyperthreading (dus 4 threads)

Kernen	**Mean**	**Std.Dev.**	**Min**	**Median**	**Max**
1	0.015	0.005	0.014	0.015	0.047
2/4	0.026	0.006	0.020	0.028	0.046

5 Discussie

Zichtbaar is dat het programma meer tijd kost om te draaien onder meerdere processorkernen. Dit duidt erop dat het gebruik van meerdere kernen meer *overhead* met zich meebrengt dan het waard is. Misschien zou de looptijd kleiner zijn als er meer werk zou worden uitgevoerd per goroutine. Dit is dan ook een mogelijkheid voor verder onderzoek.

5.1 Conclusie

Er is dus zeker een meetbaar verschil tussen de looptijd bij het gebruik van een of meer kernen. Het verschil is echter negatief, dus het is beter om het programma op een enkele kern te draaien.