Universiteit van Amsterdam

Schrijven tweede technische rapport

Invloed van processor-parallellisatie op een gelijklopende doolhofverkenner

26 augustus 2018

Student:
R.A.J. Wacanno
11741163

Tutor:
W.R.J. Vermeulen

Practicumgroep:
A1 - ALGOL

Cursus:
Programmeertalen

1 Introductie

Voor het vak Programmeertalen was het de bedoeling om in de programmeertaal *Go* een doolhofverkenner te maken. Hierbij moest echter gebruik gemaakt worden van de gelijklopende (*concurrent*) eigenschappen van de taal. Dit gelijklopen betekent dat er, gedurende de uitvoering van een programma, bepaalde onderdelen tegelijkertijd draaien. Hiermee kan in combinatie met processor-parallellisatie, ofwel het door een processor laten uitvoeren van meerdere instructies op hetzelfde moment, een verkorting van de uitvoeringstijd van het programma behaald worden.

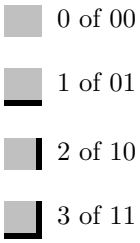
1.1 De opdracht

De te schrijven verkenner moet een gegeven bestand uitlezen welke de representatie van een doolhof bevat. Deze representatie kan er als volgt uit zien:

```
2012011122
1322210322
2013202030
2120223121
```

Figuur 1: Een voorbeeld van de doolhofrepresentatie

In deze representatie staat ieder getal voor een hokje binnen het doolhof en beschrijft het hoe het hokje ommuurd is. Om deze ommuring af te leiden moet gekeken worden naar de binaire vorm van ieder getal en dan in het bijzonder naar de twee meest rechter bits. Bij deze twee rechter bits betekende een waarde van `1` het bestaan van een muur en een waarde van `0` het bestaan van een opening tussen het betreffende hokje en diens buurman. De eerste bit van rechts gezien staat voor de ondermuur van het hokje en de tweede voor de rechtermuur. De getallen 0 tot en met 3 vertegenwoordigen de hokjes dus als volgt:



Figuur 2: Grafische duiding van de cijfers in de doolhofrepresentatie

Te zien is dat het bestaan van de boven- en de linkermuur van de hokjes niet gedefinieerd kan worden. Hun bestaan wordt bepaald door de hokjes boven en links van het betreffende hokje. Op deze manier wordt voorkomen dat muren dubbel gedefinieerd worden. Bijkomstigheid hiervan is echter wel dat de gehele boven- en linkermuur van het doolhof geen openingen kan bevatten.

Het programma dient het ingelezen doolhof te verkennen en een mogelijke uitweg vast te stellen. Het originele doolhof moet vervolgens, met de uitweg daarin verwerkt, conform dezelfde representatie als uitvoer worden gegeven.

1.2 Het onderzoek

Dit onderzoek poogt vast te stellen of er bij dit gelijklopende programma prestatiewinst behaald kan worden met het beschikbaar stellen van meerdere processor-kernen. De onderzoeksvraag luid daarom: “Heeft een verhoogde processor-parallellisatie een positieve invloed op de uitvoeringstijd van de gelijklopende doolhofverkenner?”

2 Methode

Bij het schrijven van het programma en het onderzoek moesten respectievelijk keuzes met betrekking tot implementatie en uitvoering gemaakt worden.

2.1 Algoritme

Het gelijklopen binnen *Go* is gebaseerd op zogenaamde routines. Een routine is in feite een functie die als zelfstandig proces draait en zo, los van het hoofdproces, opdrachten kan uitvoeren. Bij het gebruik van routines is het wel van belang dat deze, zodra zij niet meer nodig zijn, zorgvuldig worden afgesloten.

Verkennen van posities
Voordat beschreven kan worden hoe met de routines wordt omgegaan is eerst begrip van de werking van de voor dit programma geschreven routine nodig. In deze verkennersimplementatie heeft de routine als taak om voor een gegeven positie van een hokje te bepalen welke muren dat betreffende hokje omringen. Voor iedere opening die er rondom het hokje is wordt vervolgens een nieuwe routine gestart die de positie van het hokje die ook aan die opening grenst verkent.

Om de aanwezigheid van een muur te bepalen wordt een *AND*-operatie uitgevoerd op het getal dat het hokje representeert en de constante voor een onder- of rechtermuur. In het geval dat er geen muur is — de uitkomst van de *AND* operatie is dan 0 — wordt er zoals eerder gezegd een nieuwe routine op de gewenste positie gestart.

Als de verkennende routine bij een buitenmuur komt — dit zijn muren van de rechter of onderrand van het doolhof — en hier is een opening, dan wordt het tot dan toe genomen pad

naar de hoofdroutine gestuurd. Dit gebeurt middels een kanaal; eigenlijk een doorgeefluik voor data tussen routines.

Mocht de verkenner alle muren verkent hebben of een opening in een buitenmuur gevonden hebben, dan zal hij een sluitsignaal in een daarvoor bestemd kanaal naar de hoofdroutine sturen alvorens hij stopt.

Bijhouden van routines
Om alle routines de kans te geven te stoppen voordat de uitvoering van het programma beëindigt moet bekend zijn hoeveel routines er op ieder moment draaien. Dit wordt bewerkstelligd door steeds wanneer een routine gestart wordt een teller met 1 op te hogen om deze ook weer met 1 te verlagen zodra een routine stopt. Het starten van routines gebeurt hierom op één centrale plek in de hoofdroutine. Verkenroutines kunnen via een apart kanaal het starten van een nieuwe verkenroutine opdragen door een positie te sturen. Zo kan de routineteller dus bij het ontvangen van een startbericht opgehoogd worden en bij het ontvangen van het eerdergenoemde sluitsignaal verlaagd worden. De hoofdroutine zal in deze context pas stoppen als de routineteller op 0 staat.

2.2 Procedure

Om de invloed van processor-parallellisatie op de prestaties van de verkenner te bepalen is gekozen om de uitvoeringstijd te bepalen van het programma wanneer het hetzelfde doolhof oplost, maar een verschillende hoeveelheid processor-kernen tot zijn beschikking heeft.

Hierbij is gebruikgemaakt van twee hulpprogramma's, namelijk `taskset` en `multitime`[1]. `Taskset` maakt het mogelijk om het aantal processor-kernen dat een programma mag gebruiken te limiteren. `Multitime` had voor dit onderzoek twee toepassingen; het maakt het meermaals uitvoeren van een bepaald commando mogelijk en berekent hier ook de gewenste statistische tijdsgegevens voor uit.

Deze gereedschappen zijn gecombineerd tot het volgende commando:
`taskset -c 0,1 multitime -n 100 -q ./maze maze5.txt`. Met `-c ...` wordt in de vorm van een lijst aangegeven welke processor-kernen gebruikt mogen worden; `0` staat voor kern 1, `0,1` staat voor kern 1 en 2 en `0-3` staat voor kern 1 tot en met 4. Om aan te geven hoe vaak het commando herhaald moet worden, wordt de vlag `-n ...` meegegeven waar in dit geval 100 achter staat. Dit was tevens het aantal herhalingen dat tijdens dit onderzoek is gebruikt.

3 Resultaten

Na het succesvol uitvoeren van het bovengenoemde commando worden door `multitime` de gemiddelde uitvoeringstijd en andere statistisch relevante gegevens gepresenteerd. Deze zijn in tabel 1 uiteengezet.

Aantal kernen	Gemiddelde	Standaard-afwijking	Minimum	Mediaan	Maximum
1	0,064	0,015	0,044	0,063	0,092
2	0,076	0,019	0,048	0,084	0,101
4	0,084	0,020	0,054	0,087	0,118

Tabel 1: *Uitvoeringstijden in seconden per hoeveelheid processor-kernen.* $n = 100$

[1] `https://tratt.net/laurie/src/multitime/`

4 Discussie

De verkregen resultaten lijken de verwachtte snelheidsverbeteringen bij het combineren van gelijklopende processen en processor-parallellisatie tegen te spreken. Zo blijkt namelijk dat, bij het verhogen van het aantal beschikbare processor-kernen, de gemiddelde uitvoeringstijd van het programma toeneemt. Dit valt echter, binnen de context van dit programma, te verklaren.

Allereerst zorgt het voortdurend heen en weer sturen van berichten via kanalen tussen routines ervoor dat zij steeds op elkaar moeten wachten. Hoewel dit de stabiliteit van het programma vergroot komt dit de snelheid niet ten goede.

De andere reden is dat het parallel draaien van veel gelijklopende routines vaste tijdskosten per routine met zich meebrengt. Idealiter zijn deze kosten ten opzichte van de tijd die de routine bezig is zo klein, dat zij geen significante bijdrage leveren aan de totale uitvoeringstijd van het programma. Bij deze doolhofverkenner voeren de routines enkel een kleine taak uit alvorens zij klaar zien. Deze extra tijdskosten voor veelvoudige parallellisatie zijn op zijn beurt dus in mindere mate en nauwelijks aanwezig bij het draaien op respectievelijk 2 en 1 kernen in vergelijking met het draaien op 4 kernen.

4.1 Conclusie

Om de geparallelliseerde prestaties van deze doolhofverkenner te verbeteren zou de afhankelijkheid van routines op andere moeten worden geminimaliseerd. Zij zouden dus minder vaak op elkaars berichten via kanalen hoeven te wachten.

Een veel grotere winst valt te behalen wanneer iedere individuele routine meer werk zou verrichten, om zo de significantie van de tijdskosten van parallellisatie teniet te doen.