Programmering och hastighet

Johan Niklasson
jnikl@kth.se
Grupp B:6

2017-11-20

Sammanfattning

Kan programspråk som lanserades för 45 år sedan vara snabbare än programspråk som lanseras idag? För att undersöka detta gjordes experiment med sex olika kompilerade programspråk med ett spann på 37 år mellan det äldsta och yngsta. Mätningarna gjordes med algoritmen Collatz problem, som programmerades i de olika språken och sedan kördes på en Raspberry Pi för att uppmäta hur lång tid varje programspråk tog på sig att lösa problemet. Studien visade att programspråken faktiskt har blivit snabbare med tiden, trots att vissa teorier motsätter sig resultatet.

Innehåll

1 Inledning **2**
- 1.1 Syfte . . . 2
- 1.2 Frågeställning . . . 2
- 1.3 Avgränsning . . . 2

2 Begrepp **3**
- 2.1 Programkod . . . 3
- 2.2 Processor . . . 3
- 2.3 Kompilering . . . 3
- 2.4 Nivåer på programspråk . . . 4
- 2.5 Collatz problem . . . 4

3 Metod **4**

4 Resultat **5**

5 Diskussion **7**

6 Slutsats **7**

7 Referenser **7**

1 Inledning

Ett programspråk är en formell och strikt uppsättning av instruktioner som en dator kan läsa för att utföra uppgifter. Sedan det första programspråket utvecklades i elektroniska komponenter år 1945 (Bauer och Wössner, 1972) har det lanserats flera hundra nya programspråk. Vissa programspråk utvecklas för att lösa specifika problem - programspråket Erlang av Ericsson som exempel skapades för styrsystemen i telefonväxlar (Armstrong, 1997). Andra språk, såsom Scratch, har tagits fram i undervisningssyfte för barn (Maloney m. fl., 2010).

Men även då vissa programspråk har haft vissa typer av uppgifter i beaktelse vid sin framtagning, har man även kunnat få programspråken att arbeta snabbare? Moores Lag som beskriver att antalet transistorer på chip växer potentiellt, resulterar i datorer och processorer blir snabbare varje år (Schaller, 1997). Kan det därför finnas ett mindre behov till att nyare programspråk inte behöver vara lika snabba utan kanske istället till exempel mer lättlästa för människor?

1.1 Syfte

Syftet med studien är att ta fram ett underlag om varför nya programspråk utges. Då det kan vara flera egenskaper som påverkar framtagningen och användningen av programspråk har ett fokus koncentrerats på en av dessa - beräkningshastigheten hos ett programspråk.

1.2 Frågeställning

Frågeställningen för studien är: “Finns det någon korrelation i beräkningshastighet och utgivningsår för kompilerade mellannivåspråk?”

1.3 Avgränsning

I studien är det endast intressant att jämföra språk med liknande egenskaper. Därför kommer endast kompilerbara mellannivåspråk att testas som går att exekveras (köras) och kompileras på en Raspberry Pi (en liten enkortsdator i storleken av ett kreditkort) i operativsystemet Raspbian.

2 Begrepp

I studien används ett antal termer som är relevanta för experimentet. I detta avsnitt kommer dessa förklaras förenklade för att underlätta vad de innebär.

2.1 Programkod

En programkod är en uppsättning instruktioner som utförs en dators processor. Instruktionerna består av jämförelser eller matematiska operationer av två värden som sedan utför efterföljande instruktioner beroende på resultatet.

2.2 Processor

Processorn brukar hänvisas som "hjärnan" i en dator som utför de flesta av instruktionerna från ett programspråk. Den här studien kommer enbart fokusera på tiden för de beräkningar som processorn utför. Instruktionerna som skickas till processorn kallas för maskinkod.

2.3 Kompilering

Att kompilera ett språk innebär att man skriver om beskrivande syntaxer till maskinkod som processorn kan läsa. I programmeringsspråk brukar en så kallad IF-sats, som kontrollerar villkor, skrivas som följande:

```
IF condition THEN
    sequence 1
ELSE
    sequence 2
ENDIF
```

Detta är inte programkod som processorn kan tolka utan behöver först skrivas om till instruktioner processorn kan läsa. Denna omskrivning kallas för kompilering. Ett program som kallas för kompilator gör just detta och skriver om den programkoden till det förväntade formatet, så kallad maskinkod (Srikant och Shankar, 2008).

2.4 Nivåer på programspråk

Inom programmering är det skillnad på hur programspråk fungerar på enheten de körs. I studien kommer endast mellannivåspråk att användas.

Mellannivåspråk: I ett mellannivåspråk kompileras programkoden innan den exekveras. Det kan liknas med att samtliga läses en gång och utförs sedan direkt efter varandra utan att behöva läsa om dem.

Högnivåspråk: Ett högnivåspråk kompilerar programkoden samtidigt som instruktionerna utförs. Det innebär att varje instruktion läses precis innan den utförs och resulterar därför i att programkoden exekveras långsammare än i ett mellannivåspråk (MacLachlan, 1992).

2.5 Collatz problem

Collatz problem är ett olöst matematiskt problem inom talteorin. Problemet går ut på att utföra ett antal matematiska instruktioner eller i det här fallet kallat regler.

1. Ta ett positivt heltal n som är större än ett.
2. Om talet är jämnt, dividera det med två. Om talet är udda, multiplicera det med tre och addera ett.
3. Repetera steg två tills talet når ett.

I studien kommer detta problem skrivas om till en algoritm som utför stegen ovan för enskilda tal. Algoritmen används sedan för att utföra instruktioner som då går att mäta mot varandra i olika programspråk. Algoritmen använder endast addition, multiplikation, division och modulo (för att se om ett tal är jämnt eller udda) som matematiska operationer.

Algoritmen är oförutsägbar i den mån att det inte finns något mönster för hur många steg den den kräver innan den når ett för olika tal. Detta har dock ingen inverkan i jämförelser mellan olika programspråk så länge samma tal jämförs.

3 Metod

Ett initialt urval på programspråk gjordes utifrån de mest använda programspråken idag (Tiobe.com, 2017). Detta gjordes för det skulle finnas en

relevans till användning av de språk som ingick i studien. Av dessa 25 var det cirka hälften som var mellannivåspråk och ytterligare en halvering skedde efter att ha filtrerat bort de språk som inte var kompatibla på en Raspberry Pi. Språken med sina respektive utgivningsår som återstod blev då följande:

- C (1972)
- Ada (1980)
- C++ (1983)
- Java (1995)
- D (2001)
- Go (2009)

För att jämföra olika språk med varandra behövdes en typ av problem lösas. Problemet som användes i studien var Collatz problem som utförde olika matematiska operationer sekventiellt. Totalt användes sex språk, kategoriserade utifrån sitt utgivningsår.

Collatz-algoritmen programmerades i de olika språken och kördes sedan från en funktion som testar algoritmen för alla tal mellan 1 till 100 000 och gör detta i sin tur 100 gånger. Ju längre tid algoritmen tar på sig, desto noggrannare mätvärden gick att jämföra med språken sinsemellan. För att få fram ett konsekvent mätvärde i tid användes en inbyggd timer i en Raspberry Pi, vilket endast hade som syfte att utföra tidsmätningar.

4 Resultat

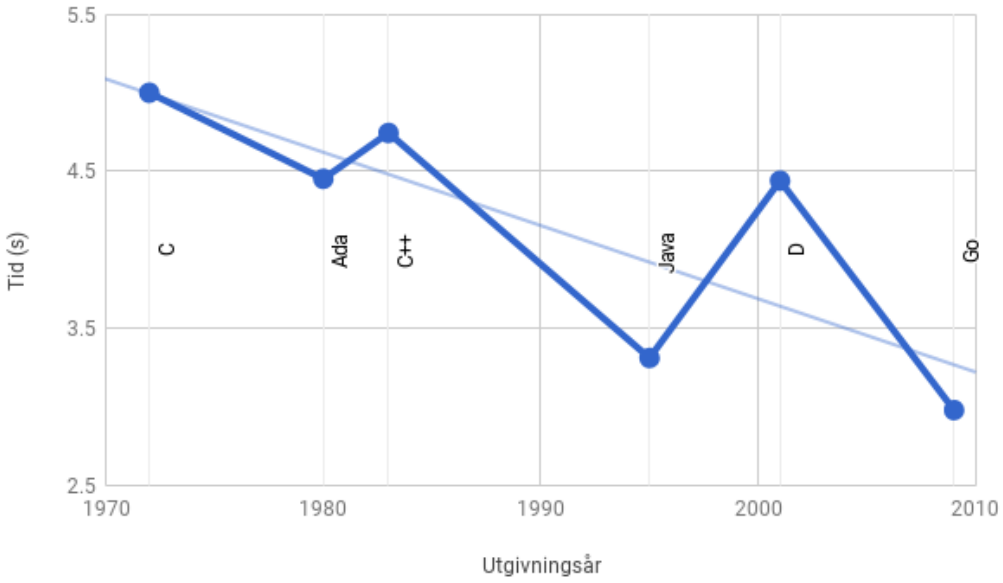
Nedan presenteras de resultat som experimentet genererade. Alla tidsmätningar som presenteras anges i sekunder med tre decimaler.

Tabell 1 presenterar den rådata som hämtades från de olika körningarna i experimentet. Språkets popularitet är även angivet i tabellen. Populariteten hämtades från "TIOBE Programming Community index" där urvalet av programspråken togs från. Populariteten mäts i antal sökträffar språket genererar i sökmotorer (Tiobe.com, 2017).

Direkt till rådatan i tabell 1 kan man utläsa att det äldsta språket C i studien var långsammast medan det nyaste språket Go var snabbast. Ändå är C det näst mest använda språket på cirka 7.7% medan Go förekommer på plats 17 med en popularitet på ungefär 2%.

Tabell 1: Resultatdata

Utgivningsår	Programspråk	Exekveringstid	Popularitet (placering)
1972	C	5.000 s	7.74% (2)
1980	Ada	4.452 s	0.78% (25)
1983	C++	4.744 s	5.18% (3)
1995	Java	3.312 s	16.38% (1)
2001	D	4.440 s	1.23% (22)
2009	Go	2.980 s	1.98% (17)



Figur 1: Exekveringstid över utgivningsår

5 Diskussion

6 Slutsats

7 Referenser

Referenser

Armstrong, Joe (1997). "The development of Erlang". I: *ACM SIGPLAN Notices* 32.8, s. 196–203. DOI: 10.1145/258949.258967.

Bauer, F. L. och H. Wössner (1972). "The Plankalkülöf Konrad Zuse: a forerunner of today's programming languages". I: *Communications of the ACM* 15.7, s. 678–685. DOI: 10.1145/361454.361515.

MacLachlan, Robert A. (1992). "The Python compiler for CMU Common Lisp". I: *ACM SIGPLAN Lisp Pointers* V.1, s. 235–246. DOI: 10.1145/141478.141558.

Maloney, John m. fl. (2010). "The Scratch Programming Language and Environment". I: *ACM Transactions on Computing Education* 10.4, s. 1–15. DOI: 10.1145/1868358.1868363.

Schaller, R.R. (1997). "Moore's law: past, present and future". I: *IEEE Spectrum* 34.6, s. 52–59. DOI: 10.1109/6.591665.

Srikant, Y. N och P Shankar (2008). *The compiler design handbook*. 2. utg. CRC Press, s. 10–15.