A3: Programopførsel

Computer Systems 2017
Department of Computer Science
University of Copenhagen

Finn Schiermer Andersen

Due: Søndag, 29. Oktober, 23:59
Version 1 (11. oktober)

Dette er den fjerde afleveringsopgave i Computersystemer og den anden (og sidste) som dækker Maskinarkitektur. Som tidligere, opfordrer vi til parprogrammering og anbefaler derfor at lave grupper at 2 til 3 studerende. Grupper må ikke være større end 3 studerende og vi advarer mod at arbejde alene.

Afleveringer vil blive bedømt med op til 3 point. Du skal opnå mindst halvdelen af de mulige point over kurset, samt mindst 2 point i hvert emne, for at blive indstillet til eksamen. Denne aflevering hører under Maskinarkitektur. Du kan finde yderligere detaljer under siden "Course description" på Absalon. Det er *ikke* muligt at genaflevere afleveringer.

Introduction

Constant time factors do matter
— Neil D. Jones, *in ACM Symposium on Theory of Computing* (1993)

Som dataloger er vi vant til at kigge på asymptotisk tidskompleksitet. Vi ved f.eks. at matrice-multiplikation er $O(n^3)$ og at en række sorteringsalgoritmer er $O(n \log(n))$. Konstant faktorer er noget vi normalt ser bort fra.

A3 handler om at bestemme disse konstant faktorer for nogle kendte programmer på 3 forskellige klasser af maskiner og om at udvikle nogle programmer med samme asymptotiske kompleksitet men lavere konstant faktorer, dvs. hurtigere.

De kendte programmer vi skal undersøge er

- mergesort,
- to varianter af Heapsort, samt
- en matrice-multiplikation.

Disse programmer findes i mappen `compSys-e2017-sim/examples/` og er navngivet `mergesort.x64`, `heapsort.x64`, `heapsort-2.x64` og `matrixmult-big.x64`. Programmerne benytter simulatorens indebyggede facilitet til generering af tilfældige datasæt i bestemte dimensioner. Brugen af denne er beskrevet i `README.md` [1].

[1] `https://github.com/kirkedal/compSys-e2017-sim/blob/master/README.md`

A3 falder i to dele.

- I den første del skal I måle på de udleverede programmer. Programmerne skal køres på tre forskellige maskiner og for forskellige størrelser af inddata. Målingerne skal gerne (a) bekræfte vores forståelse af den asymptotiske kompleksistet og (b) fastlægge konstant-faktoren.
- I den anden del skal I skrive to programmer og sammenligne deres køretider med de udleverede programmer. De to programmer skal kunne afvikles hurtigere end de udleverede (omend de har samme asymptotiske kompleksitet) på tre forskellige maskiner.

Der udleveres en performancesimulator som kan modellere de tre forskellige maskiner. De tre maskiner er

- En "realistisk" skalar pipeline
- En 4-vejs superskalar pipeline
- En 4-vejs superskalar out-of-order pipeline

Alle udleverede programmer og alle programmer der skal skrives skal bruge instruktioner fra den delmænge af x86 som vi brugte i A2, dog med tilføjelse af to ekstra instruktioner:

- `multq %rA,%rB` - heltals multiplikation
- `shrq %rA,%rB` - skift mod højre (division med potens af 2)

1 Konfiguration af de tre maskiner

Den udleverede performancesimulator kan indstilles til at modellere de tre maskiner på følgende måde:

- En simpel, realistisk pipeline. Her bruges slet og ret performancesimulatorens default indstilling, som svarer til en skalar 9-trins pipeline.
- En 4-vejs superskalar pipeline. Her bruges option `-pw=4`.
- En 4-vejs superskalar med out-of-order execution. Her bruges options `-pw=4 -ooo`.

For en generel beskrivelse af hvordan simulatoren bruges henvises til filen `sim-guide.md` i mappen `architecture-tools` [2]. **I rådes til at læse sim-guide.md før og under brug af simulatoren**.

[2] `https://github.com/kirkedal/compSys-e2017-sim/blob/master/architecture-tools/sim-guide.md`

2 Måling af matrice-multiplikation

Der udleveres et program der kan udføre matrice-multiplikation for $n \times n$ matricer af forskellig størrelse. Dette program skal måles for følgende værdier af n: 10, 15, 20, 25, 30, 35, 40, 45, 50.

For hver af de tre maskiner skal man opsamle antallet af maskin-cykler brugt til udførelsen.

De opsamlede data skal præsenteres i en graf, og det skal argumenteres for om den udleverede kode er $O(n^3)$ som forventet. Endvidere skal der bestemmes tre konstant faktorer, en for hver af de tre maskiner. Den konstante faktor **K** findes ved følgende ligning:

$$\text{antal cykler} = K \cdot n^3$$

3 Måling af sorteringsprogrammer

Der udleveres tre programmer som alle sorterer en tabel af heltal. En merge sort og to heap-sort. De tre programmer skal måles for følgende størrelser af tabellen (som skal være fyldt med tilfældige heltal – se i de udleverede programmer hvordan man genererer tilfælgdige tal): 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 og 10000.

For hver af de tre maskiner skal man opsamle antallet af maskin-cykler brugt til udførelsen.

De opsamlede data skal præsenteres i en graf, og det skal argumenteres for hvorvidt den udleverede kode som forventet er $O(n \log(n))$. Endvidere skal der bestemmes tre konstant faktorer, en for hver af de tre maskiner. Den konstante faktor **K** findes ved følgende ligning:

$$\text{antal cykler} = K \cdot n \log_2(n)$$

Såfremt hastigheden ikke er $O(n \log(n))$ skal der gives en forklaring herpå.

4 Udvikling af en hurtigere matrice-multiplikation

I skal forsøge at skrive et $n \times n$ matrice-multiplikations program der kan afvikles hurtigere end det udleverede. Det er endvidere et krav at selve funktionen der udfører matrice-multiplikationen skal tage 4 argumenter i registre:

- størrelsen n af matricen i `rdi`
- adresse på input matrice A i `rsi`
- adresse på input matrice B i `rdx`
- adresse på output matrice i `rcx`

Dette skal forstås som at overnævnte registre skal sættes til de relevante værdier af den kaldende funktion *inden* den foretager kaldet til matrixmultiplikationsfunktionen. Vi anvender her den samme *kaldkonvention* som bruges af fx. Linux på en x86_64 arkitektur. Den udleverede matrice-multiplikation overholder *ikke* dette krav.

Jeres kode skal endvidere være velkommenteret således at den er nem at forstå både for jer selv og andre. Se de udleverede programmer for inspiration til hvordan dette kan gøres.

Det resulterende programs køretid skal analyseres for de samme matricestørrelser og maskinkonfigurationer som tidligere og målepunkterne skal præsenteres grafisk så de kan ses i forhold til de tilsvarende målepunkter for det udleverede program.

Endvidere skal der bestemmes tre konstantfaktorer, en for hver af de tre maskiner.

Giv en forklaring på hvorfor det udviklede program har vist sig at være hurtigere.

5 Udvikling af en hurtigere sortering

I skal forsøge at skrive et sorteringsprogram som kan sortere de samme tilfældige tal som de udleverede programmer, men hurtigere.

I kan tage udgangspunkt i en anden algoritme med samme typiske tidskompleksitet (vi anbefaler i så fald quicksort), eller I kan tage et af de udleverede programmer og optimere det. Det er nok vanskeligere.

Det resulterende programs køretid skal analyseres for de samme størrelser af datatabel og maskinkonfigurationer som tidligere og målepunkterne skal præsenteres grafisk så de kan ses i forhold til de tilsvarende målepunkter for de udleverede sorteringsprogrammer.

Endvidere skal der bestemmes tre konstantfaktorer, en for hver af de tre maskiner.

Aflevering

Afleveringen skal indeholde følgende filer

- En rapport, `report.pdf`, i PDF-format som indeholder plots og argumentation for hver delopgave
- En zip-fil, `src.zip`, med en src-mappe som indeholder de to assemblerprogrammer som I har udviklet i forbindelse med delopgave 4 og 5. Navngiv filerne hhv. `newmatmult.x64` og `newsort.x64`.
- En tekstfil, `group.txt`, som indeholder ku-brugernavnerne på gruppemedlemmer (et brugernavn pr. linje). Hver linje i filen skal altså matches af det regulære udtryk `^[a-z]{3}[0-9]{3}$`

Afleveringen bør ikke indeholde andet end overstående. Simulator-kildekode og lign. skal altså ikke inkluderes.