A3: Karakterisering og optimering af køretid for sorteringsalgoritmer

Computer Systems 2019
Department of Computer Science
University of Copenhagen

Finn Schiermer Andersen

Due: Søndag, 27. Oktober, 10:00
Version 0.1 (skitse)

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 4 point. Du skal opnå mindst halvdelen af de mulige point over kurset, samt mindst 3 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 en række sorteringsalgoritmer er $O(n \log(n))$. Konstant faktorer er noget vi normalt ser bort fra.

A3 handler primært om at karakterisere opførsel af nogle programmer på tre forskellige klasser af maskiner samt at forbedre køretiden på to af programmerne ved at bruge forskellige teknikker.

Derudover skal I lave en skrivebords-simulation af et lille program på en simpel pipeline.

1 Karakterisering og optimering af køretid

I denne del skal I først sammenligne opførselen af nogle programmer på 3 forskellige klasser af maskiner. Dernæst skal I optimere to a programmerne.

De programmer vi skal sammenligne er:

- Heapsort,
- Quicksort, og

- Mergesort.

Mere detaljeret bliver opgaverne at:

- I skal karakterisere de 3 programmer: Programmerne skal køres på tre forskellige maskiner, to forskellige lager-hierakier og for forskellige størrelser af inddata. Det beskrives i detaljer nedenfor.
- I skal selv gennenføre to optimeringer. En lokalitets-forbedrende omskrivning af den udleverede Mergesort og en maskinnær (assembler niveau) optimering af Quicksort. Derpå skal I sammenligne jeres optimerede udgaver med de udleverede.

1.1 Udleveret materiale

Vi udleverer C-kildetekst til en implementation af Heapsort, Mergesort og Quicksort med en rutine der på køretid indlæser hvorvidt der skal printes og hvor mange elementer man vil have sorteret. Programmet indlæser (mindst) to heltal fra tastaturet. Tallene fastlægger:

- valg af test-mode versus performance-mode
- størrelse af datasæt. I performance-mode genereres en tabel af pseudotilfældige tal som så sorteres
- I test-mode indlæses i stedet en række tal som sorteres

Test-mode kan bruges til at verificere at ens sortering virker korrekt. Test-mode skal være slået fra, når der laves målinger.

Alle programmer findes i vedlagte fil `src.zip`. Denne indeholder også en `README` med nærmere beskrivelser af indholdet, læs denne. Den indeholder også hjælpe script til performance og funktionel tests.

Simulering og modellering af maskiner gøres med "prerf" som kan hentes fra GitHub repositoriet:

`https://github.com/finnschiermer/x86prime`.

Som tidligere kan den virtuelle maskine som er stillet til rådighed benyttes, hvor både x86prime og gcc er forud installeret; kør `update_x86prime.sh` for at få seneste version. Da enkelte maskiner har haft problemer med VirtualBox, vil vi igen stille en remote server til rådighed som kan køre gcc og x86prime. Dette annonceres separat.

1.2 Sammenligning af sorteringsprogrammer

I kan karakterisere et programs opførsel gøres ved hjælp af en ny version af "prerf". Dette program fungerer ligesom "prun", men kan tillige modellere forsinkelser i afviklingen af et program som kan skyldes:

- forsinkelser i lager-hierarkiet (f.eks. cache miss),
- forsinkelser på grund af hop, kald, retur (branch mispredicts, return mispredicts), og

- forsinkelser på grund af dataafhængigheder mellem instruktioner.

For hver af de tre udleverede sorteringsprogrammer skal I bestemme:

- forsinkelser fra cache-miss i lagerhierakiet, og
- køretid for 3 forskellige mikroarkitekturer af varierende kompleksitet.

Til karakterisering skal bruges følgende 3 forskellige maskiner:

- en 9-trins standard pipeline som præsenteret til forelæsningen 7. oktober,
- en 3-vejs superscalar, og
- en 3-vejs out-of-order som præsenteret til forelæsningen 9. oktober.

Dertil skal to forskellige lagerhierakier undersøges:

- et "magisk" lagerhieraki hvor al adgang til lageret tager samme tid (3 cykler) og
- et "realistisk" lagerhieraki med to niveauer af cache.

Til alle målingerne beskrevet nedenfor skal bruges inddatasæt med valgte størrelser. Hvor store data størrelser vil vi lade være op til jer. I skal sikre jer at de valgte størrelser kommer til at vise ønskede effekter i cachen samt at antallet er målepunkter er stort nok til at jeres plots beskriver opførslen.

Vær opmærksom på at kørsler med de største inddatasæt kan tage længere tid. Når I udføre målingerne bør I derfor automatisere jeres kørsler, så hver af kørslerne ikke udføres enkeltvis; brug evt. vedlagte script

1.2.1 Forsinkelser i lagerhierarkiet

Formålet med denne del er at vurdere hvor meget lagerhierarkiet betyder for de 3 programmer. Programmerne skal simuleres på den simple 9-trins pipeline, men med et mere realistisk lagerhierarki:

- 16Kb L1 cache, 3 cyklers adgang
- 128Kb L2 cache, 15 cyklers adgang
- Hovedlager, 115 cyklers adgang

Dette vælges med option "`-pipe simple -mem real`"

Cache miss i hhv. primær cache og sekundær cache for de 4 programmer skal afbildes i en graf. Derpå skal AMAT, "average memory access time", beregnes og præsenteres.

Overvej (og rapporter) hvorvidt det er muligt at sige noget om hvilke af programmerne som har bedst rummelig lokalitet hhv temporal lokalitet og dermed bedst kan udnytte caching?

1.2.2 Realistisk ydeevne

Formålet med sidste del er her ikke så meget at vurdere programmerne, som at illustrere hvordan forskellige mikroarkitekturer kan "skjule" effekten af lange latens-tider, f.eks. cache-miss, ved at finde andre instruktioner at udføre i stedet for at stalle.

Programmerne skal simuleres på alle tre mikroarkitekturer:

- simpel 9-trins pipeline "`-pipe simple -mem real`",
- 3-vejs superscalar "`-pipe super -mem real`" og
- 3-vejs out-of-order "`-pipe ooo -mem real`".

Og køretiderne skal præsenteres i en graf.

Overvej (og rapporter) hvad køretiderne siger om de tre forskellige mikroarkitekturer.

1.3 Udvikling af hurtigere sortering

I sidste del af denne opgave skal I lave jeres egne optimerede udgaver af Quicksort og Mergesort. Jeres udgaver skal sortere de samme tilfældige tal som de udleverede programmer, men bare hurtigere.

I skal bruge to forskellige fremgangsmåder til at nå målet.

1.3.1 Omskrivning med vægt på reference-lokaliet

I skal omskrive Mergesort således at der opnås bedre reference lokalitet. I skal analysere jer frem til om I kan forbedre spatial reference lokalitet eller temporal reference lokalitet eller begge dele. Afhængig af jeres analyse skal I argumentere for hvordan Mergesort kan forbedres. Til slut skal I omskrive Mergesort og eftervise forbedringen. Omskrivningen skal foregå i C. Der må ikke optimeres i den genererede x86prime assembler kode.

1.3.2 Omskrivning med vægt på maskinnære optimeringer

I skal omskrive den udleverede Quicksort på x86prime assembler niveau. Først skal I analysere jer frem til hvilke dele af assembler koden, der med fordel kan optimeres. Optimeringen skal fokuseres på at få mest mulig effekt med mindst mulig indsats (optimering på assembler niveau kan være meget tidskrævende). Optimeringen skal være rettet mod den superskalare maskine. Derpå skal I gennemføre optimeringen og eftervise forbedringen.

1.3.3 Præsentation af resultater

Jeres udviklede algoritmer skal performance testes på samme måde som de udleverede programmer og i skal i rapporten sammenligne resultaterne med disse. Husk også at lave en funktionel test af jeres algoritmer.

Til hjælp med at lave en funktionel test udleverer vi et script der kører simulatoren og tjekker at uddata fra kørslen matcher en reference kørsel med quicksort algoritmen.

2 Teoretisk opgave

TBD.

3 Bedømmelse og rapportering

De forskellige dele af afleveringen bliver vurderet omtrent efter følgende:

- 10% performance testing af eksisterende algoritmer,
- 30% udvikling og test af optimerede algoritmer,
- 20% løsning af teoretiske opgaver,
- 40% rapport som dokumenterer jeres implementation.

Rapporten skal berøre problemstillinger som er rejst under opgaven, inklusiv de overvejelser som opgaveteksten lægger op til. Beskriv alle ikke-trivielle dele af jeres løsning, samt evt. tvetydige formuleringer, som I kan have fundet i opgaveteksten. Det er forventeligt at afrapportering til karakterisering og optimering af køretid er omkring 5 sider (eksklusiv grafer) og den må ikke overskrive 7 sider. Dertil kommer de teoretiske opgaver.

For at få point skal man kunne dokumentere at opgaven er løst korrekt. Det gøres ved at udarbejde og køre testprogrammer og/eller scripts, som kan bekræfte at I kan udføre alle simulationer.

Sammen med jeres rapport, `report.pdf`, skal I aflevere en `src.zip` som indeholder alle relevante udviklede programmer, scripts og test programmer. Denne skal også indeholde de data som er genereret fra kørslerne og afbildet i graferne i rapporten. Følg den struktur som er i den udleverede zip-fil.

Dertil skal der afleveres `group.txt` som indeholder en ASCII/UTF8 formateret liste KU-id'er fra alle medlemmer i gruppen; et id pr. line, ved brug af følgende tegnsæt:

$$\{\texttt{0x0A}\} \cup \{\texttt{0x30}, \texttt{0x31}, \ldots, \texttt{0x39}\} \cup \{\texttt{0x61}, \texttt{0x62}, \ldots, \texttt{0x7A}\}$$