A3: Simulering af cache og evaluering af ydeevne

Computer Systems 2020
Department of Computer Science
University of Copenhagen

Finn Schiermer Andersen

Due: Søndag, 25. Oktober, 16:00
Version 0.2 (mindre skitse end v 0.1)

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.

Opgaven går ud på

- Tilføjelse af en cache til den eksisterende simulatorkode
- Evaluering af cachens betydning for maskinens ydeevne for nogle udvalgte programmer

1 Tilføjelse af en Cache

1.1 Overblik

Der skal tilføjes en model af en cache til den udleverede simulatorkode.

Den modellerede cache skal:

- Have en størrelse på 16 Kb.

- Have 8 ord (64 bytes) pr cache-block
- Være 4-vejs associativ
- Have skrive politik: "writeback" og "write-allocate"
- Bruge "LRU replacement"

Modellen skal skrives i 'C'. I skal altså ikke holde jer til den meget begrænsede delmængde af 'C' som var til rådighed for A2. Modellen skal ikke forsøge at afspejle hardwares funktionsmode i detaljer.

Bemærk at der er tale om en enkelt delt cache mellem instruktioner og data, ikke to forskellige caches.

Modellen skal styre returværdierne fra funktionerne `memory_access()` og `memory_read_into_buffer()` erklæret i `memory.h` og implementeret i `memory.c`.

Returværdierne fra disse to funktioner skal være "true" hvis et opslag kan fuldføres i den aktuelle clk-cyklus og "false", hvis det ikke kan fuldføres. Et cache-miss skal således resultere i at der returneres "false" i et antal efter hinanden følgende clk-cykler.

Den udleverede simulatorkode er organiseret således, at hvis der returneres "false" fra en af de nævnte funktioner, så vil den relevante del af pipelinen "stalle" og dermed cyklus efter cyklus fremsætte præcis den samme forespørgsel til cachen - indtil der returneres "true" og afviklingen kan fortsætte.

I den udleverede kode styres disse returværdier af to hjælpefunktioner `cache_access()` og `cache_miss_update`, og det er jeres opgave at ændre disse to funktioner til noget der giver en korrekt model af cachens opførsel.

I finder en mere præcis beskrivelse af de to funktioner i filen `memory.c`

Det er udelukkende nødvendigt at ændre i `memory.c`

Implementationen kan frit placeres i `memory.c`, eller man kan vælge at tilføje den i nye filer dedikeret til formålet.

I x8prime understøttes det at en instruktion eller data kan være placeret i to cache blokke, startende i den ene blok og fortsættende i den anden. Det behøver I ikke modellere korrekt. Det er tilstrækkeligt at basere modellen af cachens opførsel på adressen på begyndelsen af det efterspurgte data eller den efterspurgte instruktion, og ignorere at det efterspurgte område strækker sig ind i en efterfølgende cache blok.

1.2 Håndtering af cache-miss (forbiere)

Cachen kan være "parat" eller "optaget". At en cache er "optaget" dækker over at den læser eller skriver en cache-blok til lageret. Hvis cachen er "optaget" skal funktionerne `memory_access()` og `memory_read_into_buffer()` returnere false. Hvis cachen er "parat" skal de to funktioner afspejle om opslaget vil være et "hit" eller "miss".

Når en cache-bloks indhold er up-to-date med hensyn til lageret siger man at cache blokken er "clean". Hvis en cache-bloks indhold ikke er up-to-date med til lageret siger man at cache blokken er "dirty".

I en cache med "writeback" skrive-politik placerer man skrive-data direkte i cachen, men udskyder den endelige skrivning til lageret til det er nødvendigt.

Når en cache blok hentes fra lageret som et resultat af et "cache-miss" er den hentede blok "clean". Skrivning til en cache blok gør den pågældende block "dirty".

Ved et cache-miss skal det modelleres at den ønskede cache-blok hentes fra lageret og at eventuelle "dirty" data skrives til lageret. Efter et "miss" til en "clean" blok, skal cachen være "optaget" i 12 clk-cykler. Efter et "miss" til en "dirty" blok, skal cachen være "optaget" i 16 clk-cykler.

Den udleverede kode håndterer allerede skift mellem "optaget" og "parat" og timingen deraf (de 12 hhv 16 clk cyklers ventetid).

1.3 Interaktion mellem data tilgang og instruktionshentning

Cache miss håndtering for hhv instruktioner og data er ikke adskilte, men deler kredsløb (structural hazard). Derfor kan instruktions hentning ikke foregå hvis cachen er "optaget", selvom den er "optaget" som konsekvens af et "miss" for data tilgang. Omvendt: Data tilgang kan ikke foregå hvis cachen er "optaget", selvom den er "optaget" som konsekvens af et "miss" for instruktionshentning.

Det kan forekomme at der i samme maskin cyklus laves opslag i cachen efter både instruktioner og data, og det kan ske at begge opslag udgør et "miss". I dette tilfælde skal cachen opfører sig som om opslaget efter instruktioner sker *før* opslaget efter data. Dette er trivielt opfyldt i den udleverede kode, derved at main() altid kalder `memory_read_into_buffer()` før `memory_access()`.

2 Evaluering af ydeevne

Der udleveres nogle ".prime" programmer som skal køres med den implementerede cache og forskellige inddatasæt. Disse inddatasæt udleveres også. I skal rapportere antallet af data-cache-opslag, antallet af data-cache-miss, CPI (cycles per instruction) og AMAT (average memory access time for data-tilgang) for kørsler med de forskellige inddatasæt.

I kan se bort fra instruktionshentning, da de programmer vi bruger er så små, at det ikke har betydning for ydeevnen.

Resultaterne skal holdes op imod en tænkt maskine uden cache der har en AMAT på 12 clk-cykler, og følgende spørgsmål skal besvares: Hvor meget hurtigere er jeres maskine med cache i forhold til den tænkte maskine uden cache for de udleverede programmer og inddatasæt?

3 Bedømmelse og rapportering

De forskellige dele vurderes omtrent som følger

- implementation af model af cache, inklusiv test af samme (40%)
- evaluering af betydning for ydeevne (20%)
- rapport (40%)

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 om evaluering af betydning for ydeevne er omkring 3 sider og den må ikke overskrive 5 sider.

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 simulationen af cachen virker.

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 præsenteret 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:

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

4 Gode råd

Bemærk at det kun er *effekten* af en cache, der skal modelleres. En rigtig cache vil indeholde værdier, men det udleverede kode sørger allerede for at modellere læsning og skrivning fra lageret korrekt, så det behøver jeres kode ikke at tage hensyn til. Det er kun nødvendigt for jer at modellere de aspekter af cachens opførsel som udmøntes i om der er hit eller miss.

Det indebærer at I for hver cache-blok er nød til at vedligeholde følgende data

- Blokkens "tag" eller evt start adresse, hvis I finder det simplere.
- "valid". Angivelse af om blokken er gyldig.
- "dirty". Angivelse af om blokken er "dirty".
- Last access. Tilstrækkelig med data til at kunne implementere "LRU replacement".

Med hensyn til "LRU replacement", så anbefales det at bruge følgende implementation: Lav en tæller der inkrementeres ved hver eneste opslag i cachen, og gem værdien af denne tæller i "last access" feltet, når cache-blokken tilgås. Det gør det nemt at finde den ældste cache-blok, som er den der skal erstattes. Man vil naturligvis aldrig implementere LRU på den måde i en virkelig cache (det kræver alt for meget data). Men vi er kun interesseret i at modellere *effekten* af "LRU replacement", og så er det en fin fremgangsmåde.

Erklæringer af egnede datastrukturer er allerede placeret i den udleverede kode.