

A6: Maskinnær Optimering

Jeg har implementeret en prioritets-kø i C. Opgaven er at få den til at køre hurtigere!

En prioritetskø understøtter to operationer:

```
Add(value); // tilføj 'value' til prioritetskøen  
Take();      // udskriv og fjern mindste element i prioritetskøen  
Take(N);     // som N * take()
```

Vi udleverer:

- En Riscv simulator, forbedret
- Implementation af prioritetskøen i C, inklusiv optimeret I/O så du ikke skal tænke på det
- En inddata generator - som også kan generere tilsvarende uddata til tjek

Besvarelse

Besvarelsen skal indeholde:

- En kort rapport. Rapporten skal indeholde:
 - En kort præsentation af hvordan man har løst opgaven og hvorfor
 - En analyse af om løsningen også vil være hurtigere for større datasæt end de, der praktisk kan afvikles indenfor rammerne af opgaven.
- En hurtigere implementation end den udleverede
- Dokumentation for at denne implementation virker
- Dokumentation for at denne implementation er hurtigere (se nedenfor).

Den udleverede implementation

Er baseret på et radix-træ. Kodet uden særlig opmærksomhed på ydeevne.

Indsættelse og udtagning i rækkefølge har $\log(B)$ asymptotisk kompleksitet, hvor B er antallet af elementer i prioritetskøen. Det skulle gerne være optimalt.

Men konstant faktorer har også betydning.

En forbedret simulator

Der er to store forbedringer i vores riscv simulator:

- En timing model. Den beregner hvor mange clock-cycles en programkørsel tager
- Timing modellen er simpel.
 - In-order pipeline med realistisk cache tilgang.
 - 2 niveauer af Cache.
 - Yderligere detaljer i opgaveformuleringen.
- Muligheden for at få en eksekveringsprofil
 - Antal gange en instruktion er udført
 - Antal pipeline stalls.
 - Latenstid
 - Lagertilgang, hit-fordeling på L1, L2 og lager.

Adresse på instruktionen	Opstart af instruktionen	Opstart af pipeline	Opstart af L1 cache	Opstart af L2 cache	Opstart af lageret	Opstart af resultat	Opstart af input data	Opstart af output data	Opstart af output data
10074 :	177	0	1	0	0	0	addi	sp,sp,-16	
10078 :	177	0	1	98	0	1	sw	ra,12(sp)	
1007c :	177	0	1	100	0	0	sw	s0,8(sp)	
10080 :	177	0	1	100	0	0	sw	s1,4(sp)	
10084 :	177	0	1	0	0	0	mv	s0,a0	
10088 :	177	0	1	0	0	0	li	a5,1	
1008c :	177	0	1	0	0	0	bgeu	a5,a0,100b0	<fib+0x3c>
10090 :	88	0	1	0	0	0	addi	a0,a0,-1	
10094 :	88	0	1	0	0	0	auipc	ra,0x0	
10098 :	88	0	1	0	0	0	jalr	-32(ra)	# 10074 <fib>
1009c :	88	0	1	0	0	0	mv	s1,a0	
100a0 :	88	0	1	0	0	0	addi	a0,s0,-2	
100a4 :	88	0	1	0	0	0	auipc	ra,0x0	
100a8 :	88	0	1	0	0	0	jalr	-48(ra)	# 10074 <fib>
100ac :	88	0	1	0	0	0	add	a0,s1,a0	
100b0 :	177	0	4	100	0	0	lw	ra,12(sp)	
100b4 :	177	0	4	100	0	0	lw	s0,8(sp)	
100b8 :	177	0	4	100	0	0	lw	s1,4(sp)	
100bc :	177	0	1	0	0	0	addi	sp,sp,16	
100c0 :	177	0	1	0	0	0	ret		

Anbefalet fremgangsmåde

Forstå det udleverede program. Det kan være svært. Forslag:

- Opstil et helt lille eksempel på inddata og forventet uddata.
- Gennemløb C programmet på papir med dit minimale inddata.
- Tegn data strukturen undervejs
- Indsæt nogle udskrivninger som kan vise, hvilke valg programmet træffer
- Kør programmet med dit minimale inddata. Hold programmets valg op i mod dine forventninger
- Iterer indtil det giver mening :-)

Når du har dannet et overblik over hvordan programmet virker, kan du forsøge at optimere det. Formodentlig har selve arbejdet med at forstå programmet affødt spørgsmål du kan bruge. Endvidere kan du bruge simulatoren til at lave en eksekveringsprofil. Den kan give en ide om, hvor det kan betale sig at optimere.

Det er også en mulighed at lave sin egen løsning fra scratch. Det kan synes tillokkende bare at starte forfra, hvis det synes svært at forstå den udleverede kode. Men det er også svært. Overvej nøje om det er en god ide.

Det her er en opgave man bør begynde på tidligt.

Spørgsmål og Svar