dConc Aflevering 1, DA6

Christian Zhuang-Qing Nielsen

201504624, christian@czn.dk

November 12, 2016

1 Introduktion

Pointen med den her aflevering er at vise, at man ved hjælp af *concurrency* kan mindske tidsforbruget for programmer. I afleveringen viser jeg netop dette, ved først at sammenligne tidsforbruget i et simpelt program der tæller antallet af primtal i de første 10 mio. naturlige tal, når det ikke er multithreaded i modsætning til når det er.

2 Tidsmåling og kode

For at måle tiden har jeg i selve koden lavet et timestamp ved starten af programmet, og så et nyt timestamp i slutningen, hvorefter den så printer forskellen på de to i millisekunder. Jeg forsøgte i første iteration af lave et program som tog to argumenter: Det tal der skulle gennemsøges (i denne opgave 10 mio.), samt antallet af ønskede tråde. Her kunne jeg dog ikke få det til at virke, så jeg har bare ladet fejlende ligge og så derefter initialiseret antallet af tråde manuelt i en række forskellige klasser, så her beklager jeg for den ulækre kode på forhånd. Jeg har valgt også at vise tiden hvis man i Java selv kører bare én tråd (I modsætning til ikke at skrive noget), hvilket gør programmet langsommere med få millisekunder som forventet.

Ved at måle tidsforbruget ved et antal af tråde fra 0 til 10, har jeg så lavet en tabel og graf af denne, som kan ses i bilaget.

Ved hjælp af en algoritme som uddeler ækvivalente mængder af tællearbejde til hver tråd, kan de så gøre deres arbejde, og derefter kalde instantieringen af en optæller-klasse, som holder styr på hver af trådenes tællearbejde, og til sidst summerer og printer dem. For at sørge for, at den samme værdi ikke bliver læst af alle på en gang (således at der i stedet for at blive summeret bliver overskrevet), har jeg gjort feltvariablen i denne klasse *volatile*. Eftersom jeg har målt tiderne på min stationære computer, og at denne har 4 kerner, vil forbedringerne kun marginalt blive bedre efter 4 tråde, hvilket er grunden til at grafen flader ud dér. Flere tråde bruger computerens ressourcer bedre, og Intels

hyperthreading-teknologi spiller også en rolle i dette tilfælde, hvilket beskriver den forsatte forbedring af tiden. Blev man ved med at lave flere og flere tråde, så vil instantieringen og kørslen af disse overgå forbedringerne, og dermed ville effektiviteten falde igen. Grafen vil da blive en parabel (eftersom man jo ikke kan få uendelig performance uden understøttende hardware). Havde min computers CPU kun haft én kerne, ville tidsforbedringerne havde været trivielle, eftersom den ikke reelt er i stand til at udføre flere opgaver på samme tid.

3 Bilag

3.1 Kode

Koden for disse programmer er vedhæftet i samme arkivfil som denne .pdf.

3.2 Figurer

Table 1: Effektiviteten af programmerne ved flere tråde.

Amount of threads	**Time in ms**	**Efficiency increase compared to not using conc.**
0	6970	0%
1	6976	-0,086%
2	4456	36,69%
3	3391	51,35%
4	2687	61,45%
5	2439	65,08%
6	2164	68,95%
7	2009	71,18%
8	1955	71,95%
9	1950	72,023%
10	1903	72,697%

Figure 1: Graf af table 1