Threaded Merge Sort

Kenan de Vries

2VA

27/02/2023

Inleiding

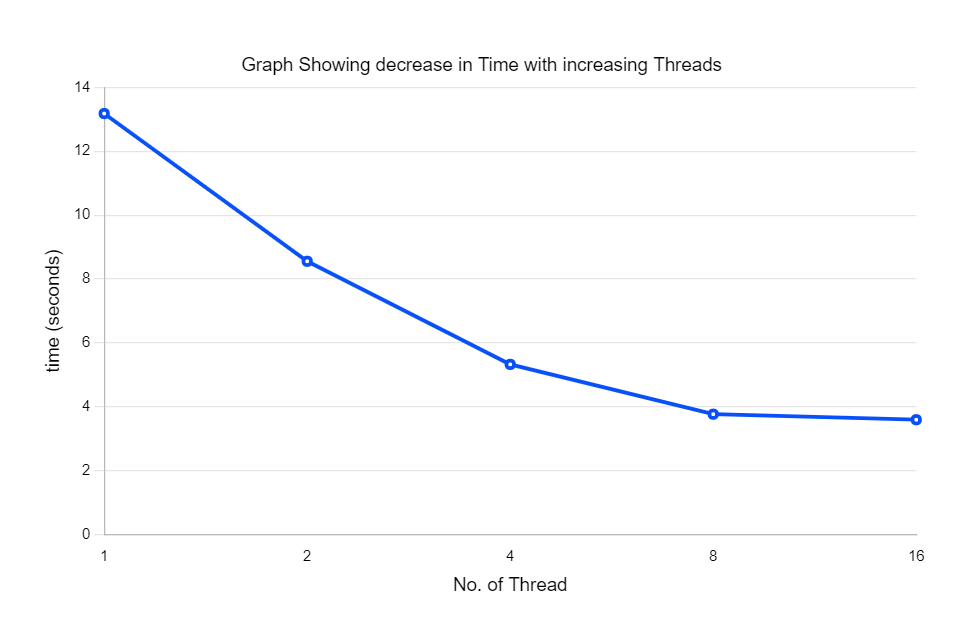
Afgelopen weken hebben wij tijdens het vak High peformance programming geleerd over threads en hoe we deze kunnen benutten om bepaalde algoritmes sneller te kunnen laten uitvoeren. Aan ons is nu de opdracht om van een voorgaande opdracht, de merge sort, een nieuwe versie te maken: de threaded merge sort. Deze versie gebruikt threads om de merge sort te parallelliseren. Dit verslag omvat mijn implementatie van het algoritmen.

Merge sort vs Threaded merge sort

Een basis merge sort is heel simpel in werking. Een array wordt eerst onderverdeeld in kleinere delen, dus 1,2,3,4,8,7,6,5. Wordt bijvoorbeeld een array van 1,2,3,4 en 8,7,6,5. Hierna worden beide arrays apart gesorteerd en aan het einde weer bij elkaar gevoegd. Het enige verschil met een threaded merge sort, is dat meerdere threads (minimaal 2, anders krijg je een normale merge sort) de lijsten opzich nemen. 1,2,3,4 zou dan worden gedaan door thread 1 en 8,7,6,5 door thread 2. Deze worden tegelijkertijd gesorteerd. Als deze beide klaar zijn worden ze aan het einde samengevoegd. Er kan dus overhead zijn omdat de threads op elkaar moeten wachten. In de bijlage wordt dit proces beter uitgelegd en geïllustreerd.

Analyse

Zoals verwacht presteert het algoritme met gebruik van threads over het algemeen aanzienlijk beter dan de basis merge sort. De enige uitzondering hierop is met kleine arrays. Bij een array van 100 200 300 etc. Is het merge sorten met 1 thread aanzienlijk sneller dan bij gebruik van meerdere threads. Dit komt waarschijnlijk omdat het mergen van de gesorteerde lijst aan het einde langer duurt door de hoeveelheid threads. Rond een array van de 900 is de benodigde tijd vrijwel gelijk van alle arrays, al is de single threaded merge sort enkele microseconden slomer dan de rest en is 2 threads optimaal. Bij een array van 3000 en hoger is past echt duidelijk verschil te zien. 1 thread doet er 2x zolang over om te sorteren dan bij gebruik van 2 of meer threads en de hoeveelheid threads vermindert de benodigde tijd. Het interessantste voor mij was een enorme array, namelijk een van 50 miljoen testen. Hieronder zijn daarvan de resultaten te zien die de ware kracht van parallel merge sorten demonstreert:



De complexiteit van de merge sort verschild afhankelijk van of er threading wordt gebruikt. Met threading is het O(N\* Log N). Hierbij is N de lengte van de input lijst. Zonder threading is het O(1/T N\*Log N), waarbij T hier voor threads staat. De time complexiteit blijft hetzelfde. Natuurlijk gaat er door overhead communicatie loss bij de laatste merge stap wat extra tijd bijkomen.

Conclusie

Zoals in de analyse eigelijk al werd gezegd is de standaard merge soort beter voor ‘kleinere’ arrays. Het middenpunt ligt bij de ongeveer 900/1000 elementen, en bij sprake over duizende elementen is de standaard merge soort erg traag vergeleken met de threaded. Als er echter meer dan 8 threads worden gebruikt lijkt er weinig verbetering te zitten in de runtime. Mijn conclusie is dus dat parallel merge soort beter is bij arrays boven de 1000, en dat het eigenlijk geen nut heeft om meer dan 8 threads te gebruiken voor dit algoritmen omdat de performance nauwelijks omhooggaat.

Bijlages:

