Projekt Programowanie Współbieżne 2021/22

Daniel Barczyk

November 2021

1 Abstrakt

Zadanie, które chcę zrównoleglić, to zadanie na liczenie otoczki wypukłej ('L: Pole wypukłej otoczki' z 'ASD2, 2020/21'). Rozwiązanie przebiega w 2 etapach. Najpierw należy posortować punkty, co można zrobić quicksortem z biblioteki standardowej w złożoności $O(n \log n)$. Będę próbował przyśpieszyć ten etap za pomocą mergesorta, w którym podzadania zostaną podzielone między wątki. Następnie w złożoności $O(n)$ idąc po posortowanych punktach sprawdzamy, czy po dodaniu następnego punktu całość będzie dalej skręcać w lewo i jeśli nie usuwamy ostatni wzięty punkt aż będą. Pomysł to podzielić dolną część otoczki oraz górną część między wątki, a następnie je połączyć między sobą. Jako, że złożoność 2 etapu jest istotnie mniejsza niż pierwszego, będę badał jak duży wpływ będzie miał ten podział na całość czasu wykonania programu.

2 Zadanie

Pole wypukłej otoczki Dany jest zbiór punktów na płaszczyźnie. Wyznacz pole powierzchni jego wypukłej otoczki.

2.1 Wejście

Pierwsza linia wejścia zawiera liczbę całkowitą z – liczbę zestawów danych, których opisy występują kolejno po sobie. Opis jednego zestawu jest następujący: W pierwszym wierszu zestawu danych znajduje się liczba naturalna n ($3 \leq n \leq 300000$), oznaczająca liczbę punktów. W n kolejnych wierszach podane są współrzędne punktów – dwie liczby całkowite z przedziału [-109,109]. Nie wszystkie punkty są współliniowe.

2.2 Wyjście

Dla każdego zestawu należy wypisać jedną liczbę całkowitą, będącą dwukrotnością pola szukanej wypukłej otoczki.

3 Etap 1: sortowanie

3.1 Metoda zrównoleglenia

Zamiast quicksorta w $O(n \log n)$ korzystam z mergesorta, który rekurencyjnie sortuje najpierw pierwszą połowę w nowym wątku, a drugą rekurencyjnie w tym samym. Następnie czeka, aż nowy wątek skończy liczyć pierwszą połowę i łączy obie połowy. Teoretyczna złożoność takiego rozwiązania, przy nieograniczonej liczbie wątków, to $n + n/2 + n/4 + ... + 1 \in O(n)$, ale w praktyce liczba wątków jest skończona i tworzenie nowego wątku wiąże się z pewnym narzutem wydajnościowym. W związku z tym, dla małych rozmiarów tablic będziemy korzystali z quicksorta, ponieważ wtedy narzut z tworzenia nowego wątku przekracza zysk uzyskany z zrównolegnienia.

3.2 Wyniki

Następujące wyniki są uśrednionym czasem działania programów, mierzonym za pomocą komendy 'time', na 10 testach z dużymi liczbami. Dostęp do maszyn zdalnych odbywał się przez 'ssh'. Quicksort oznacza std::sort, a Mergesort to zrównoleglona funkcja, gdzie liczba w nawiasach wskazuje na rozmiar tablicy, poniżej którego używany jest Quicksort.

	Quicksort	Mergesort(512)	Mergesort(1024)	Mergesort(2048)
local	real 0m0,154s user 0m0,150s sys 0m0,005s	real 0m0,131s user 0m0,205s sys 0m0,070s	real 0m0,115s user 0m0,197s sys 0m0,036s	real 0m0,111s user 0m0,185s sys 0m0,031s
student	real 0m0.175s user 0m0.163s sys 0m0.012s	real 0m0.140s user 0m0.221s sys 0m0.221s	real 0m0.132s user 0m0.242s sys 0m0.138s	real 0m0.128s user 0m0.311s sys 0m0.056s
miracle	real 0m0.326s user 0m0.320s sys 0m0.004s	real 0m0.259s user 0m0.425s sys 0m0.239s	real 0m0.240s user 0m0.352s sys 0m0.191s	real 0m0.229s user 0m0.418s sys 0m0.044s

Dokonałem również testów dla większych fragmentów, ale nie zapisałem tych danych, ponieważ miały one podobne czasy do 'Mergesort (2048)'. Pierwsza obserwacja jest taka, że do fragmentów rozmiaru 2048 działanie programu przyśpieszało, jeśli sortowaliśmy je Quicksortem, zatem narzut wydajnościowy spowodowany tworzeniem nowego wątku jest większy niż koszt posortowania tablicy zawierającej 2048 elementów. Jeżeli nawet dla małych fragmentów będziemy tworzyć nowe wątki, to nasz Mergesort będzie znacznie wolniejszy od Quicksorta (real 0m1,413s; user 0m1,083s; sys 0m4,150s). Kilka wniosków:

1. Udało się otrzymać przyśpieszenie rzeczywistego czasu wykonania o średnio niecałe 30%.
2. Całkowity czas pracy procesora, dla najszybszego wariantu, wzrósł o średnio 40%.

3. Na maszynie lokalnej wszystkie programy działały najszybciej, miracle był najwolniejszy.

4 Etap 2: liczenie otoczki

Jako, że druga część liczenia zadania jest liniowa nie jest możliwe znaczne zrównoleglenie, które zredukowałoby złożoność czasową obliczeń. Górna i dolna część otoczki są niezależne, dlatego możemy je obliczyć bez potrzeby połączenia wyników. W samej dolnej otoczce możliwe jest wyłącznie podzielenie tablicy punktów na kilka fragmentów i policzenie częściowej otoczki na każdym podfragmencie, a następnie połączenie ich. W pesymistycznym przypadku nie uzyskamy przyśpieszenia, dlatego dla tej części skupię się na policzeniu górnej i dolnej otoczki osobno i połączeniu ich wyników.

5 Podsumowanie

Łącząc zrównoleglenia w obu etapach otrzymujemy następujące wyniki dla całego zadania, liczone tak samo jak poprzednio:

	Bez zrównoleglenia	Zrównoleglony
local	real 0m0,206s user 0m0,193s sys 0m0,012s	real 0m0,108s user 0m0,230s sys 0m0,014s
student	real 0m0.528s user 0m0.225s sys 0m0.004s	real 0m0.327s user 0m0.222s sys 0m0.015s
miracle	real 0m0.404s user 0m0.387s sys 0m0.016s	real 0m0.213s user 0m0.417s sys 0m0.100s

Jak widać zrównoleglając oba etapy liczenia otoczki wypukłej udało się utrzymać znaczne przyśpieszenie. Zrównoleglony program działa średnio o 47% krócej od programu niezrównoleglonego! Dalej obserwujemy zwiększone zużycie czasu procesora.