PW projekt sprawozdanie

Krótkie omówienie rozwiązania zadania Wyprzedaż

Sortujemy wszystkie przedziały po ich wartości (malejąco), wtedy możemy brać je zachłannie (jeśli po dodaniu kolejnego przedziału do wyniku na prefiksie, nadal można wziąć wszystkie wybrane przedziały, to ten przedział także bierzemy). Sprawdzenie czy dany przedział można wziąć robimy zachłannie, poprzez posortowanie przedziałów po końcach i przyporządkowywanie liczbom przedziału który kończy się najwcześniej, a nie jest jeszcze wzięty.

Interpretacja rozwiązania

Można myśleć o tym rozwiązaniu, jak o szukaniu największego (leksykograficznie) działającego wektora n bitów (1 gdy bierzemy i-ty przedział 0 jak nie), z możliwością sprawdzenia czy wektor działa w czasie $O(n \cdot log\ n)$. Może to przypominać binary search, gdzie za każdą iteracją dostajemy 1 bit informacji (czy szukana wartość jest po lewej czy po prawej w tablicy).

Podstawowy sposób na zrównoleglenie:

Będziemy wprowadzali równoległość do części zadania odpowiedzialnej za znajdywanie wektora bitów, a część odpowiedzialną za sprawdzanie czy dany zbiór przedziałów można wziąć zostawiamy tak jak w niewspółbieżnym rozwiązaniu (funkcja canTake())

Załóżmy, że wiemy że rozwiązanie (wektor bitów czy bierzemy przedział) zaczyna się bitami (1,0,1), w rozwiązaniu niewspółbieżnym sprawdzaliśmy czy następny bit można wziąć, czyli sprawdzaliśmy wektor (1,0,1,1). W rozwiązaniu współbieżnym będziemy sprawdzali równolegle wszystkie możliwe $(2^k - 1)$ wektory które możemy uzyskać, po dodaniu k-bitów do naszego wektora z przynajmniej 1 bitem zapalonym, bo wiemy, że wektor bez zapalonych dodatkowych bitów działa. czyli np. dla k = 2 sprawdzalibyśmy wektory:

- (1,0,1,0,1)
- (1,0,1,1,0)
- (1,0,1,1,1)

Takie rozwiązanie pozwala nam w najlepszym wypadku (gdybyśmy mogli tworzyć wątki natychmiastowo i kopiować do nich dane natychmiastowo) usprawnić nasz czas działania do $\frac{T}{log\ k}$, gdzie T to wyjściowy czas działania. Jednak w praktyce tworzenie wątków jest na tyle nieefektywne, w porównaniu do czasu potrzebnego na sprawdzenie 1 wektora (należy pamiętać że rozwiązanie jest $O(n^2 \cdot log\ n)$ i dane mamy rzędu 10^5 więc sprawdzanie wektora w czasie $O(n \cdot log\ n)$ jest całkiem szybkie), że już dla $k = 4$, czyli 16 wątków, program działa wolniej niż dla $k = 3$.

Zatem granicą tego rozwiązania jest 3-krotne przyspieszenie, a to i tak przy założeniu, że możemy szybko tworzyć wątki (co jest nieprawdą)

Heurystyki

Chciałem sprawdzić 2 pomysły jak jeszcze można podobnie wykorzystać równoległość w tym zadaniu, można je wprowadzać równolegle do głównej metody (tak mam zaimplementowane). Minusem poprzedniej metody było niewielkie maksymalne przyspieszenie, ale za to nie potrzebowaliśmy dużej ilości wątków, żeby je uzyskać (tylko 8). Zatem możliwe, że mamy wolne wątki i można je do czegoś użyć.

Peeking

Polega na sprawdzeniu w przód czy dany bit można zaświecić. Czyli powiedzmy że już wiemy, że rozwiązanie zaczyna się na (1,0,1), główna metoda sprawdza równolegle 3 wektory (1,0,1,0,1), (1,0,1,0,1), (1,0,1,0,1). Wtedy peeking polega na sprawdzeniu wektorów np.: (1,0,1,0,0,1), (1,0,1,0,0,0,1), (1,0,1,0,0,0,0,1), jeśli jakiegoś z nich nie możemy wziąć, to wiemy że nigdy nie będziemy mogli wziąć danego bitu. Peeking może w oczywisty sposób przyspieszyć nasze rozwiązanie, jeśli wektor rozwiązania posiada dużo 0 i są one nagromadzone w grupy.

Faith leaps

W tym sposobie wierzymy, że można wziąć następne k bitów i sprawdzamy czy można wziąć $k+1, k+2, \ldots$, czyli sprawdzamy wektory (1,0,1,1,1,1), (1,0,1,1,1,1,1), (1,0,1,1,1,1,1,1). Jeśli któryś z nich jest możliwy do wzięcia, to możemy pominąć wszystkie wektory krótsze, bo wiemy, że ten będzie największy. Ta metoda jest użyteczna, jak wiemy że weźmiemy dużą część przedziałów (czyli będzie dużo 1).

Programy

Napisałem 3 programy:

- wyprzedazNew.cpp (wn)
  Rozwiązanie nierównoległe, dostaje OK na satori
- wyprzedazOMP.cpp (womp)
  Implementuje tylko główną metodę zrównoleglenia korzystając z OMP, przyjmuje jako argument $log\ k$, gdzie k to jest liczba wektorów sprawdzanych na raz.
- wyprzedazParallelFinal.cpp (wpf)
  implementuje główną metodę, oraz peeking i faith leaps. Bierze 3 argumenty, $log\ k$ tak jak poprzedni, liczbę wektorów sprawdzanych przy metody Peek, oraz liczbę wektorów sprawdzanych przy użyciu Faith Leaps

---

Testy:

Jako, że chciałem wyznaczyć optymalne argumenty programów, to każdy test jest odpalany na:

- wn
- womp [2...4]
- wpf [2, 3] [0...6] [0..6]
  Zatem faktycznie programów jest koło 100, więc testów jest odpowiednio mniej, żeby dało się je przeprowadzić

Sprawdzałem 4 zestawy danych $(z = 10)$, w każdym zestawie dla każdego z mamy 10 000 przedziałów, zestawy różnią się maskymalną długością (length) losowanych przedziałów oraz szerokością (breadth) przedziału z którego losujemy końce.
Dla każdego testu obliczyłem średni czas wykonania i poniżej zamieszcze po 5 najszybszych programów oraz program niewspółbieżny.

Test 1 (breadth = 500, length = 60)

Top 5:
womp3t: 254.73 ms
womp2t: 301.33 ms
wpf_2_6_1t: 421.8 ms
wpf_3_3_0t: 422.23 ms
wpf_2_3_0t: 436.49 ms
wn: 468.37ms

maksymalne przyspieszenie: 1.84 (główna metoda bez heurystyk k = 8)

Test 2 (breadth = 3000, length = 100)

Top 5:
wpf_2_5_5t: 1072.9 ms
wpf_3_6_5t: 1080.5 ms
wpf_2_5_4t: 1085.8 ms
wpf_2_4_5t: 1098.2 ms
wpf_2_4_4t: 1106.3 ms
wn: 2576.8ms

maksymalne przyspieszenie: 2.4 (główna metoda z k = 4, Peek 5 w przód, Faith leap 5 w przód)

Test 3 (breadth = 500, length = 500)

Top 5:
womp3t: 605.97 ms
wpf_2_5_1t: 660.13 ms
wpf_2_5_2t: 665.96 ms
wpf_2_6_2t: 674.03 ms
wpf_2_6_3t: 679.84 ms
wn: 1397.5ms

maksymalne przyspieszenie: 2.31 (główna metoda z k = 4, Peek 5 w przód, Faith leap 1 w przód)

Test 4 (breadth = 3000, length = 300)

Top 5:
wpf_2_5_4t: 1101.2 ms
wpf_2_6_3t: 1118.1 ms
wpf_2_5_3t: 1144.2 ms
wpf_2_6_2t: 1188.1 ms
wpf_2_4_5t: 1189.3 ms
wn: 2965.5ms

maksymalne przyspieszenie: 2.69 (główna metoda z k =4, Peek 5 w przód, Faith leap 4 w przód)

Podsumowanie

Najlepsze przyspieszenie jakie udało się uzyskać w testach to 2.69, dla testu gdzie losowaliśmy długie przedziały na długim zakresie końców. Nie jest to duże usprawnienie co pokazuje jak nieefektywne jest paralelizowanie czegoś przypominającego bin search. Jednak możemy zauważyć, że heurystyki w większości testów pomogły, a 2.69 jest bliskie 3, które jak wcześniej zauważyłem jest najlepszym (praktycznie mocno zawyżonym) przyspieszeniem jaki moglibyśmy wyciągnąć z głównej metody równoleglenia.

Możemy też zauważyć, że te usprawnienia zdobyliśmy przy użyciu nie największej liczby wątków odpowiednio: 8, 14, 10, 13. Więc może to nie być złym rozwiązaniem na maszynie z małymi możliwościami na zrównoleglenie.

Hubert Bernacki, 26.11.2023