Sprawozdanie  
Badanie Algorytmu równoległego sortowania kubełkowego

Marcin Sośniak

1. Środowisko  
   testy algorytmow 1 i 3:
   * odbyly sie na maszynie pod adresem vnode-01.dydaktyka.icsr.agh.edu.pl, który posiada 4 procesory
   * wykonane na tablicy o dlugosci 2 000 000 (testy głowne)
   * kazdy test byl powtórzony 50 razy, a potem uśredniony
   * można też zauważyć pewne nie równości między przyspieszeniem generowania tablicy. Jedyna obserwowalny wniosek na to jest ‘kaprys’ maszyny, gdyz ten sam kod jednego dnia moze nam dac idealnie skalujacy sie generowanie maszyny (jak na laboratoriach), albo skalujacy sie z koncowa efektywnoscia mnie wiecej 0.8 0.9. Nie udało sie znaleść reguły dlaczego tak się dzieje i jak to wyeliminować.
   * Niestety ze względu na złożoność pamięciową algorytmów (w tym sortowania kubełkowego jako takiego) i małą ilość ramu, czasy wykonania są niskie, co za tym idzie błędy wynikające z mierzenia czasu mogą być spore.

testy algorytmu 2:

* a

1. Algorytm własny: Algorytm 1  
   Każdy z wątków przechodzi całą tablice, ale posiada rozłączny z innymi zakres kubełków. Po przejsciu nastepuje sortowanie kubełków, po czym scalanie tablicy, poprzez nadpisanie oryginalnej tablicy.

Zauważmy więc iż wątki sobie nie przeszkadzają, za wyjatkiem jednego punktu: ustalenia gdzie ktory kubelek nalezy zapisac. Ta czesc algorytmu jest niestety sekwencyjna i nie wymyslilem obejscia tego problemu, na szczęście jest ona *bardzo* szybka (znacząco poniżej błedu mierzenia czasu). Oprócz tego jednego momentu wątki wykonują swoje operacje bezpiecznie, a więc nie ma potrzeby na żadną ochronę dostepu do danych, gdyz ten etap i tak mozna wykonac tylko sekwencyjnie.

Wykorzystałem N^(2/3) kubełków

Zlożoność teoretyczna wyności O(N+N^2/k+k) gidze nasze k = N^(2/3) = O(N+N(4/3) + N^(2/3)) = O(N^4/3)

Zauważmy też iż ten algorytm zrównolegla tylko zapis do kubełków, a nie odczyt z tablicy, co będzie zawsze stanowiło jego dolne ograniczeni.e

Implementacja zaklada obiekty, ktore przechowuja wszystkei potrzebne informacje zeby wykonac dany krok algorytmu i sa przydzielane watkom. Takie rozwiazanie jest wygodne programistycznie i wmiare eleganckie, ale ma swoje wady efektywnosciowe. Mozna przyjac iz poprstu stosujemy jakis rodzaj zrownoleglonej petli for, do sortowania kubelkow, przy odpowiednim algorytmie (np. guided) wszystkie watki powinny byc rowno wykorzystane. W mojej implementacji w etapie sortowania kubelków tak nie jest, jeśli jeden z obiektow bedzie zawierał bardziej czasochlonne kubełki, to wszystkie inne będa musiały na niego poczekać.

1. Próby usprawnienia algorytmu 1

Okazuje się iż najprawdopodniej przyczyna nie idelanego skalowania sie rozkładania do kubełkow jest odczyt z tej samej tablicy przez wiele watkow. Z drugiej strony z testów bez odczytu z tablicy (generowanie losowej liczby), makysmalnie przyspieszenie wynosi miedzy 2 a 3 (z bardzo dużą wariancja wynikow) dla 4 watków, co jest rozsądne wliczając jego nature, gdzie wszystkie wątki przechodzą cała tablice, i jedyny zysk mamy z tego, ze im więcej wątków tym więcej liczb każdy wątek może zignorować a zapisy do kubelków są równoległe.

Na szybkość odczytu nie miało wpływu czy wątki zaczynały czytać w tym samym miejscu czy w różnych.

1. Skalowalność

Z wykresu 5.5 widać iż efektywna złożność obliczeniowa, jest bliższa złożoności oczekiwanej, niż teoretycznej. Może to wynikać z faktu iż 2 000 000 elementów to nie jest tak dużo elementów a narzut z samego faktu wywołania w funkcji sortującej może być jeszcze na tyle duży, że znacząco sam z siebie wpływa na czas sortowania poszczególnych kubełków.