Michał Gozdera
5.01.2021

Klastrowanie metodą K-średnich

z użyciem Nvidia CUDA

Informacje podstawowe

Używam następujących oznaczeń: N – liczba obiektów wejściowych, k – liczba centroidów, n – liczba współrzędnych obiektów wejściowych.

Zakładam dużą liczbę obiektów wejściowych (N>=512). Jeżeli zajdzie potrzeba uruchomienia programu dla mniejszego N, należy zmienić wartość tc w pliku CudaModule3.cu (#define tc) na mniejszą równą N.

Dostępne są 3 sposoby mierzenia czasu – poszczególnych części algorytmu (#define TM), iteracji algorytmu - jedno wykonanie pętli głównej (#define LTM), całego algorytmu (#define GTM). Należy odkomentować odpowiednie define'y w plikach CudaModule.cu, CudaModule2.cu, Cudamodule3.cu, NoCudaModule.cpp.

Pili testowe, których używałem do przeprowadzenia pomiarów czasu dla wszystkich algorytmów znajdują się w folderach N_objects_data, n_centroids_data oraz test_data. W pierwszym folderze pliki różnią się liczbą obiektów, w drugim - współrzędnych. Trzeci plik zawiera dane testowe, na których łatwo sprawdzić poprawność działania algorytmu. Aby zmienić liczbę centroidów/klastrów należy zmienić w dowolnym pliku trzecią linię. Dane w pliku: pierwsza linia: N, druga: n, trzecia: k, kolejne N linii: współrzędne każdego obiektu oddzielone spacją. Aby zmienić dane używane w programie, należy przenieść wybrany plik z folderu N_objects_data, n_centroids_data lub test_data do folderu, w którym znajduje się solucja, a następnie zmienić jego nazwę na „data.txt". Można też wygenerować nowy plik. Wyniki zapiszą się w folderze solucji (w plikach testowych).

Opis algorytmów

Zaimplementowane i porównane są 4 algorytmy (pliki CudaModule.cu, CudaModule2.cu, CudaModule3.cu, NoCudaModule.cpp):

1. **NoCudaModule**.cpp – wersja całkowicie sekwencyjna.
2. **CudaModule**.cu – zrównoleglone obie części algorytmu:
    a. Przyporządkowanie obiektów do centroidów: embarrassingly parallel, z użyciem pamięci współdzielonej. Dla każdego obiektu uruchamiany jest oddzielny wątek, współrzędne obiektu przechowywane są w pamięci współdzielonej – duża oszczędność czasu ze względu na fakt, że współrzędne używane są wielokrotnie podczas obliczania (kwadratu) odległości od centroidów.
    b. Obliczenie nowych centroidów: indeksy obiektów są sortowane po numerach centroidów do jakich należą (sort_by_key, gdzie key to numer centroidu). W wyniku operacji dostajemy dwie tablice: posortowane numery centroidów dla wszystkich obiektów oraz odpowiadające im indeksy obiektów. Następnym etapem jest przygotowanie danych do redukcji – wyliczamy gdzie w wynikowej tablicy zaczynają się i gdzie kończą centroidy, a kolejne współrzędne ustawiane są w kolejności:

współrzędne nr 0 obiektów w centroidzie 0,
współrzędne nr 1 obiektów w centroidzie 0,
…,
współrzędne nr n-1 obiektów w centroidzie 0,
… ,
…,
współrzędne nr 0 obiektów w centroidzie k-1,
współrzędne nr 1 obiektów w centroidzie k-1,
… ,
współrzędne nr n-1 obiektów w centroidzie k-1.

Z tą uwagą, że jeżeli dany klaster nie zawiera żadnych obiektów, nie będzie uwzględniany w powyższej tablicy. W ten sposób obliczane są sumy na każdej współrzędnej każdego centroidu (i układane kolejno). Analogicznie wyliczane są liczby obiektów przynależnych do danego centroidu. Odpowiednie ułożenie sum i liczności centroidów w tablicach przyspiesza w znaczny sposób następny etap – redukcję. Stosowana jest tzw. segmented reduction (reduce_by_key) – w wyniku dostajemy tablicę sum na każdej współrzędnej (ułożone w odpowiedniej kolejności) oraz tablice liczności kolejnych klastrów. Używany jest algorytm z biblioteki thrust. Aby obliczyć nowe centroidy wystarczy podzielić wartości tablicy sum przez wartości tablicy liczności embarassingly parallel).

3. **CudaModule2**.cu – zrównoleglona tylko część dotycząca przyporządkowania obiektów do centroidów (analogicznie jak w 2a) bez użycia pamięci współdzielonej. Obliczenie nowych centroidów sprowadza się wtedy do jednokrotnego przejścia po tablicach: obiektów (obliczenie sum i liczności klastrów) oraz centroidów (zaktualizowanie wartości).
4. **CudaModule3**.cu – ponownie zrównoleglone obie części:
    a. Przyporządkowanie obiektów do centroidów: analogicznie jak w 2a.
    b. Obliczenie nowych centroidów: równoległa modyfikacja sekwencyjnego algorytmu z punktu 3. Tablicę współrzędnych obiektów dzielimy na tc części (zostało ustalone tc = 512, zakładając dużą – przynajmniej >= 512 – liczbę obiektów wejściowych). Uruchamiane jest tc wątków, każdy niezależnie liczy sumy i liczności klastrów dla każdego centroidu (jeżeli wątek nie znajdzie żadnych obiektów w danym centroidzie – w tablicy wynikowej wpisywane jest 0. Warto zauważyć, że analogicznego problemu nie ma w algorytmie z punktu 2). Obliczone wartości wpisywane są w wynikowej tablicy w sposób ułatwiający późniejszą redukcję. Np. dla tc=2, n=2, k=3 mamy:

$a_0^0,\ b_0^0,\ a_1^0,\ b_1^0,\ a_0^1,\ b_0^1,\ a_1^1,\ b_1^1,\ a_0^2,\ b_0^2,\ a_1^2,\ b_1^2$

gdzie:
a – sumy częściowe obliczone przez wątek nr 0,
b – sumy częściowe obliczone przez wątek nr 1,
górny indeks – numer centroidu,
dolny indeks – numer współrzędnej.

Analogicznie dla liczności klastrów. Następnie przeprowadzana jest redukcja – sumujemy wyniki od poszczególnych wątków i obliczamy nowe centroidy jako średnie arytmetyczne. Używany jest własny algorytm redukcji i pamięć współdzielona.

UWAGA: Pomiary czasu oraz wykresy znajdą się w pliku **KMeans_results_comparison.xlsx.**