K means clustering

Piotr Piesiak

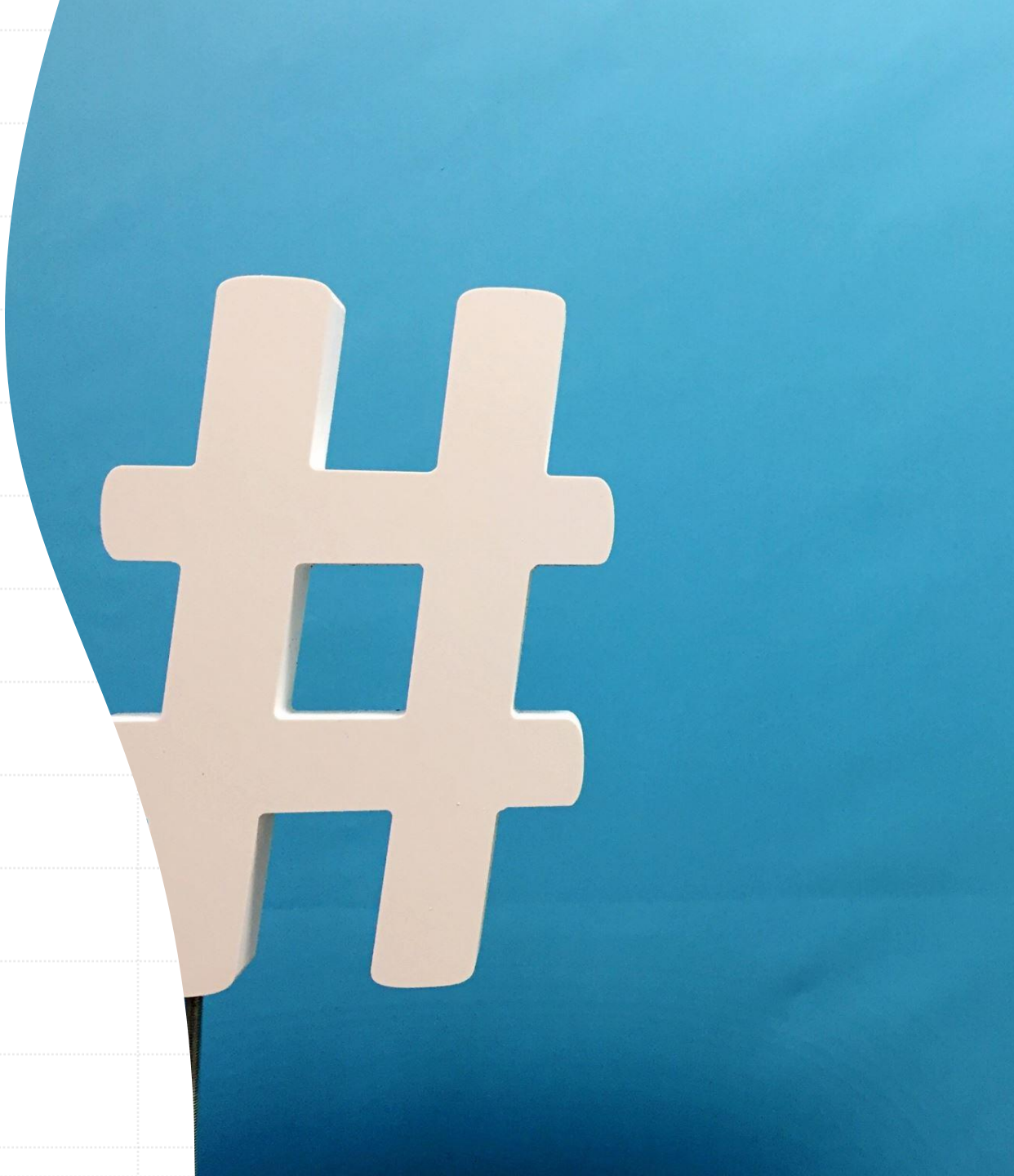
Działanie algorytmu

1. Losowo wybieramy k punktów – “znaczniki”.

2. Przypisujemy każdemu punktowi jego grupę (1,2...,k). Punkt należy do grupy l, jeśli jego odległość do znacznika l jest najmniejsza spośród k znaczników.

3. Obliczamy centroid każdej grupy (środek masy punktów)

4. Powtarzamy krok 2,3 aż obliczone centroidy w punkcie 3. nie będą się zmieniać (lub iterujemy się określoną liczbę razy)



Co można zrównoleglić?

Klasyfikowanie punktów (każdy thread – jeden punkt)

Liczba bloków zależy od liczby punktów (ceiling(points_num / threads_num))

Aby nie czytać centroidów z pamięci globalnej, każdy blok utrzymuje w pamięci współrzędne K centroidów

Uaktualniamy globalne przypisanie grupy dla danego punktu

Obliczanie centroidów - zmodyfikowana redukcja

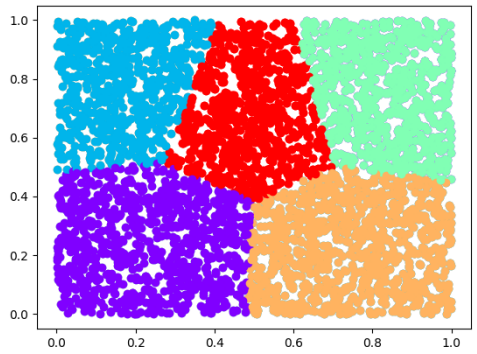
Obliczanie nowych centroidów

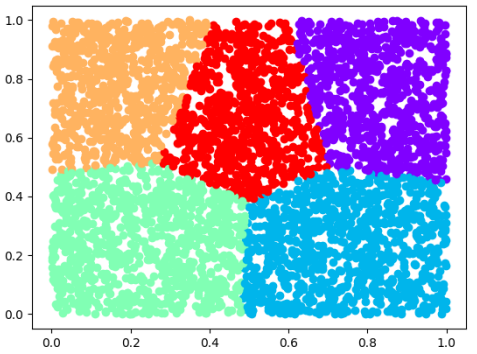
- Nie jest to klasyczny problem sumowania tablicy, chcemy zsumować współrzędne punktów wewnątrz poszczególnych grup
- W każdym wątku robimy pętlę po klastrach (1,…,k) i wewnątrz niej wykonujemy redukcję.
- Każdy wątek (i odpowiadający punkt) przed redukcją dla danego klastra zapisuje do współdzielonej pamięci swoje współrzędne – jeśli aktualny klaster to jego grupa – lub 0 w.p.p.
- Na koniec zapisujemy sumę współrzędnych w danym bloku danego klastra do globalnej tablicy (rozmiar: CLUSTERS_NUM * BLOCKS_NUM)

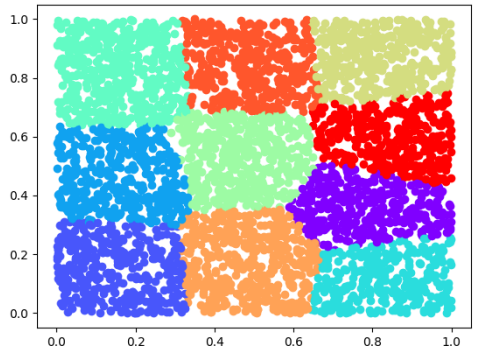
```
temp[3 * local_tid] = (assigned_cluster == c) ? x : 0;
temp[3 * local_tid + 1] = (assigned_cluster == c) ? y : 0;
temp[3 * local_tid + 2] = (assigned_cluster == c) ? 1 : 0;
__syncthreads();
```

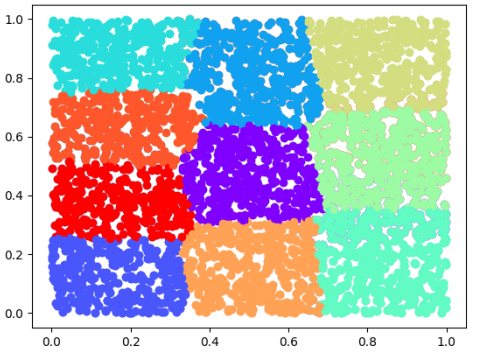
Przykładowe wizualizację pogrupowanych punktów

- 4 tysiące punktów dla 5 i 10 grup









Eksperymenty i wnioski

- Dla niewielkiej liczby punktów (~100) różnice w czasie są niezauważalne, a nawet przemawiają na korzyść k-means na CPU.

- Przy dużej liczbie punktów (~100 000) punktów działa 30 krotnie szybciej. Przy trudnym problemie (100 grup - bardzo duża złożoność 100^liczbapunktów) program na CPU działał ponad 20 sekund, podczas gdy GPU poniżej sekundy.