K-Means (4-Means)

Paralelní a distribuované algoritmy 2022/2023

Projekt 2

Vojtěch Mimochodek (xmimoc01)

Algoritmus

Paralelní K-Means algoritmus funguje na principu iterativního přiřazování datových bodů k nejbližším středům shluků a následného výpočtu nových středů shluků. Tento proces se opakuje do té doby, dokud se dané středy shluků nepřestanou měnit.

Analýza algoritmu

Odvození časové složitosti:

- MPI_Scatter():
  – lineární časová složitost O(n)
  – [(P – 1) / k] , kde *(P-1)* je počet odeslaných zpráv a *k* výstupních linek
- MPI_Gather():
  – lineární časová složitost O(n)
  – [(P – 1) / k] , kde *(P-1)* je počet přijatých zpráv a *k* komunikačních linek

- MPI_Bcast():
  – logaritmická časová složitost O(log n)
  – [$\log_{k+1} P = (\log P) / (\log k + 1)$] , kde *k* je počet linek, P je počet zpráv

- MPI_Reduce():
  – logaritmická časová složitost O(log n)

Před hlavní smyčkou v programu je jednou volána funkce *MPI_Scatter()* k rozeslání N čísel na N procesorů. Toto rozeslání pracuje v lineární časové složitosti *O(n)* pro *n* čísel. Uvnitř hlavního cyklu programu je v každé iteraci volána třikrát *MPI_Reduce()* a dvakrát *MPI_Bcast(),* tedy časová složitost cyklu je logaritmická *O(log n).* Za hlavním cyklem se nachází volání funkcí *MPI_Gather()* k sesbírání výsledků a to v lineární časové složitosti *O(n)*.

Celkovou časovou složitost lze spočítat následujícím způsobem:

$$t(n) = O(n) + O\big(n \cdot (\log n + 2 \cdot \log n + \log n)\big) + O(n) = O(n) + O(n \cdot \log n)$$
$$= \boldsymbol{O(n \cdot \log n)}$$

Odvození prostorové složitosti:

V programu se pracuje s několika vektory a číselnými proměnnými. První vektor *vector<uint8_t> numbers* obsahuje načtenou vstupní posloupnost. Jeho prostorová složitost je tedy lineární *O(n)*. Další proměnné a vektory jsou inicializovány na určité velikosti a jejich prostorové složitosti jsou tedy konstantní.

Lze říci, že celková prostorová složitost je lineární p(n) = **O(n)**.

Cena:

$$c(n) = t(n) \cdot p(n) = O(n \cdot \log n) \cdot O(n) = \boldsymbol{O(n^2 \cdot \log n)}$$

Implementace

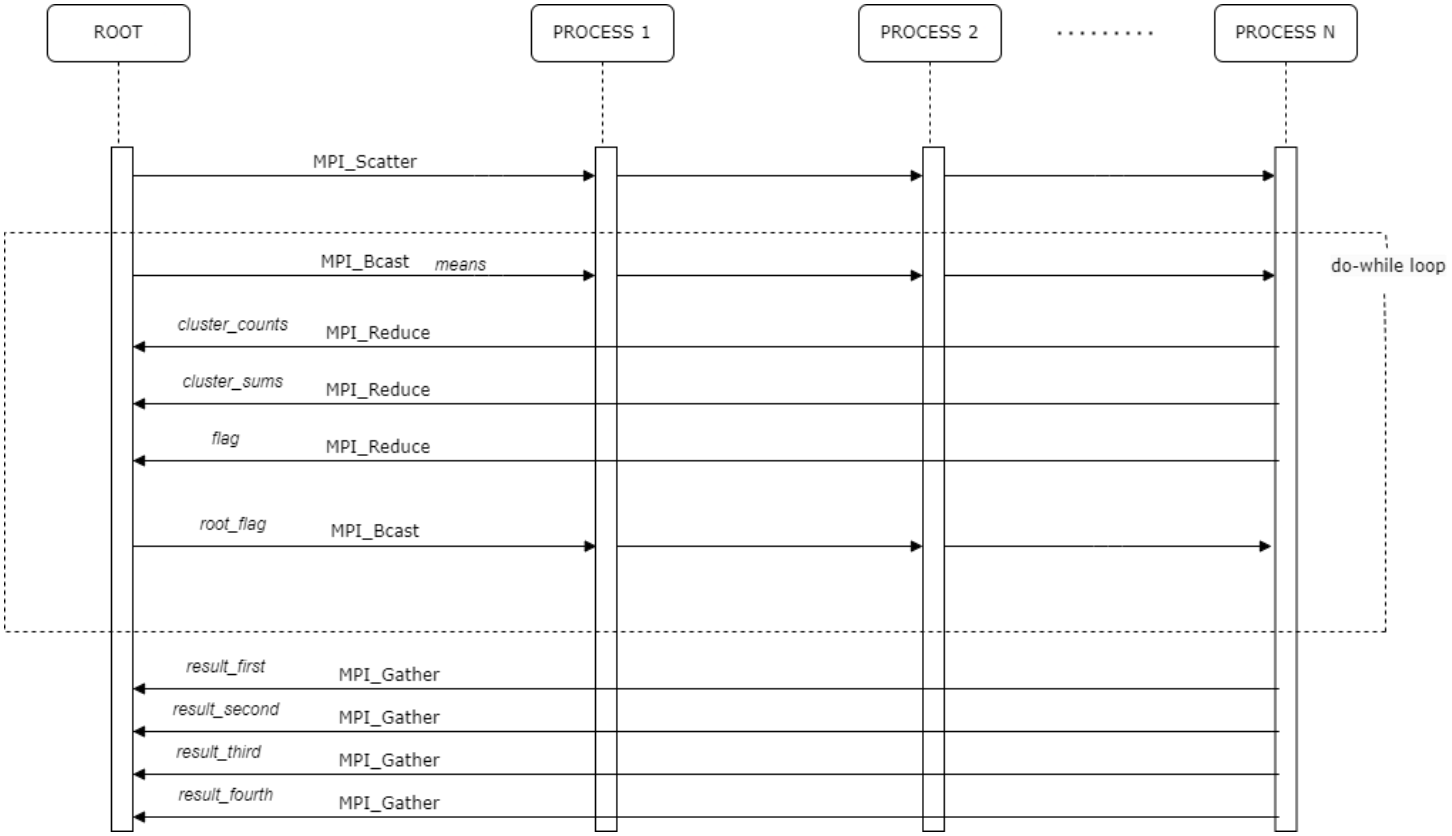
Program nejprve načte ze souboru *numbers* posloupnost čísel v rozmezí 4-32. Následně je provedena kontrola vstupní posloupnosti, zdali bylo načteno opravdu 4-32 čísel a program byl spuštěn s patřičným množstvím procesorů, rovněž v rozmezí 4-32. V případě, že je čísel méně než procesorů, je vypsána chybová hláška a program ukončen. Naopak kdy je procesorů méně je pracováno pouze s prvními N čísly, a to tak aby bylo na N procesorů N čísel.

V další části programu jsou vybrány počáteční hodnoty shluků podle prvních 4 prvků v načtené posloupnosti. Pomocí funkce *MPI_Scatter()* je posloupnost čísel rozdělena a rozeslána na jednotlivé procesy.

V hlavní části se nachází *do-while* cyklus ve kterém skrze *MPI_Bcast()* funkci jsou všem procesům odeslány hodnoty středů jednotlivých shluků. Na základě těchto hodnot je pro číslo, které daný procesor zpracovává vypočítána vzdálenost jako absolutní hodnota čísla od středu. Dle nejmenší vzdálenosti je číslo přiřazeno do shluku. Do pole *cluster_counts* je na index symbolizující shluk kam číslo patří, uložena hodnota 1. Na ostatních indexech jsou hodnoty 0. V poli *Cluster_sums* je na index shluku uložena hodnota čísla. Do pomocné proměnné *flag* je uložena binární hodnota, která je získána na základě porovnání předchozího indexu shluku se současným. Do lokálních proměnných pro jednotlivé procesory jsou uloženy čísla jako zatím dosažené výsledky. Poté je volána třikrát funkce *MPI_Reduce()*. Dvakrát jsou s operací *MPI_SUM* redukovány pole *cluster_counts* a *cluster_sums*. Tak docílíme získání součtů a počtů čísel z jednotlivých shluků. Pomocí těchto hodnot spočítáme nové středy shluků jako průměry. Třetí volání funkce redukce je k redukování hodnot v proměnných *flag* s operací *MPI_LOR*. Díky hodnotě získané touto redukcí určíme, zdali čísla na jednotlivých procesech změnila shluk. Pokud ne, cyklus a výpočet končí. Vzhledem k tomu, že každý proces vykonává daný cyklus sám o sobě, je potřeba mu dát informaci o ukončení iterací vědět. Proto je redukovaná hodnota ze třetí redukce pomocí *MPI_Bcast()* odeslána zpět všem procesům.

V poslední části jsou po ukončení cyklu pomocí operací *MPI_Gather()* sesbírány výsledky do výsledných 4 polí a výsledek vypsán na *STDOUT*.

Komunikační protokol



Obrázek č. 1: Sekvenční diagram

Závěr

V rámci tohoto projektu byla vytvořena implementace algoritmu *parallel 4-Means clustering*. Jak bylo ukázáno, implementace tohoto algoritmu má časovou složitost $O(n \cdot \log n)$, prostorovou složitost $O(n)$ a cenu $O(n^2 \cdot \log n)$.

Řešení bylo testování pomocí scriptu *test.sh*, který je obdobný jako v prvním projektu.