Paralelní a distribuované algoritmy (PRL) – 2022/2023
Projekt 2 – Implementace paralelního algoritmu K-means

Domink Nejedlý (xnejed09)

1 Rozbor a analýza algoritmu

Implementovaný algoritmus 4-means řeší rozdělení vstupních hodnot do čtyř shluků podle toho, jak daleko od středu (počítán jako průměr hodnot, jež shluk obsahuje) daného shluku se nachází. V každé iteraci je každá hodnota zařazena do shluku, k jehož středu zaujímá nejkratší vzdálenost (absolutní hodnota rozdílu středu shluku a dané hodnoty). Pokud pak mezi dvěma po sobě jdoucími iteracemi nedojde ke změně žádného středu ani jednoho ze všech čtyř shluků, je nalezeno řešení a výpočet končí. Jelikož implementovaný algoritmus pracuje se skalárními (jednorozměrnými) hodnotami a pouze čtyřmi shluky ($k = 4$) jistě konverguje v lineárním počtu provedených iterací (počet iterací je shora omezen lineárně) – $\mathcal{O}(n)$ – vzhledem k n vstupním hodnotám. Tohle je dokázáno v článku How Fast Is k-Means? pro případ, kdy $k < 5$ a algoritmus pracuje se skalárními (jednorozměrnými) daty.

Rozeberme celkovou časovou a prostorovou složitost implementovaného algoritmu. Samotné načtení dat – n vstupních hodnot – a vytištění výsledků hlavním procesem nemusíme uvažovat, jelikož obě tyto operace lze provést v lineárním čase $\mathcal{O}(n) + \mathcal{O}(4n) = \mathcal{O}(n)$. Pro vytištění výsledků je uvedeno $\mathcal{O}(4n)$ pro zdůraznění čtyř průchodů polem, které obsahuje identifikaci odpovídajícího shluku pro každé číslo (index tohoto pole odpovídá indexu odpovídající hodnoty v poli se vstupními hodnotami). Po načtení n vstupních hodnot hlavním procesem dochází k inicializaci středů shluků (v konstantním čase, jelikož jsou vždy čtyři) – $\mathcal{O}(1)$ – a k distribuci načtených hodnot mezi všech N běžících procesů. Je předpokládáno, že každý proces má na starosti právě jednu hodnotu, tedy $n = N$. K rozdělení hodnot mezi procesy je využita funkce `MPI_Scatter`, jejíž časová složitost jistě nepřesáhne $\mathcal{O}(n)$. Následně dochází k běhu jednotlivých výpočetních iterací zajišťujících opakované rozřazování hodnot do shluků a přepočítávání středů těchto shluků. Nejprve jsou aktuální středy shluků (vždy pole o čtyřech hodnotách) rozhlášeny z hlavního procesu mezi všechny ostatní procesy pomocí funkce `MPI_Bcast` – logaritmická časová složitost $\mathcal{O}(\log N)$. Následně je v konstantním čase ($\mathcal{O}(1)$) každým procesem odpovídající hodnotě určen patřičný shluk (pomocí indexu 0-3 do pole se středy shluků) a následně jsou pomocí jedné redukce funkcí `MPI_Reduce` získány do hlavního procesu (0) součty hodnot přiřazených do jednotlivých shluků. Ta je provedena v logaritmickém čase – $\mathcal{O}(\log N)$. Další redukcí (funkcí `MPI_Reduce`) jsou pak do hlavního procesu získány počty hodnot v jednotlivých shlucích – opět s logaritmickou časovou složitostí $\mathcal{O}(\log N)$. Hlavní proces poté v konstantním čase – $\mathcal{O}(1)$ – z těchto získaných hodnot vypočítá nové středy shluků a nastaví řídící proměnnou cyklu (v němž probíhají tyto iterace pro rozřazování prvků) `doNextIter` podle toho, zdali se nějaký ze středů po přerozdělení hodnot změnil, či nikoli – `true`, pokud ke změně středu nějakého shluku došlo, jinak `false`. Hodnota této řídící proměnné je pak v logaritmickém čase – $\mathcal{O}(\log N)$ – rozhlášena z hlavního procesu i všem procesům ostatním (aby bylo zajištěno ukončení běhu tohoto cyklu i v nich). Po ukončení běhu tohoto cyklu je s využitím funkce `MPI_Gather` v hlavním procesu shromážděna identifikace odpovídajícího shluku pro každou rozřazovanou vstupní hodnotu. Časová složitost tohoto kroku jistě nepřekročí $\mathcal{O}(n)$. Hlavní proces nakonec vytiskne všechny výsledky (v lineárním čase – $\mathcal{O}(n)$ – jak již bylo zmíněno) a výpočet je úspěšně ukončen.

Z popisu chování implementovaného algoritmu je zřejmé, že časová složitost, vzhledem k tomu, že $N = n$ (tedy počet procesů je roven počtu rozdělovaných hodnot), je

$$\mathcal{O}(1)+\mathcal{O}(n)+\mathcal{O}(n)(\mathcal{O}(\log n)+\mathcal{O}(1)+\mathcal{O}(\log n)+\mathcal{O}(\log n)+\mathcal{O}(1)+\mathcal{O}(\log n))+\mathcal{O}(n) = \mathcal{O}(n \log n) \text{ (linearitmická)},$$

jelikož časová složitost provedení jedné iterace je logaritmická ($\mathcal{O}(\log n)$) a jejich počet je omezen lineárně ($\mathcal{O}(n)$) – časová složitost načtení vstupních hodnot a vytištění výsledků pak nemá na celkovou výslednou časovou složitost žádný vliv (ta bude i s nimi stále linearitmická – $\mathcal{O}(n \log n)$).

Prostorová složitost algoritmu je pak lineární ($\mathcal{O}(n)$), jelikož počet procesorů $N = n$, kde n je počet rozřazovaných hodnot a každý proces pracuje vždy pouze s jednou z nich, přičemž dále disponuje nějakou konstantní – $\mathcal{O}(1)$ – částí paměti pro uchování pomocných hodnot a středů shluků. Výjimkou je pak proces hlavní, který navíc obsahuje pole vstupních hodnot a na konci výpočtu pak ještě pole obsahující identifikaci odpovídajícího shluku pro každou vstupní hodnotu. Pokud je tedy prostorová složitost vyjádřena prostřednictvím těchto dílčích popsaných nároků na paměť je jasně vidět, že je omezena lineárně:

$$n\mathcal{O}(1) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n).$$

Celková cena tohoto algoritmu, jež odpovídá součinu jeho časové složitosti a počtu procesorů N, je

$$N\mathcal{O}(n \log n) = n\mathcal{O}(n \log n) = \mathcal{O}(n^2 \log n),$$

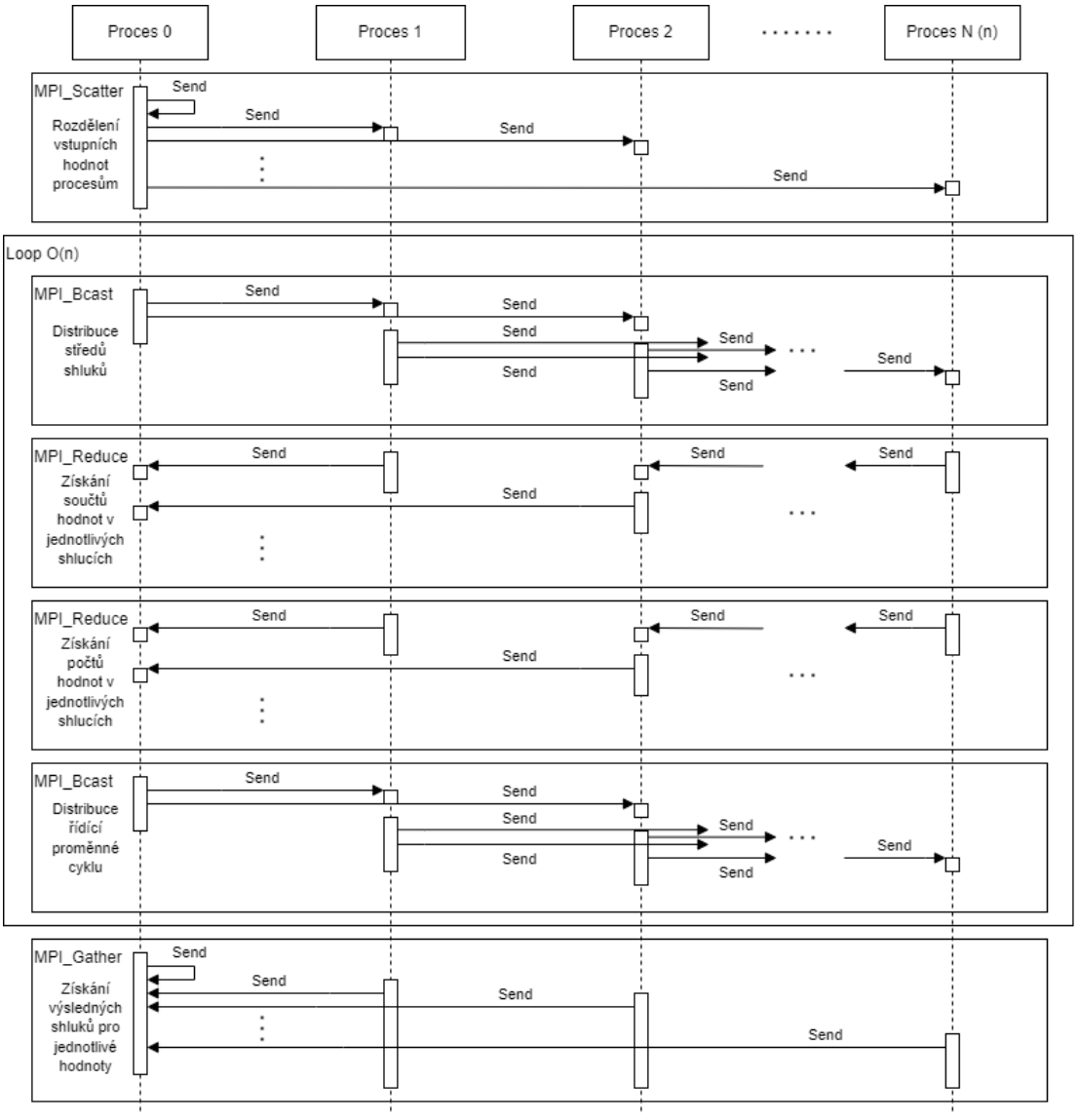
kde n je počet vstupních rozdělovaných hodnot. Tato cena není optimální, jelikož není těžké nahlédnout, že časová složitost sekvenčního řešení tohoto algoritmu je kvadratická ($\mathcal{O}(n^2)$) – k provedení iterace je nutné sekvenčně projít všechny vstupní hodnoty a pro každou v konstantním čase nalézt odpovídající shluk. Zrychlení tohoto algoritmu oproti sekvenčnímu řešení je $\frac{\mathcal{O}(n^2)}{\mathcal{O}(n \log n)} = \frac{\mathcal{O}(n)}{\mathcal{O}(\log n)}$, přičemž je jeho efektivnost rovna $\frac{\mathcal{O}(n^2)}{\mathcal{O}(n^2 \log n)} = \frac{1}{\mathcal{O}(\log n)} < 1$.

Počet procesorů $p(n)$	n
Časová složitost $t(n)$	$\mathcal{O}(n \log n)$
Prostorová složitost $s(n)$	$\mathcal{O}(n)$
Celková cena $c(n)$	$\mathcal{O}(n^2 \log n)$

Tabulka 1: Shrnutí výsledků z rozboru a analýzy implementovaného algoritmu (n představuje počet vstupních hodnot k rozřazení)

Komunikační protokol

Komunikaci mezi jednotlivými běžícími procesy této paralelní implementace lze znázornit sekvenčním diagramem (viz obrázek 1).



Obrázek 1: Sekvenční diagram zobrazující komunikační protokol znázorňující komunikaci mezi jednotlivými běžícími procesy implementovaného algoritmu

2 Závěr

V rámci tohoto projektu bylo v *C++* s využitím *Open MPI* úspěšně implementováno paralelní řešení algoritmu 4-means nad skalárními vstupními hodnotami. Pro tuto implementaci byla dále zanalyzována časová složitost, prostorová složitost a celková cena. Bylo zjištěno, že časová složitost tohoto řešení je linearitmická a prostorová složitost je lineární. Cena tohoto paralelního řešení pak není optimální. Zrychlení oproti sekvenčnímu řešení je rovno $\frac{\mathcal{O}(n)}{\mathcal{O}(\log n)}$, přičemž efektivnost této paralelní implementace odpovídá $\frac{1}{\mathcal{O}(\log n)} < 1$.