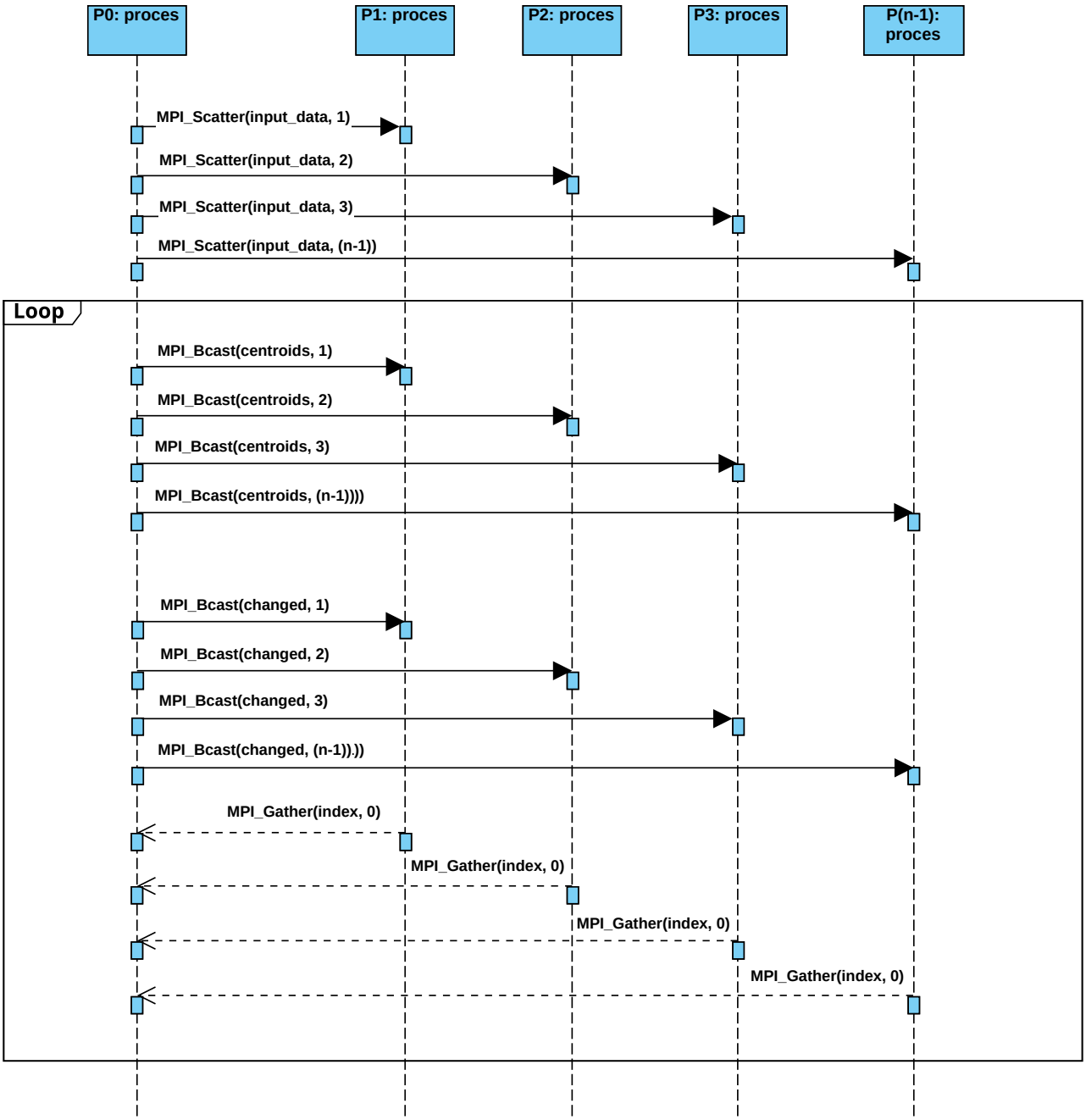
1 Komunikační protokol

Z důvodu omezení na počet stran dokumentace jsem byl nucen umístit komunikační protokol1 na první stránku, v sekcích níže se k němu odkazuji.



Obrázek 1: Sekvenční diagram komunikace procesorů

2 Analýza implementovaného algoritmu

V kódu jsem implementoval následujicí algoritmus 1, který byl poskytnut spolu se zadáním.

Algorithm 1 K-Means 4

```
Zvolte 4 středy (průměry) shluků 1 – 4
repeat
    for každý prvek v datasetu do
        Přiřaď prvek do shluku s nejbližším středem
    end for
    for každý shluk do
        if shluk obsahuje alespoň jeden prvek then
            Spočítej nový střed jako průměr všech prvků v shluku
        else
            Zachovej stávající střed
        end if
    end for
until žádný střed se již nezměnil
Vypiš jednotlivé shluky na obrazovku
```

Diagram komunikace výše 1 znázorňuje zasílání dat mezi jednotlivými procesory. V kódu jsem využil tyto funkce:

`MPI_Scatter(input_data)` Hlavní *P0* procesor nejprve zašle všem, tedy alespoň dalším třem procesorům, data obsahujicí proměnnou typu celé číslo s jednou hodnotou ze vstupního *streamu*[1]. Rozdělení vstupních hodnot mezi procesory závisí na jejich pořadovém čísle tedy *procesor id (rank)*.

Následující funkce jsou využity v cyklu, který je ukončen, když se hodnoty středů shluků nezmění, tedy algoritmus dosáhne konvergence:
`MPI_Bcast(centroids)` Zaslání hodnot středů čtyř shluků, jež jsou při první iteraci inicializovány prvními čtyřmi hodnoty ze vstupního *streamu*.

`MPI_Bcast(change)` Zaslání indikátorové (flag) proměnné *changed*, která signalizuje zda bude probíhat další iterace výpočtu či nikoliv, protože se již středy shluků po přepočítání nezměnily.

Funkce `getClusterIndex()` vypočítá vzdálenost hodnoty od jednotlivých středů shluků jako absolutní rozdíl. Její návratová hodnota je index nejbližšího shluku.

Každý procesor zpracuje svou část dat a vypočítá index shluku, do něhož jemu přidělené číslo patří.
`MPI_Gather(index)` funkce slouží v každém procesoru k zaslání zjištěných indexů zpět do hlavního (rank 0) procesoru.

V hlavním procesoru je vstupní *stream* rozdělen do čtyř shluků na základě přijatých indexů. Následně jsou přepočítány nové hodnoty středů shluků a pakliže se hodnoty liší od původních, tak je zahájena další iterace. V opačném případě je výpočet ukončen a výsledný shluky včetně jejich středů jsou vypsány na obrazovku.

[1] data přečtená na vstupu, například vygenerovaná do souboru numbers

Teoretická složitost algoritmu

Časová složitost:

- `MPI_Scatter(input_data)`, `MPI_Bcast(input_data)`, `MPI_Gather(input_data)`: Funkce mají časovou složitost O(n/p), kde n je počet vstupních čísel a p je počet procesorů, jelikož n=p, tak se dá složitost zjednodušit jako O(1).
- `getClusterIndex()`: Funkce počítá vzdálenost od čtyř středů shluků, tedy její časová složitost je O(4).

Před hlavním cyklem algoritmu je zavolána funkce `MPI_Scatter(input_data)` a dvakrát funkce `MPI_Bcast(change)`, jejich složitost se dá zjednodušit jako O(n).
V hlavním cyklu je v každé iteraci volána funkce `getClusterIndex()`, `MPI_Bcast(change)`, `MPI_Gather(input_data)`, jejich časová složitost se dá zapsat jako O(i(4+n)), kde i je počet vykonaných iterací a n je počet vstupních hodnot.

Celková časová složitost je tedy: O(1)+O(i(4+n)), což se dá zjednodušit jako **O(i*n)**, protože O(1) je zanedbatelně malé, a proto se může vypustit.

Prostorová složitost:

- Pro *input_data*: Potřebujeme velikost n, pro uložení n vstupních hodnot.
- Pro *centroids*: Potřebujeme velikost 4, pro uložení hodnot středů shluků.
- Pro *clusters*: Potřebujeme velikost 4n, pro uložení hodnot, které budou patřit do jednotlivých shluků.
- Pro *index_count*: Potřebujeme velikost 4, pro uložení počtu čísel, které patří do každého shluku.
- Pro *indexis*: Potřebujeme velikost n, pro uložení vypočítaných indexů.

Celková prostorová složitost je tedy: O(n+4+4n+4+n), což se dá zjednodušit jako **O(n)**.

3 Závěr

Spustil jsem kód pro různé počty procesorů a za pomocí vytvořeného jednoduchého shell skriptu, kdy jsem provedl výpočet rozdílu času před spuštěním programu a po něm, jsem změřil jednotlivé doby běhů algoritmu. V tabulce níže 1 jsou zobrazeny výsledné časy. Vzhledem k různým dobám trvání algoritmu pro různé kombinace vstupních hodnot a počtu procesorů, nelze z dat jednoznačně vyvodit, zda se algoritmus vždy chová podle teoretické časové složitosti O(i*n), kde i je počet iterací a n je počet vstupních hodnot. Avšak vzhledem k pozorovanému nárůstu času s rostoucím počtem iterací a vstupních hodnot, se zdá, že se algoritmus může chovat aspoň v některých případech podle této teoretické časové složitosti.
Nekonzistence může pramenit například z neoptimální implementace algoritmu nebo zatížení testovacího serveru. Myslím si, že jsem však zadání splnil a jsem s prací spokojený.

Počet procesorů/čísel	4	8	12	16	20	24
Doba výpočtu(μs)	645	1006	888	944	1115	1023
Počet iterací	1	3	5	2	6	3

Tabulka 1: Délky běhů programu pro rozdílné počty procesorů