1 Rozbor a analýza algoritmu

1.1 Časová složitost

Vzhledem k tomu, že algoritmus *Merge-splitting sort* popsaný ve studijních materiálech nebere v potaz načítání prvků a jejich prvotní distribuci mezi procesory (stejně tak jako distribuci prvků po skončení řazení), nebudu v rozboru analýzy algoritmu tyto části zohledňovat, aby bylo možné algoritmus teoreticky a prakticky porovnat.

Každý procesor seřadí svou posloupnost čísel standardní funkcí `std::sort`, která říká, že od standardu `C++11` je její časová složitost linearitmická, v našem případě tedy

$$O((n \cdot \log n)/p).$$

Samotné paralelní řazení algoritmu probíhá v sekvenčním cyklu do poloviny počtu procesorů, tedy

$$O(\lceil p/2 \rceil).$$

V rámci tohoto cyklu si procesory zasílají své prvky, spojí seřazený seznam a odešlou prvky zpět s časovou složitostí

$$O(n/p) + O(2 \cdot n/p) + O(n/p),$$

což po kombinaci a zkrácení dává složitost

$$O(n).$$

Celková složitost tedy bude součtem

$$O((n \cdot \log n)/p) + O(n),$$

kde k určení složitější části potřebujeme znát počet procesorů. Pokud bude počet procesorů přesně $p = \log n$, složitost bude lineární.

1.2 Prostorová složitost

V průběhu algoritmu si každý procesor musí pamatovat svá čísla, procesory díky tomu spotřebujítí $n/p \cdot p$ paměti. Při přijímání prvků od souseda si ale každý procesor musí alokovat pro tyto prvky paměť, stejně tak jako pro spojené posloupnosti, což značí, že prostorová složitost bude

$$(n/p + n/p + 2 \cdot n/p) \cdot p,$$

tedy po zkrácení $O(n)$ - lineární prostorová složitost.

1.3 Celková cena

Cena algoritmu se spočítá jako časová složitost krát počet procesorů, v případě *Merge-splitting sort* tedy

$$O(n \cdot \log n) + O(n \cdot p).$$

Zda je algoritmus optimální se určí porovnáním s časovou složitostí ideálního řadícího sekvenčního algoritmu. Po takovém porovnání bychom zjistili, že pokud $p \leq \log n$, bude algoritmus optimální.

2 Implementace

Po inicializaci knihovny *MPI* první procesor (rank 0) načte vstupní soubor s čísly a rozešle prvky ostatním procesorům. Dynamické určení, kolik prvků bude který procesor obsluhovat, se dá popsat vzorcem

$$\lfloor n/p \rfloor,$$

kde n = počet ještě nezaslaných prvků a p = počet zbývajících procesorů. Tento přístup řeší případ, kdy počet prvků není dělitelný počtem procesorů a prvky se tak rozloží mezi procesory rovnoměrně.

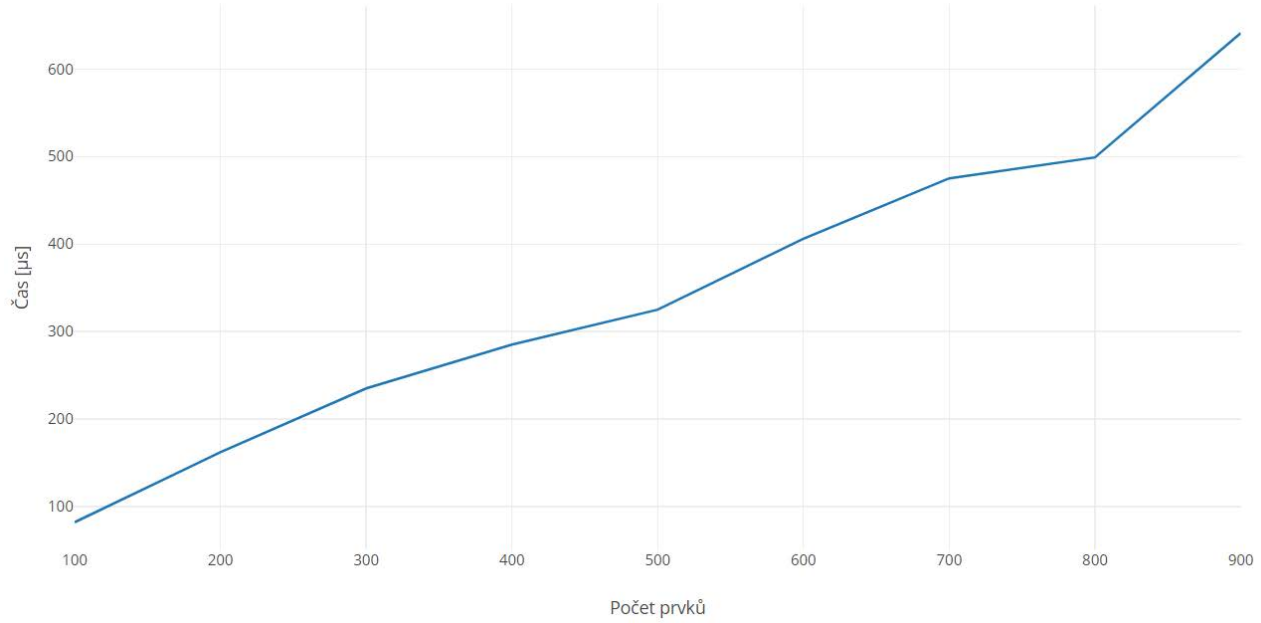
Následně každý procesor seřadí svou posloupnost standardní funkcí `std::sort`. Poté se provede sekvenční cyklus $\lceil p/2 \rceil$krát, kdy v každém cyklu liché procesory zašlou své prvky svému levému sousedu (blokující funkce `MPI_Send` a `MPI_Recv`) - ten prvky spojí a seřadí jednoduchým algoritmem a následně zašle svému sousedu zpět kus tohoto spojeného seznamu (stejný počet hodnot, které procesor původně poslal). To stejné následuje pro sudé procesory.

Po seřazení čeká procesor s rankem 0, až mu ostatní procesory ve správném pořadí zašlou své seřazené prvky tak, aby je mohl vytisknout do zadané podoby na standardní výstup.

3 Experimenty

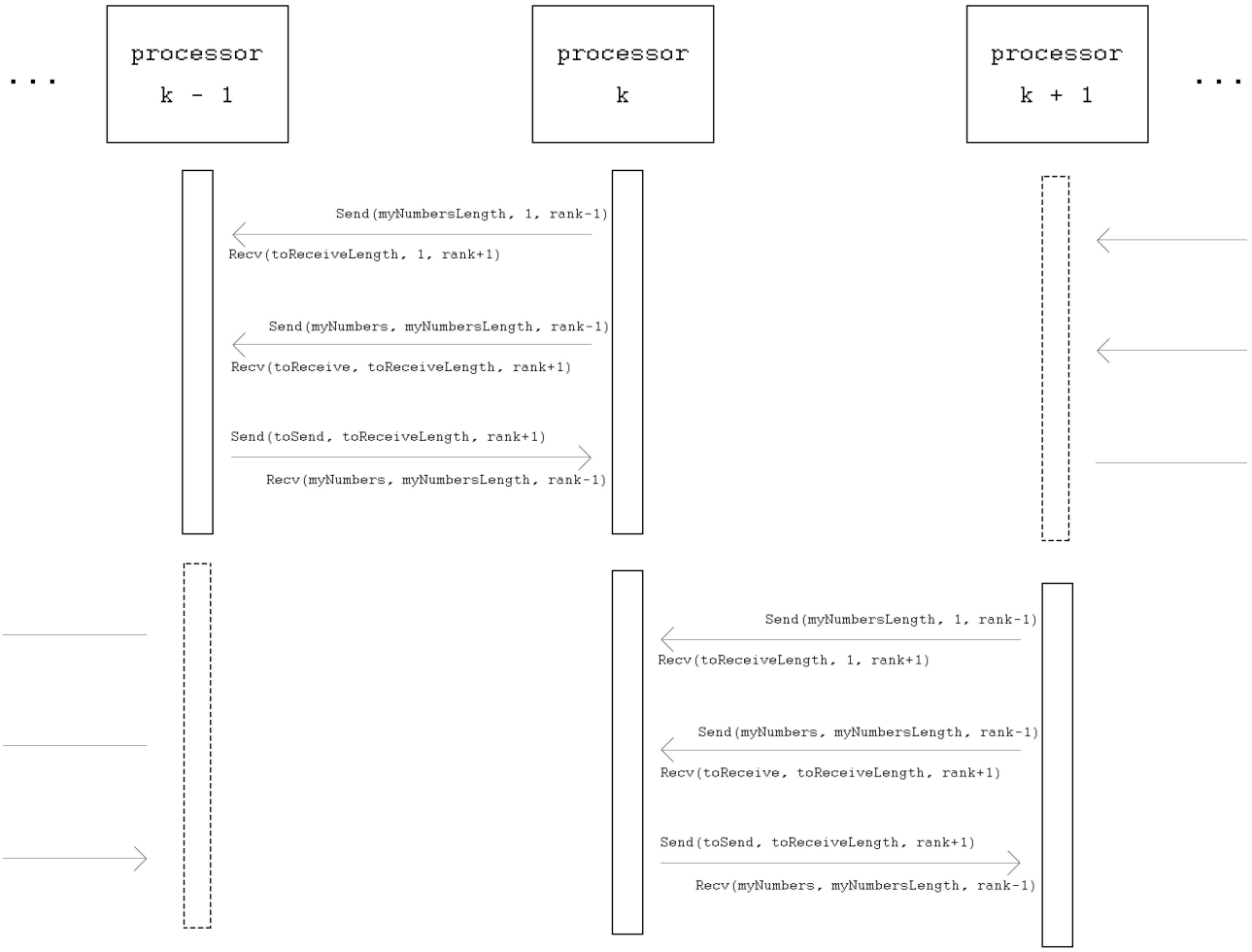
Testování proběhlo na systému *merlin* tak, aby byl algoritmus optimální - počet procesorů byl zvolen jako $p = \log n$, v případě následujícího testu $p = 2$. Čas byl měřen od předzpracování prvků optimálním sekvenčním algoritmem až po ukončení řazení (tak, aby složitost odpovídala teoreticky popsané složitosti výše), přičemž pro každý počet prvků bylo provedeno pět měření, která byla do výsledného grafu zprůměrována. K měření času byla využita knihovna `chrono` a konkrétně struktura `chrono::time_point`, kdy byly zaznamenány dva časové body a proveden jejich rozdíl.

Díky zvolenému počtu procesorů by měla být časová složitost lineární - výsledný graf tomu v podstatě odpovídá.



4 Komunikační protokol

Komunikační protokol mezi procesory znázorňuje níže přiložený sekvenční diagram, který ukazuje obecnou komunikaci mezi procesory v průběhu řazení (kde $0 < k < n$) - pokud má procesor zasílat své prvky, jako první pošle sousednímu procesoru počet těchto prvků a následně samotné prvky. Dále čeká na navrácení své části seřazené spojené posloupnosti.



5 Závěr

Pokud pomineme již zmíněnou prvotní a finální distribuci prvků mezi procesory, dá se říct, že implementovaný algoritmus odpovídá očekávanému předpokladu (při počtu procesorů $p = \log n$), tedy lineární časové složitosti. Nedokonalost přímky mohlo způsobit kupříkladu nerovnoměrné zátěže serveru *merlin* studenty.