Odd even transposition sort

1. Teoretická analýza algoritmu

Algoritmus pre *odd even transposition sort* by mal mať časovú zložitosť **O(n)**. Podľa prednášky h003.pdf strana 12, kde je časť algoritmu, ktorý som implementoval. Popíšme si zložitosť algoritmu v nasledujúcich odrážkach:

- V jedno kroku procesor spraví jedno porovnanie a dva prenosy – **O(c)**
- Jeden procesor vykoná až n porovnaní - **O(n)**
- Všetky procesory **– O(n)**

Výsledná časová zložitosť **je t(n) = O(n)** a priestorová zložitosť **c(n) = O(n^2).**

Cena optimálneho algoritmu pre zoradenie je **c(n) = n.log(n)**. Cena nášho algoritmu je **c(n) = n^2**. Optimálny algoritmus potrebuje **O(log(n))** času a **O(n)** priestoru. *Odd even transposition sort* potrebuje **O(n)** času a **O(n)** priestoru, čo nie je optimálne voči najlepšiemu algoritmu pre zoradenie.

2. Implementácia algoritmu

Algoritmus je implementovaný pomocou jazyka *C++* a knižnice *openMPI*. V nasledujúcich kapitolách si popíšeme implementáciu algoritmu.

2.1. Načítanie čísiel zo súboru

V programe sa čítajú jednotlivé čísla zo súboru, ktorý je otvorený pre binárne čítanie. Na jednotlivé čítanie čísla používam cyklus *while*, ktorý sa vykoná iba v koreňovom procesore. V cykle *while* načítam číslo do *readValue* a následne hodnotu v premennej pošlem do odpovedajúceho procesora pomocou funkcie *MPI_Send*.

2.2. Porovnanie hodnôt

Každý procesor príjme jedno číslo pomocou *MPI_Receive*. Potom sa vypočíta posledné číslo pre párny a nepárny procesor a do premennej *halfCycles* polovica procesorov zo zaokrúhlením nahor. Potom v cykle *for*, ktorý sa vykoná maximálne do hodnoty určenou halCycles.

2.3. Výpočet kroku v cykle *for*

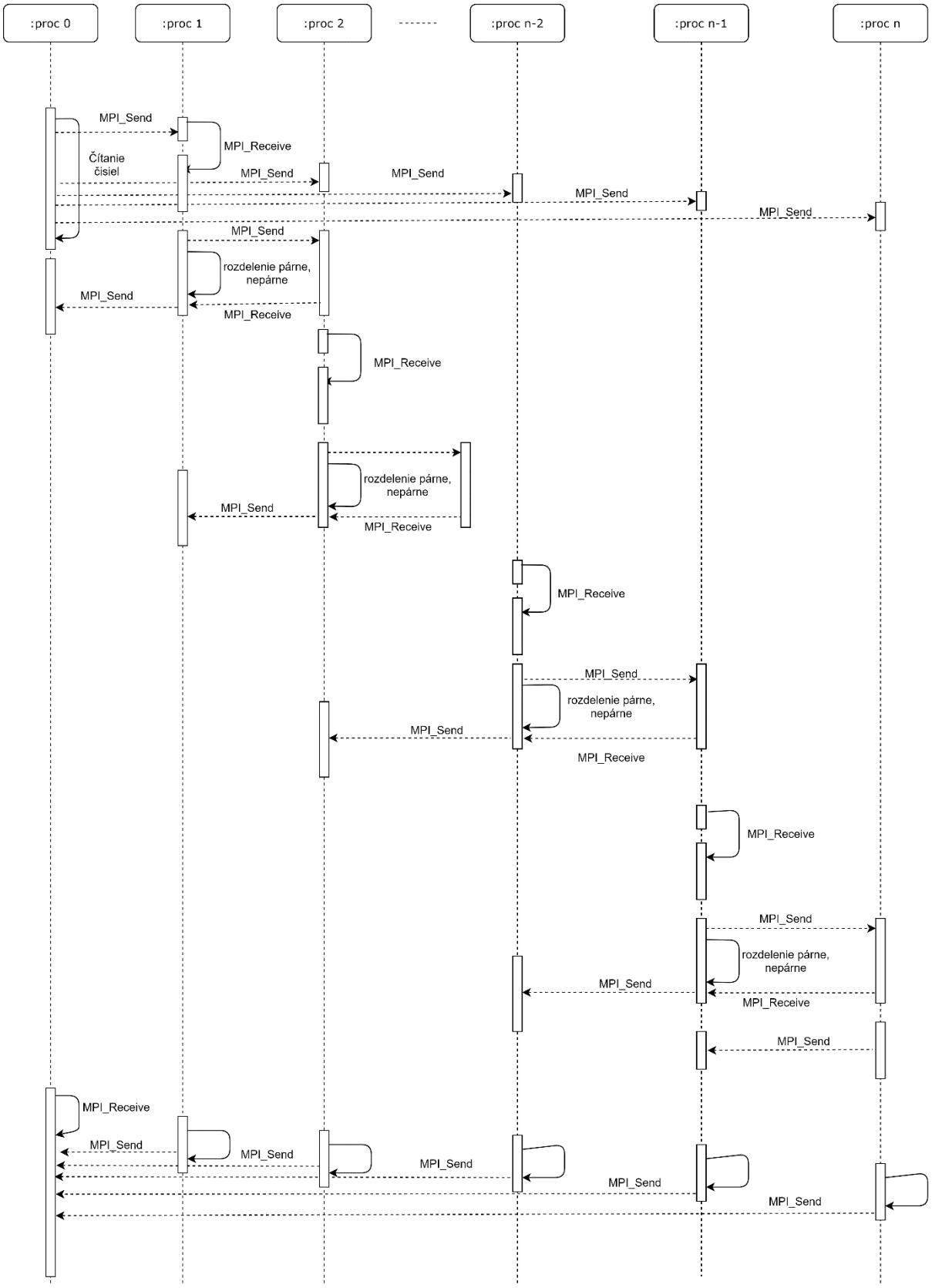
Pre jeden krok sa skontroluje, či je párny alebo nepárny alebo či to nie je posledný párny alebo nepárny.

Ak je procesor koncový párny alebo nepárny tak príjme hodnotu a pošle ju susedovi. Inak robí porovnanie a pošle hodnotu podľa porovnania.

2.4. Zoradené hodnoty

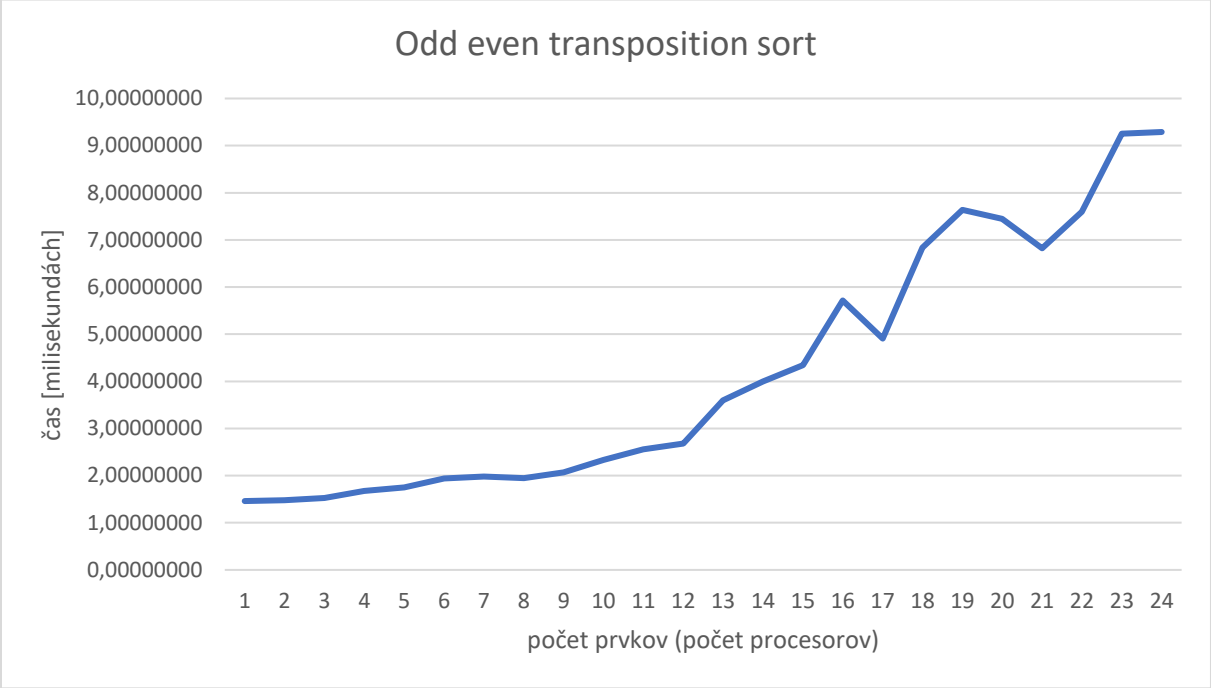
Na konci každého výpočtu sa posielajú hodnoty do koreňového uzlu, kde sa zozbierajú a nakoniec sa vypíšu.

3. Komunikačný protokol



4. Experimenty

S implementáciou som robil experimenty. Spravil som *bash* skript, ktorý spustil program 6000 krát, teda 250 krát od 1 do 24 náhodných jedno bajtových čísiel. Graf č.1 zobrazuje tieto hodnoty v milisekundách a je to priemer 250 meraní.



Graf č.1

5. Záver

Z grafu č.1 je vidieť, že algoritmus má lineárnu časovú zložitosť. Skoky, ktoré je vidieť na grafe môžu byť spôsobené rôznou vyťaženosťou procesorov na serveri *merlin*. Jemné odchýlky môžu byť spôsobené vygenerovanými hodnotami, niektoré vstupy mohli byť úplne zoradené alebo úplne nezoradené alebo čiastočne zoradené hodnoty.