Odd-even transposition sort

Peter Grofčík
xgrofc00

March 27, 2020

1 Úvod

Cieľom tohto projektu je naimplementovať paralelný radiaci algoritmus *Odd-even transposition sort*. Projekt tiež obsahuje skript *test.sh*, ktorý vytvorí postupnosť náhodných čísel v súbore numbers a po jeho skončení daný súbor vymaže.

2 Princíp algoritmu

Princíp algoritmu spočíva v radení prvkov pomocou porovnávania a výmeny hodnôt medzi dvoma susediacimi procesormi. Každý z procesorov tak oplýva práve jednou hodnotou a k výmene čísiel medzi nimi dochádza v dvoch krokoch. V prvom kroku každý nepárny procesor "kontaktuje" po ňom idúceho suseda a porovná si s ním veľkosť svojho čísla. V prípade, že je jeho číslo väčšie ako číslo jeho suseda tak dôjde k ich výmene. V druhom kroku Vykonajú rovnakú akciu všetky párne procesory. Tieto kroky sa opakujú v cykle *n*-krát, kde *n* je hodnota odpovedajúca *(počtu procesorov)/2*. Hodnoty sú tak zoradené maximálne po *n*2* krokoch, čo odpovedá počtu procesorov a zároveň algoritmus počíta s rovnakým počtom nezoradených čísel ako je počet procesorov použitých na ich zoradenie.

3 Analýza algoritmu

1. Master procesor rozpošle každému procesoru číslo, čo sa dá vykonať v konštantnom čase, a teda zložitosť je možné zapísať ako *O(1)*.

2. Každý procesor porovnáva číslo v cykle n/2 krát (prípadne n/2+1 krát pri nepárnom počte). Každá iterácia je tvorená konštantným počtom krokov z čoho dostávame (n/2)*c (resp, (n/2-1)*c). Z toho môžme výslednú zložitosť zapísať ako *O(n)*.

3. Každý procesor odosiela svoje číslo Masterovi, ktorý ich v cykle ktorý vykonáva konštantný počet krokov ukladá. Zložitosť je teda *O(n)*.

Zistená časová zložitosť je teda *O(n)*. Počet procesorov potrebných na algoritmus je rovný počtu radených prvkov, a teda *p(n=n)*. Z toho môžme odvodiť cenu paralelného riešenia, ktorá je teda $c(n) = p(n) * t(n) = O(n^2)$, čo nie je optimálne keďže optimálny sekvenčný algoritmus, založený na porovnávaní prvkov má cenu $c(n) = O(n * logn)$.

4 Implementácia

Algoritmus je implementovaný v jazyku **C++** za pomoci knižnice **Open MPI**. Na začiatku dôjde k inicializácii MPI prostredia pomocou funkcie MPI_Init. Následne každý z procesorov načíta svoje osobné číslo **ID** a celkový počet procesorov **CPU** pre výpočet počtu cyklov a indexovanie preposielaných správ spojim susedom. Následne dôjde k načítaniu dát zo súboru *numbers* a ich uloženie do jednorozmerného poľa, ktoré je vykonané iba jedným procesorom *Masterom*. Uloženie dát do poľa má, z hľadiska experimentov, za účel oddelenie načítania dát zo súboru a samotný výpočet algoritmu pre presnejšie časové výsledky experimentov.

V samotnej časti algoritmu tak dôjde k rozoslaniu čísel *Masterom* všetkým ostatným. Následne tak v cykle komunikujú jednotlivé procesory pomocou funkcii *MPI_Send* a *MPI_Recv*. Počet cyklov je algoritmom implicitne daný ako polovica počtu vstupných čísel (resp. procesorov), avšak v prípade nepárneho počtu by toto číslo nepredstavovalo reálne číslo. Práve preto bolo potrebné docieliť zaokrúhlenie počtu cyklov nahor (resp. pridať cyklus navyše), pretože by v extrémnom prípade kedy je najmenšie číslo práve na poslednej pozícii nedošlo k jeho zaradeniu na prvú pozíciu.

Poslednú fázu tvorí už len prijatie jednotlivých už zoradených čísel Masterom v cykle od indexu 1. Každý procesor tak odošle svoje číslo teraz už staticky nultému procesoru, pričom najmenšie číslo je práve to, ktoré patrí nultému procesoru.

5 Komunikačný protokol

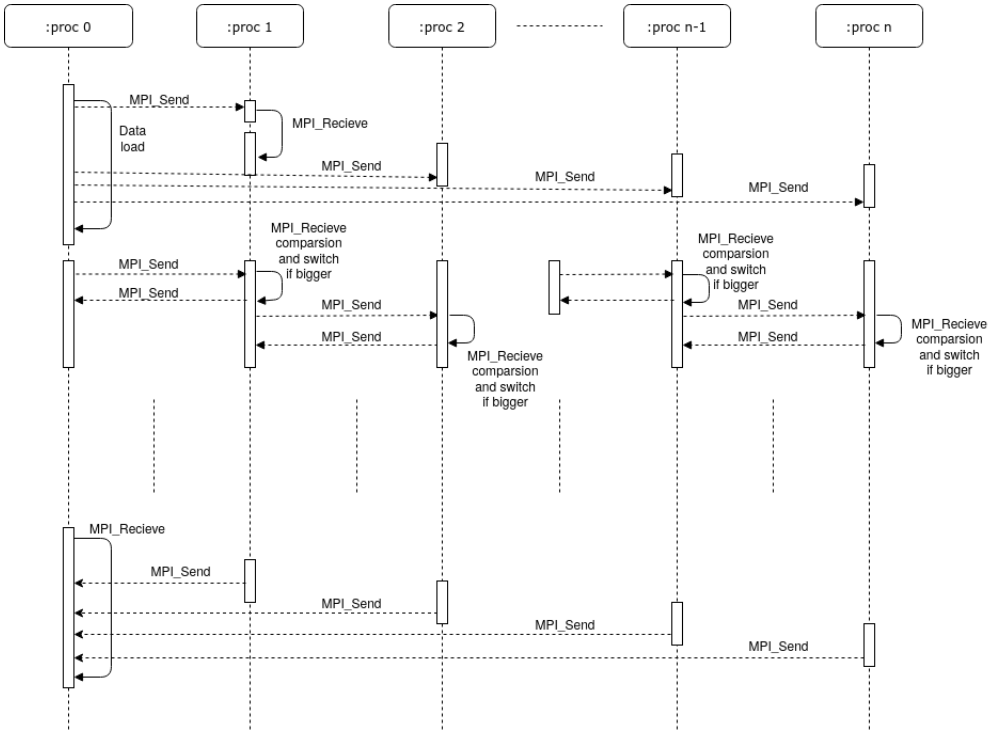

Figure 1: Sekvenčný graf komunikácie procesorov

6 Experimenty

Cieľom experimentov bolo overiť časovú zložitosť algoritmu. Použitím funkcie **MPI_Wtime** na vhodných miestach implementácie (tj. po načítaní čísel a pred výpisom zoradenej postupnosti) som bol schopný zistiť čas pred a po spustení algoritmu. Zistené časi som od seba odčítal a následne previedol na mikrosekundy z dôvodu zaokrúhľujúcich výsledkov blížiacich sa nule. Školský server Merlin bol schopný poskytnúť až 27 voľných procesorov pre experimenty, čo znamenalo 27 možných meraní pre rôzny počet nezoradených čísel. Pre čo najpresnejší výsledok som pre každý rozdielny počet nezoradených čísel previedol 250 meraní a získané hodnoty následne spriemeroval. Merania je možné spustiť zámenou vstupného čísla za argument **"EXP"** a ich výstupom je súbor **measure.txt** obsahujúci výsledky z jednotlivých meraní, na základe ktorých som vytvoril nasledovný graf.

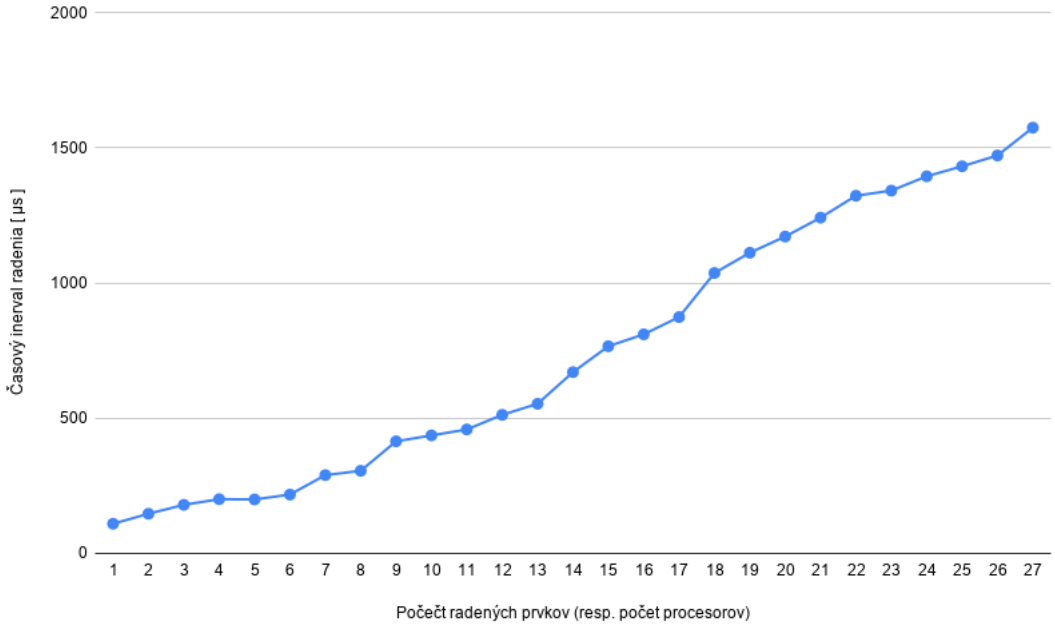

Figure 2: Meranie času experimentov

7 Záver

Výsledky meraní vyobrazené v grafe odpovedajú tvrdeniu o lineárnom raste času potrebného na radenie prvkov s ich rastúcim počtom, čo potvrdzuje zistenú lineárnu zložitosť algoritmu. Mierne výkyvy pri tak nízkych časových hodnotách možno pripísať zaťaženiu školského servera.