2. projekt v PRL 2018/2019: Paralelný algoritmus na výpočet úrovní vrcholov
Meno a priezvisko: Dávid Bolvanský
Login: xbolva00

1 Úvod

Cieľom tohto projektu je implementovať paralelný algoritmus na výpočet úrovní vrcholov. Súčasťou je aj shell skript, ktorý vypočíta potrebný počet procesorov pre vstupný reťazec, ktorý reprezentuje vstupný binárny strom.

2 Popis algoritmu

Algoritmus paralelne počíta úrovne vrcholov. Počet potrebných procesorov odpovedá počtu hrán (n), tj. $p(n) = n$. V prípade binárneho stromu je počet hrán definovaný vzťahom $n = 2 * pocet\_uzlov - 2$. Každý procesor spracováva jednu z dopredných alebo spätných hrán v binárnom strome. Procesor pomocou zoznamu susedných uzlov vypočíta následníka pre svoju hranu. Hrana smerujúca z pravého syna koreňa do koreňa má za následníka samu seba. Následne procesor pre svoju hranu zistí jej váhu - ak ide o doprednú hranu, váha je -1, inak 1. Nové váhy sa získajú aplikovaním algoritmu pre výpočet sumy suffixov, kde sa za pole následníkov berie získaná eulerova cesta (následníci hrán) a za pole hodnôt prvotné váhy hrán. Úroveň uzla sa získava z novej váhy, okrem prípadu, kde ak je uzol koreň, úroveň sa nastaví na 0. Ak je hrana dopredná, novú váhu inkrementujeme a táto hodnota je úroveň uzla, do ktorého hrana smeruje.

3 Analýza algoritmu

1. Výpočet eulerovej cesty. Zložitosť tohto kroku je konštantná, $\mathcal{O}(c)$.
2. Prvotná inicializácia váhy hrany podľa jej typu. Zložitosť tohto kroku je tiež konštantná, $\mathcal{O}(c)$.
3. Výpočet sumy sufixov. Zložitosť použitého algoritmu pre sumu suffixov je $\mathcal{O}(log_2(n))$.
4. Korekcia výslednej hodnoty (váhy). Zložitosť tohto kroku je konštantná, $\mathcal{O}(c)$.

Zistená časová zložitosť je $t(n) = \mathcal{O}(log_2(n))$. Potrebný počet procesorov je $p(n) = \mathcal{O}(n)$. Cena paralelného riešenia je všeobecne definovaná ako $c(n) = t(n) \cdot p(n)$, v našom prípade cena paralelného výpočtu úrovní vrcholov je $c(n) = \mathcal{O}(n \cdot log(n))$. Algoritmus má optimálnu cenou v prípade, že platí $c(n)_{optim} = t_{seq(n)}$. Keďže časová zložitosť optimálneho sekvenčného algoritmu na výpočet úrovní vrcholov je $\mathcal{O}(n)$ a cena jeho paralelnej verzie je $\mathcal{O}(n \cdot log(n))$, paralelný algoritmus na výpočet úrovní vrcholov nie je optimálny.

4 Implementácia

Algoritmus je implementovaný za pomoci knižnice Open MPI. Jedná sa kód v jazyku C++. Reťazec, ktorý reprezentuje strom, je získaný prečítaním prvého argumentu programu. Počet uzlov je dĺžka tohto reťazca. Na samotnom začiatku sa spustí funkcia `MPI_Init` na inicializáciu MPI prostredia. Prvým krom algoritmu je výpočet eulerovej cesty. Keďže pracujeme s binárnym stromom, vieme, že susedia uzlu sú jeho synovia a jeho rodič - preto v implementácii sa nezostavuje zoznam susednosti, ale pre každú hranu je možné priamo určiť jej následníka. Každý procesor pošle svoj rank zistenému následníkovi a čaká na prijatie ranku od svojho predchodcu. Hrana smerujúca do koreňa nemá svojho následníka, následník tejto hrany sa nastaví na -1. Hodnota -1 slúži ako zarážka, ktorá sa neskôr využije v implementácii algoritmu pre výpočet sumy sufixov. Obdobne, hrana z koreňa nemá svojho predchodcu, a preto sa predchodca nastaví na -1. Po získaní eulerovej cesty sa pre každú hranu zistí jej prvotná váha. Ak ide o doprednú hranu, váha je -1, inak 1. Implementačne sa typ hrany určuje podľa ranku procesora. Ak je rank v hornej polovici rozsahu rankov procesorov, procesor spracováva spätnú hranu. Inak spracováva doprednú hranu. Následne sa vykoná algoritmus pre výpočet sumy suffixov nad zistenými následníkmi, za vstupné hodnoty do algoritmu sa berú prvotné váhy. Keďže vstupné pole hodnôt do sumy prefixov sú prvotné váhy, ktoré nadobúdajú hodnôt -1 a 1 (platí, že posledná hodnota teda v našom prípade nie je 0), je potrebné urobiť korekciu výsledných váh, tj. pričítať hodnotu poslednej váhy. V $log_2(n)$ iteráciách, presnejšie v $ceil(log_2(n))$ (počet uzlov môže byť rôzny, nie len mocnina 2), prebieha jadro algoritmu pre výpočet sumy suffixov. Ak existuje následník, pošlú sa mu dva ranky - rank aktuálneho predchodcu a rank procesora. Ďalej, ak existuje predchodca, čaká sa na prijatie dvoch rankov od neho. Prvá hodnota je rank predchodcu môjho predchodcu a za nového aktuálneho predchodcu sa nastaví práve táto hodnota. Druhá prijatá hodnota

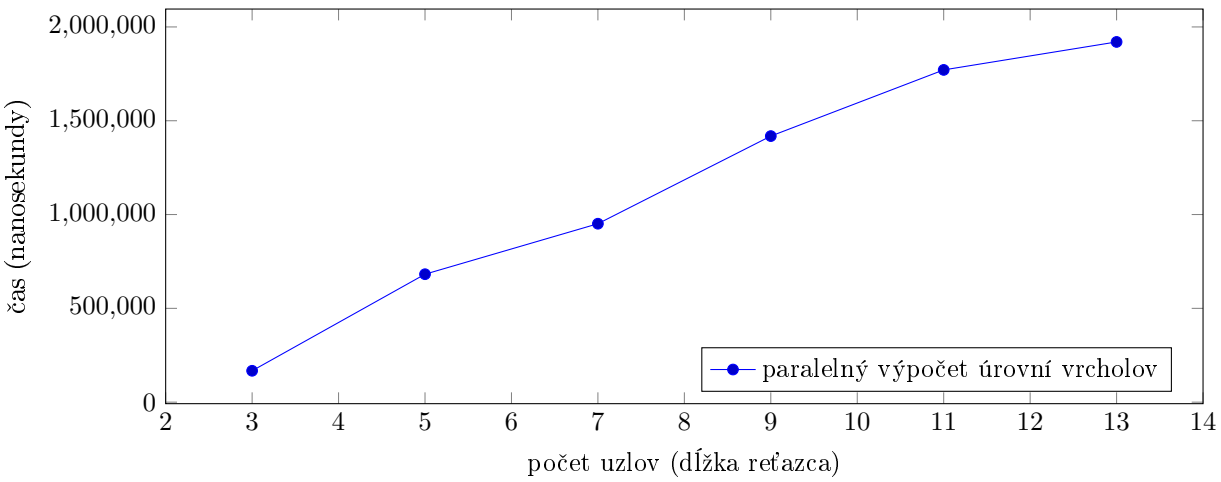
je rank predchodcu, ktorému sa pošlú tiež dve hodnoty - rank aktuálneho následníka a váha hrany procesora. Následne, ak existuje následník, čaká sa na prijatie dvoch čísiel od neho. Za nového následníka sa nastaví prvá hodnota, čo je rank identifikujúci procesor, ktorý je následník následníka. K aktuálnej váhe sa pričíta druhá hodnota, tj. váha následníka. Po skončení algoritmu pre výpočet sumy sufixov je pre získanie správnych úrovní previesť korekciu. Ak procesor spracováva doprednú hranu, novú váhu získanú algoritmom pre výpočet sumy sufixov inkrementuje, a koreňovému uzlu pošle dve hodnoty - inkrementovanú váhu (= úroveň) a číslo uzla, do ktorého hrana smeruje. Koreňový proces prijíma získané dvojice, a do pola úrovní si pre každý uzol ukladá jeho úroveň. Pre koreňový uzol nastaví úroveň na 0. Následne vypíše na výstup pre každý uzol jeho vypočítanú úroveň a spustí sa funkcia `MPI_Finalize` pre ukončenie MPI prostredia.

Implementáciu som priebežne testoval na svojom systéme Ubuntu 18.04 LTS a na školskom serveri *merlin* s CentOS 7.6. Na otestovanie implementácie a shell skriptu na výpočet procesorov som si napísal testovací skript, ktorý porovnával výstupy mojej implementácie paralelného algoritmu na výpočet úrovní vrcholov s výstupom referenčného programu na sekvenčný výpočet úrovne vrcholov. Tento testovací skript je dostupný na `https://github.com/davidbolvansky/PRL2-Tree-Node-Levels-Tester`.

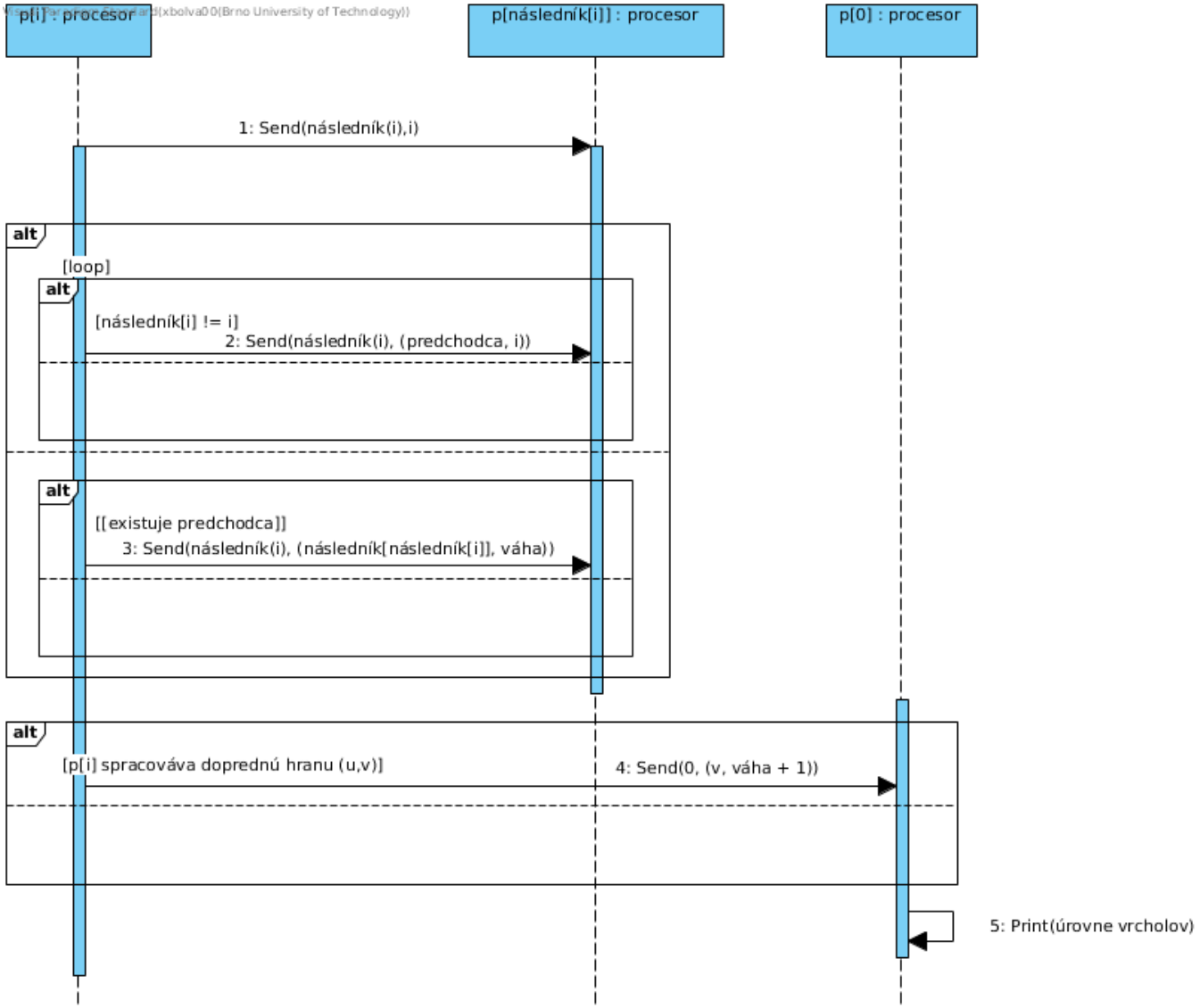
5 Experimenty

Cieľom experimentov bolo overiť časovú zložitosť paralelného algoritmu na výpočet úrovní vrcholov. Pre účely zistenia časovej zložitosti bolo potrebné eliminovať rušivé vplyvy, ktoré nesúvisia so samotným algoritmom. Vhodným vložením meracích miest do implementácie bolo možné správne zistiť časovú zložitosť - do výsledných časov sa nerátajú činnosti ako sú napr. načítanie reťazca, výpis výsledných úrovní vrcholov, a pod. Pomocou funkcie `MPI_Wtime` som si zistil aktuálny čas pred spustením algoritmu (tj. pred výpočtom eulerovej cesty) a čas po skončení algoritmu (tj. pred výpisom výsledných úrovní vrcholov). Tieto zistené časy som od seba odčítal pre zistenie doby behu samotného algoritmu. Rozdiel časov som následne zo sekúnd previedol na nanosekundy. Pre každý zvolený počet čísiel n som vykonal 13 meraní na školskom serveri *merlin* - prvotný výsledok so šumom som zahodil a následne som taktiež zahodil najlepší a najhorší výsledok. Zo zostávajúcich 10 meraní sa spriemerovaním dosiahla finálna reprezentatívna hodnota pre daný počet čísiel. Režim merania doby behu algoritmu je možné zapnúť definovaním makra `MEASURE_TIME`.

uzly	3	5	7	9	11	13
ns	167185	681987	951113	1418190	1770973	1920120



6 Komunikačný protokol



Obr. 1: Sekvenčný diagram popisujúci komunikáciu n procesov u paralelného výpočtu úrovní vrcholov

7 Vyhodnotenie experimentov a záver

Výsledky meraní a zhotovený graf v kapitole 5 poukazujú na fakt, že by sa malo jednať o logaritmickú časovú zložitosť. Toto zistenie podložené vykonanými experimentami je v súlade s odvodenou teoretickou časovou zložitosťou v kapitole 3. Mierne výkyvy pri meraní možno pripísať rôznemu zaťaženiu školského servera v čase, kedy prebiehali experimenty.

8 Informačné zdroje

1. *Algoritmy nad seznamy, stromy a grafy* (prednáška v predmete PRL)
   `https://www.fit.vutbr.cz/study/courses/PDA/private/www/h007.pdf`
2. *Introduction to Parallel Connectivity, List Ranking, and Euler Tour Techniques*
   `https://www.fit.vutbr.cz/study/courses/PDA/private/Reif02_SaraBaase.pdf`