3. projekt - Výpočet úrovne vrcholu

Bc. Michal Ormoš
Paralelné a distribuované algoritmy (PRL) 2018/2019

26. apríla 2019

Abstrakt

Dokumentácia k tretiemu projektu do predmetu Praralelné a distribuované algoritmy (PRL). Obsahuje popis zadania, rozbor a analýzu algoritmu *Výpočet úrovne vrcholu*, jeho implementáciu pomocou OpenMPI, zhrnutie časovej a pamäťovej náročnosti, výsledky experimentov.

1 Analýza algoritmu

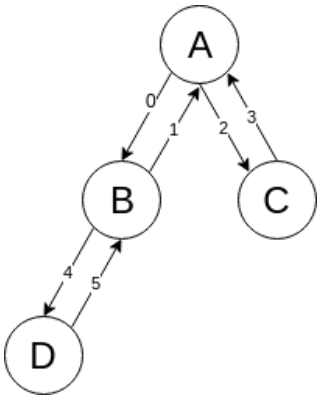
1.1 Teoretický rozbor

Algoritmus *Výpočet úrovne vrcholu*, ktorý patrí do skupiny algoritmov pre prácu nad stromom, bol spracovaný tak, ako bol uvedený na prednáškach [1], vykonáva výpočet vrcholov stromu s n procesmi. Strom obsahuje m uzlov, z toho na ich prepojenie potrebujeme $m-1$ hrán, avšak každú hranu potrebujeme zdvojiť (pre hranu ktorá ide smerom ku koreňu, nazvime ju spätná hrana zvolíme konštantu -1 a pre hranu ktorá ide smerom od koreňa, nazvime ju dopredná hrana, zvolíme hodnotu 1). Teda budeme potrebovať $n = 2 * m - 2$ procesov.

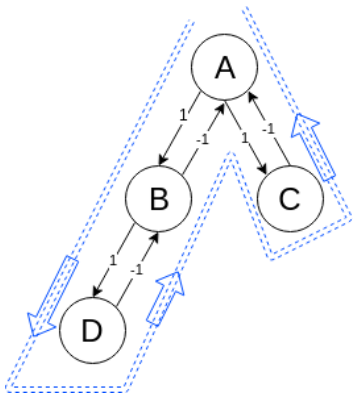
1.2 Rozbor zložitosti

- Časová zložitosť
  - Etour $O(c)$
  - Inicializácia weight $O(c)$
  - Výpočet SuffixSums $O(log * n)$
  - Korekcie výsledku $O(c)$
  - $\sum$ = **t(n) = O(log*n)**
- Pamäťová zložitosť: **p(n) = O(2*n-2)**
- Cena závisí na implementácii SuffixSums, predpokladajme, že by sa nám mohla podariť ako $O(log * n)$, teda **c(n)**= $t(n) * p(n)$ =**O((2*n-1)*(log*n))**

[1] https://www.fit.vutbr.cz/study/courses/PDA/private/www/h007.pdf



(a) Príklad rozdelenia procesov pre strom s 4 uzlami.



(b) Eulerova cesta stromom

Obr. 1: Grafické znázornenie stromu v implementácií

2 Implementácia

Príprava behu (demonštrujme na príklade):

- Majme 4 uzly, ktorým chceme prideliť váhu ich vrcholov. Vstupom je reťazec, ktorý reprezentuje hodnoty uzlov v binárnom strome. Hlavný koreňový uzol, označme si ho ako A, bude mať úroveň vrcholu 0, jeho synovia B a C, zľava doprava úroveň 1 a syn vrcholu B, teda D, úroveň 2. Počet procesov potrebných na paralelný výpočet bude počet hrán, ktoré vzniknú medzi uzlami a to pre každu doprednú a spätnú hranu, teda $2 * m - 2$. Pre náš príklad štyroch uzlov to činí 6 procesov. Grafický načrt vyzerá nasledovne, obrázok 1a.

Implementácia paralelného výpočtu vrcholov stromu v C++:

- Vieme počet procesov, môžme teda úspešne spustiť OpenMPI, s počtom procesov 6. Každý procesor sa bude starať o jednu z dvoch hrán, ktoré spájajú dva uzly. Spôsob akým program jednotlivé procesory na hrany priraďuje je demonštrovný na obrázku 1a. Spracovanie vstupu prebehne v každom procese osobite. Každý jeden proces si načita vstup, zistí koľko procesov v danej inštancií beži a zistí si `id` svojho procesu.

  Následne pomocou týchto informácii každý proces individuálne vypočíta vzťahy pre svoju hranu. Kto je jeho následník, predchodca, `id` procesu hrany k hrane jemu inverznej a akú ma váhu hrana samotného procesu (1 alebo -1). Váhu si vypočíta podľa toho či je hrana dopredná alebo spätná. Ak je dopredná tak ma váhu 1 ak je spätná, tak -1. Prečo práve tieto konštanty bude ozrejmené neskôr. Pomocou týchto informácií si následne môže vypočítať zoznam súsedností (adjacency list), kde sa uložia tieto informácie a proces si týmto može vytvoriť predstavu kde v strome sa nachádza.

  Dalším krokom je hľadanie Eulerovej cesty, kde si každý proces do pripraveného poľa Eulerovej cesty uloží pod indexom svojho `id` procesoru hranu, ktorú považuje za svojho následníka. Eulerovska cesta avšak nejde od uzlu až ku koreňu priamou cestou, ale robí rôzne zákutia 1b, kde v tých zákutiach ide vždy nahoru a dolu. V tom, čo tam pre nás robí naviac mi môžeme navzájom vyrušiť (smer kedy ide dolu a nahoru) a zostanú nám len tie hrany, ktoré sa s ničím nevyrušia a to je ta najkratšia cestu ku koreňu. Na toto vykrátenie nám poslúži fakt, že sme si dopredné hrany označili ako 1 a spätné hrany ako -1.

  Každý jeden procesor túto informáciu distribuuje pomocou `MPI_Allgather` medzi všetky ostatné procesy. `MPI_Allgather` slúži v svojej podstate aj ako `MPI_Barrier`, čo znamená, že

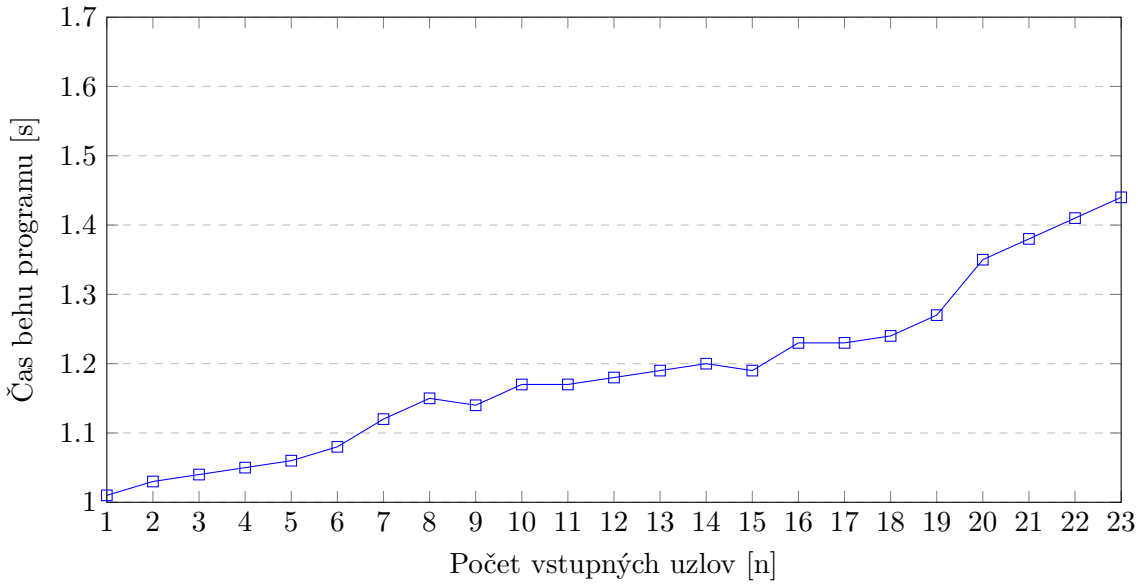
každý proces čaká až všetky procesy dorazia do tohto bodu programu, aby mohol pokračovať. V tomto bode vieme, že každý proces vlastní rovnaké pole, kde je celá Eulerova cesta stromom.

Predposledným krokom je spočitanie suffix sum pre danú Eulerovu cestu na ktorom sa podieľajú všetky procesy okrem procesu posledného. Každý proces, okrem procesu posledného, pošle pomocou už dobre známeho `MPI_Send` hodnotu svojho `id` nasledujúcemu procesu. Potom každý prcoes, okrem prvého, pomocou `MPI_Recv` príjme hodnotu svojho predchodcu. Posledný proces Eulerovej cesty ukazuje sám na seba, neutrálny prvok.
Vypočet summy suffixov prebieha v cykle. Proces pošle svoju hodnotu váhy svojemu prechodcovi, ďalej mu rovnako pošle aj hodnotu svojho následníka. Na konci pošle svojemu nasledovníkovi hodnotu svojho predchodcu. V ďalšej fáze prijme procesor 3 hodnotym váhu svojho nasledovníka, odkaz na následníka následníka a odkaz na predchodcu predchodcu. Krajné procesy ošetrujeme a tie svoju hodnotu neposielajú.

V poslednom kroku už len v našom prípade už štandardne nultý proces príjme suffix sum od každého procesoru, vykoná korekciu výsledku a ten vypíše na štandardný výstup.

3 Experimenty

Časová zložitosť bola nameraná pre počet uzlov 1 až počet ktorý dovolil testovacie zariadenie, v našom prípade 23. Pre každé čislo bolo vykonaných 40 meraní, kde krajných 5 hodnôt bolo odstránených a zvyšných 30 bolo šatisticky vhodne upravených a spriemerovaných. Tento priemer je nanesený v grafe. Experimenty boli vykonané na VUT FIT servery `Merlin` pomocout jednoduchej shell funkcie time. Narozdiel od prvého projektu kde sme merali špecifickú časť algoritmu a to tú kde algoritmus prvky radí, v tomto projekte som sa rozhodol merať beh program ako celok, kedže v našom prípade každá časť daného programu hraje vo výsledku rolu. Výsledky reprezentované graficky:



4 Záver

V rámci tohto projektu sa podarilo úspešne implementovať paralelný algoritmus na výpočet úrovne vrcholu. Výsledkom je funkčný paralelný program schopný zistiť úrovne vrcholu binárneho stromu do maximálnej dĺžky 23 prvkov (tento limit je určený serverom Merlin, ktorý nám viac procesorov neposkytol). Uskutočnené merania ukázali, že reálna časová zložitosť programu je skutočne približne logaritmická (ako sa očakávalo).

A Sekvenčný diagram

Obr. 2: Sekvenčný diagram programu