3. projekt - Výpočet úrovne vrcholu

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

11. apríla 2019

Abstrakt

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

1 Analýza algoritmu

1.1 Teoretický rozbor

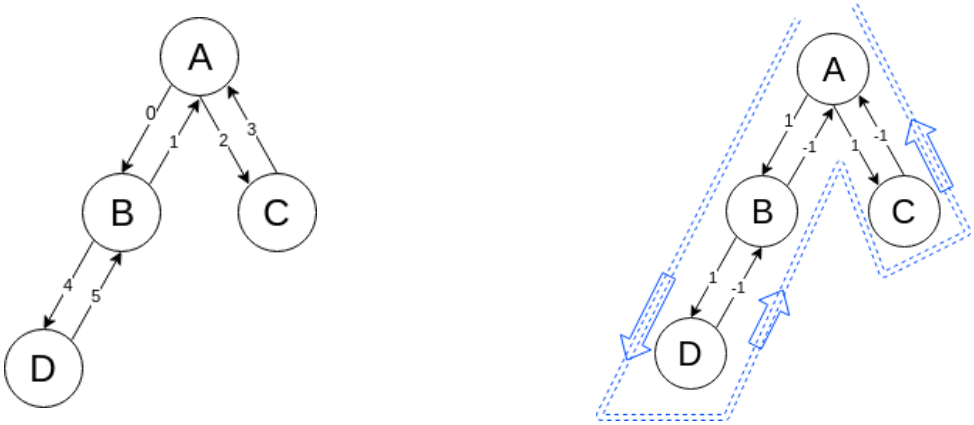
Algoritmus Výpočtu urovne vrchola bol spracovaný tak ako bol uvedený na prednáškach [1], vykonáva výpočet vrcholov stromu s n procesormi. Strom obsahuje m uzlov, z toho na ich prepojenie potrebujeme $n-1$ hrán, avšak každú hranu potrebujeme zdvojit. Pre hranu ktorá ide smerom ku koreňu, nazvyme ju zpatná hrana zvolíme -1 a pre hranu ktorá ide smerom od koreňa, nazvyme ju dopredná hrana, zvolíme hodnotu 1. Teda budeme potrebovať $m = 2 * n - 2$ procesorov.

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

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, zlava doprava úroveň 1 a syn vrcholu B, teda D, úroveň 2. Počet procesorov potrebných na paralelný výpočet bude počet hrán, ktoré vzniknú medzi uzlami a to pre každu doprednú a spatnú hranu, teda $2 * m - 2$. Pre náš príklad 4 uzlov to činí 6. Grafický načrt vyzerá nasledovne, obrázok 1a.



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

(b) A subfigure

Obr. 1: Grafické znázornenie Eulerovej cesty stromo.

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 priraduje je demonštrovný na obrázku 1a. Spracovanie vstupu prebiehe v každom procese osobite. Každý jeden procesor 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 nemu inverznej a akú ma váhu. Váhu si vypočíta podľa toho či je hrana dopredná alebo spetná. Ak je dopredná tak ma váhu 1 ak je spetná, 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 procesor 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 uloži pod indexom svojho id procesru hranu, ktorú považuje za svojho následníka. A to tak, že si v zozname súsedností zistí, kde je jeho inverzná hrana a tak odhadne 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

2.1 Rozbor zložitosti

Každý jeden procesor túto informáciu distribuuje pomocou MPI_Allgather medzi všetky ostatné procesory a pomocou MPI_Barrier za týmto príkazom čaká, až všeetky procesory dorazia do tohto bodu programu, aby mohol porkačovať. V tomto bode vieme, že každý proces vlastní rovnaké pole, kde je celá Eulerova cesta stromom.

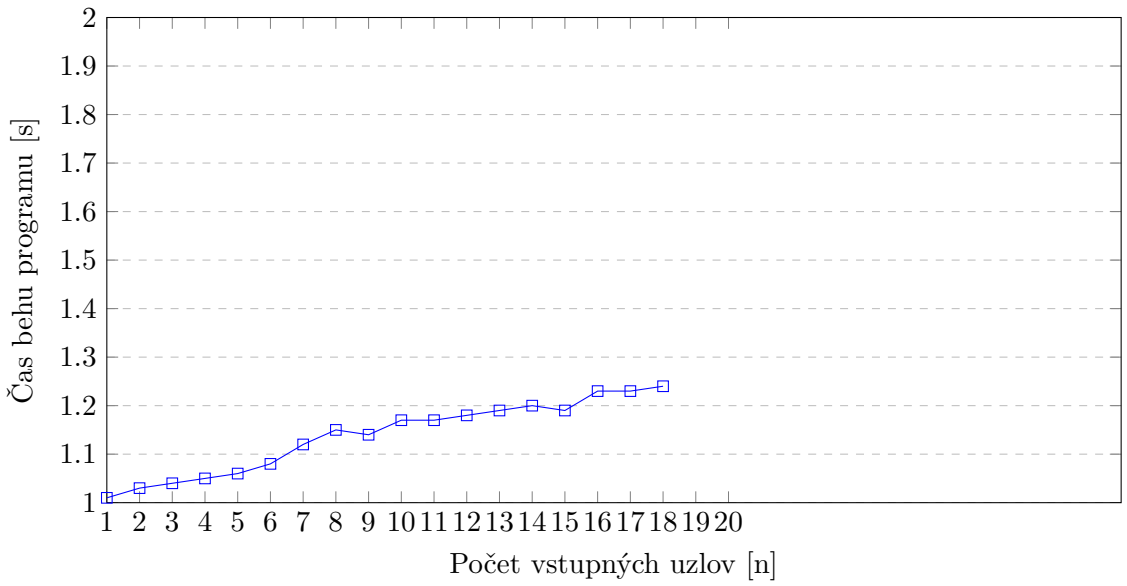
Predposledným krokom je spočitanie suffix sum pre danú Eulerovu cestu na ktorom sa podieľajú všetky procesory.

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

V poslednom kroku už len v našom prípade štandardne nultý proces pr9jme 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 32. Pre každé čislo bolo vykonaných 40 meraní, kde krajných 5 hodnôt bolo odstránených a zvyšných š0 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 radí, v tomto projekte som sa rozhodol merať 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 radenie postupností – Bucker Sort. Výsledkom je funkčný paralelný program schopný radiť postupnosti do maximálnej dĺžky 100 000 prvkov (tento limit je určený serverom Merlin, ktorý nám viac procesorov neposkytol). Uskutočnené merania ukázali, že reálna časová zložitosť programu je skutočne približne lineárna (ako sa očakávalo).

A Sekvenčný diagram

Obr. 2: Sekvenčný diagram programu