Dokumentace projektu do předmětu Paralelní a distribuované systémy
Implementace algoritmu pro výpočet úrovně vrcholu

Tomáš Aubrecht

1 Úvod

Cílem tohoto projektu bylo implementovat algoritmus pro výpočet úrovně vrcholu v binárním stromě za pomoci jazyka C++ s využitím knihovny Open MPI. Také bylo potřeba vytvořit řídící skript, který vypočítá počet procesorů na základě vstupního parametru. Tímto parametrem je řetězec, jenž reprezentuje hodnoty jednotlivých uzlů binárního stromu. Nad tímto řetězcem se spustí implementovaný algoritmus.

2 Rozbor a analýzu algoritmu

2.1 Eulerova cesta

Eulerova cesta nebo také eulerovský tah představuje takový tah, který projde každou hranu grafu právě jednou. V oblasti stromových struktur se jedná o obecný průchod binárním stromem. Pro tento průchod je potřeba mít orientovaný graf. Ten z daného stromu získáme nahrazením každé hrany (u, v) dvěmi orientovanými hranami *<u, v>* a *<v, u>*. Tím získáme Eulerovský graf, který obsahuje orientovanou kružnici, která prochází každou hranu právě jednou. Speciálními případy Eulerovy cesty jsou například průchody preorder, inorder a postorder.

2.2 Suma suffixů

Suma suffixů je operace, jež má na vstupu binární asociativní operátor $\oplus$ a uspořádanou posloupnost prvků $[a_0, a_1, a_2, \ldots a_{n-1}]$. Výsledkem této operace je posloupnost $[(a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1}), \ldots (a_{n-3} \oplus a_{n-2}), a_{n-1}]$. Tedy *i-tý* prvek ve výsledné posloupnosti se vypočítá následovně: $y_i = x_i \oplus y_{i+1}$. Tato operace se používá například pro implementaci některých stromových operací, v našem případě pro výpočet úrovně vrcholu.

2.3 Výpočet úrovně vrcholu

Algoritmus pro výpočet úrovně vrcholu je paralelní algoritmus, který pracuje na stromové architektuře. Samotná úroveň vrcholu představuje počet hran, které se nachází na cestě od kořene k danému vrcholu nebo také rozdíl počtu zpětných a dopředných hran na zbytku Eulerovy cesty od daného vrcholu ke kořeni. A právě tohoto rozdílu využívá i tento algoritmus.

Při výpočtu postupuje tak, že v prvním kroce ohodnotí každou hranu orientovaného grafu. Pokud se jedná o dopřednou hranu, přiřadí ji hodnotu *-1*, pokud se jedná o zpětnou hranu, přiřadí ji hodnotu *1*. Ve druhém kroce provede výpočet výše zmiňované sumy suffixů, jejímiž vstupními parametry je Eulerova cesta daného stromu a inicializované váhy hran. V posledním kroce projde všechny hrany a pokud je daná hrana *<u, v>* dopředná, nastaví úroveň vrcholu *v* na hodnotu váhy dané hrany *+ 1*. Úroveň kořene se musí zvlášť nastavit na hodnotu *0*. To je z důvodu nepřítomnosti hrany, která by byla dopředná a směřovala by do kořene.

Asymptotická časová složitost tohoto algoritmu je *O(log(n))*, kde *n* je celkový počet hran. Časová složitost vychází ze složitosti jednotlivých kroků. Inicializace vah všech hran v prvním kroce je provedena v konstantním čase. Výpočet sumy suffixů na posloupnosti s *n* prvky na stromové architektuře je provedeno s časovou složitostí $O(log_2(n))$. Pro ohodnocení všech vrcholů stromu s *m* vrcholy je potřeba *2m – 2* procesorů. Výsledná cena algoritmu pro výpočet úrovní vrcholů je tedy $O(n \cdot log(n))$.

3 Implementace

Algoritmus je implementovaný v jazyce C++ za použití knihovny OpenMPI pro paralelní výpočty, kde pracuje na binárním stromě procesorů. Na začátku programu je nutné volat funkci MPI_Init(), která nainicializuje paralelní prostředí. Dále je třeba si uložit celkový počet procesorů vykonávající daný program a ID konkrétního procesoru. Každý procesor si následně vypočítá index uzlu, ze kterého jeho hrana vychází, a index uzlu, do kterého směřuje. K tomuto je využito preorder průchodu stromem, kde jsou hrany přidělovány procesorům postupně podle jejich ID. Index procesoru tedy odpovídá pořadí dané hrany v Eulerově cestě. Pro reprezentaci hrany je zde naimplementovaná třída *Edge*.

Samotný výpočet začíná inicializací vah všech hran, kde každý procesor si určí váhu na základě toho, zdali je jeho hrana zpětná či dopředná. To zjistí podle indexu uzlu, ze kterého vychází a do kterého směřuje. Pokud je index výchozího uzlu menší než cílového, pak je to hrana dopředná a její váha je -1, jinak 1. Dalším krokem je výpočet sumy suffixů.

Při výpočtu sumy suffixů se využívá toho, že index každého procesoru udává pořadí jeho hrany v Eulerově cestě. Každý procesor si tedy určí počet iterací, který odpovídá $log_2(n)$, kde *n* je celkový počet procesorů/hran. Následně si v cyklu s tímto počtem iterací vypočítá index předchozího procesoru a index následujícího procesoru. Index předchozího procesoru se rovná $id - 2^i$ a následujícího procesoru $id + 2^i$, kde *id* je index daného procesoru a *i* je číslo dané iterace. Pokud je index předchozího procesoru větší nebo roven nule(má předchůdce), odešle mu svoji sumu suffixů, která je inicializovaná na váhu jeho hrany. Obdobně pokud je index následujícího procesoru menší než je celkový počet procesorů, tak od něj příjme jeho sumu suffixů. Tu si pak přičte ke své sumě a pokračuje další iterací. Po dokončení všech iterací má každý procesor vypočtenou sumu suffixů.

Po získání sumy suffixů si procesory reprezentující dopředné hrany provedou korekci, kde upraví svou váhu, a tím nastaví úroveň uzlu, do kterého dopředná hrana směřuje. Daná úroveň uzlu odpovídá sumě suffixů + 1. Nakonec každý procesor, který reprezentuje dopřednou hranu, vytiskne daný vrchol a jeho úroveň. Pro synchronizaci tisku je použita bariéra, kde se v cyklu, jehož počet iterací odpovídá počtu vrcholů, postupně tisknou vrcholy v původním pořadí.

Na konci programu je volána funkce *MPI_Finalize()*, která uvolní a ukončí paralelní prostředí.

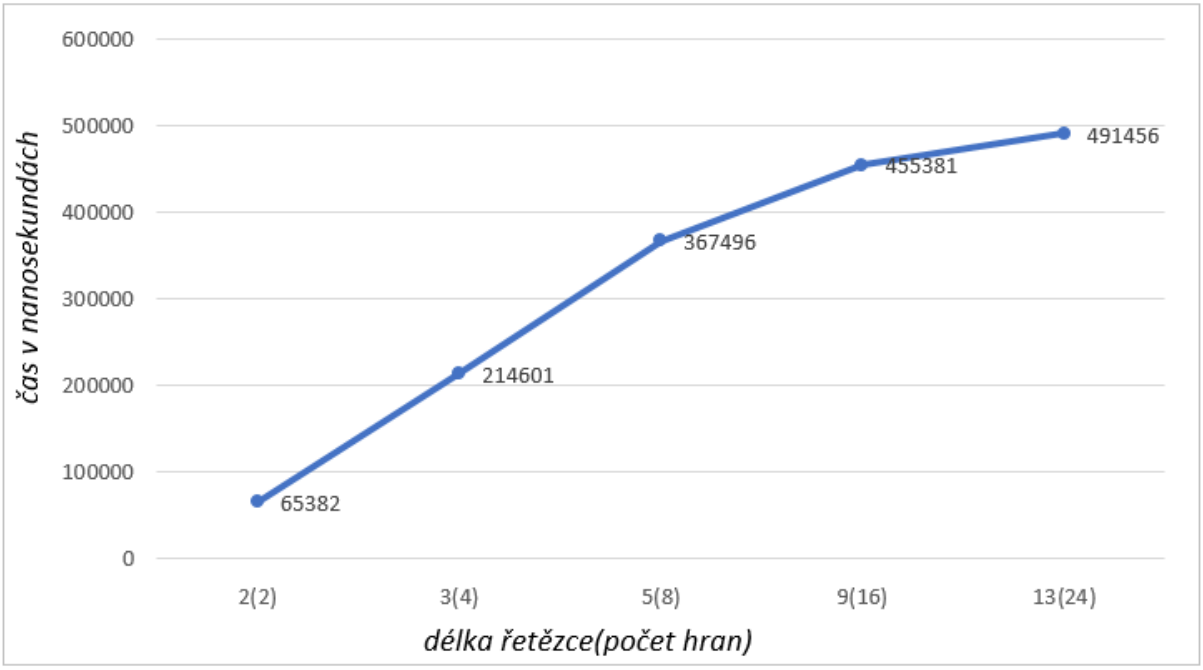
4 Experimenty

Na naimplementovaném algoritmu bylo provedeno několik experimentů s různě dlouhými vstupními řetězci pro ověření časové složitosti. Knihovna *Open MPI* neobsahuje žádné

implicitní metody pro měření složitosti algoritmů, proto byl naimplementován vlastní způsob měření času pomocí knihovny <*time.h*>.

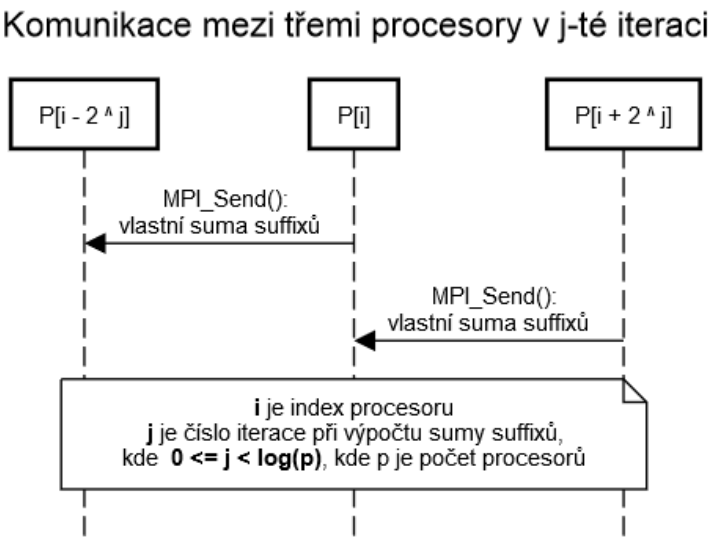
Algoritmus byl testován na několika různých vstupech o délkách 2, 3, 5, 9 a 13 znaků, kde nezáleží na jednotlivých znacích, ale pouze na délce vstupu, proto nebylo třeba vytvářet různé řetězce pro jednotlivé délky. Každý ze vstupních řetězců byl testován 120 krát. 10 minimálních a 10 maximálních naměřených hodnot na každém vstupu bylo zahozeno a zbylých 100 se zprůměrovalo. Dohromady tedy bylo provedeno 600 testů.

Pro účely ověření časové složitosti algoritmu není do měření započtena inicializace MPI prostředí, inicializace hran(ne jejich vah) a veškeré výpisy do konzole. Měřily se tedy pouze jednotlivé kroky algoritmu popsané v kapitole 2. Pro spuštění programu za účelem měření stačí předefinovat macro *MEASURE_PERFORMANCE* ve zdrojovém kódu na hodnotu *true*. Jednotlivé naměřené hodnoty můžete vidět v grafu uvedeném níže.



5 Komunikační protokol

Na níže uvedeném sekvenčním diagramu můžete vidět komunikaci mezi třemi procesory z pohledu prostředního, který se právě nachází v *j-té* iteraci smyčky pro výpočet sumy suffixů. Procesory mezi sebou komunikují pomocí funkcí *MPI_Send()* a *MPI_Recv()*.



6 Závěr

Po provedení experimentů bylo zjištěno, že teoretická časová složitost algoritmu pro výpočet úrovně vrcholu popsána v kapitole 2 odpovídá grafu naměřených hodnot, na kterém lze vidět, že s rostoucím počtem hran logaritmicky roste i celkový čas výpočtu.

Testování bylo provedeno na referenčním serveru merlin s operačním systémem CentOS pomocí výše popsané testovací sady. Veškeré testy skončili úspěchem.