Výpočet úrovně vrcholu

Tomáš Blažek (xblaze31)
PRL - Paralelní a Distribuované algoritmy
Fakulta Informačních Technologií, Brno

1 Rozbor a analýza algoritmu

Algoritmus pro Výpočet úrovně vrcholu je algoritmus, který pracuje s binárním stromem, který má místo jedné orientované hrany mezi dvěma uzly hned dvě orientované hrany, a to jednu dopřednou a druhou zpětnou, přičemž procesory reprezentují hrany vedoucí grafem. Počet použitých procesorů je závislý od počtu uzlů grafu n. Celkový počet procesorů je tedy roven $p = 2n - 2$, což udává prostorovou složitost $O(n)$. Každému procesoru přiřadíme index a první polovina indexů procesorů bude znázorňovat dopředné hrany a druhá polovina hrany zpětné. Algoritmus pracuje tak, že nejprve každá hrana si zjistí svoji váhu (1 nebo -1), zdali je dopředná (-1) nebo zpětná (1), a to dokážeme určit v konstantním čase, tedy dostaneme časovou složitost $O(c)$. Dále podle algoritmu vypočteme váhu jednotlivých hran za pomoci paralelního algoritmu Suffix Sum, který bude pracovat s polem následníků tvořeného podle Eulerovi cesty. Za použití Eulerovy cesty vznikne takový graf, který obsahuje orientovanou kružnici, která prochází každou hranou právě jednou.

Následníka Eulerovi cesty lze určit na základě pravidel dvou úrovní. V první úrovni se zjišťuje zdali je hrana dopředná nebo zpětná $O(c)$. Pokud je hrana dopředná, tak se zjišťuje zdali uzel na něž hrana míří má levé syna. V případě, že levého syna má, tak následník $next(x)$ má index $next(x) = x + (n - 1)$. V opačném případě má následník index $next(x) = 2 \cdot (x+1)$. Ovšem pokud je hrana zpětná, tak se zjišťuje zdali uzel na něž zpětná hrana míří má pravého syna. Pokud má uzel má pravého syna a hrana nevychází z daného pravého syna, tak index následníka je roven $next(x) = x-(n-1)+1$. V opačném případě je následníkův index $next(x) = ((x-n-1)/2)-1$ až na výjimku, kde pokud hrana míří na kořenový uzel, tak její následník je daná hrana (kvůli algoritmu Suffix Sum). Z toho plyne, že výpočet následníka podle Eulerovy cesty má konstantní časovou náročnost $O(c)$.

Paralelní algoritmus Suffix Sum má tedy připravené inicializační pole následníků a připraví se tedy ještě pole hodnot, kde hodnota na indexu je rovna váze hrany, kromě hrany jejíž následník je hrana samotná. Ta má potom hodnotu 0. Tedy časová náročnost tohoto kroku je konstantní $O(c)$. Další krok je samotný výpočet sumy suffixů a ten pracuje v $log_2(n)$ krocích, jelikož v každém dalším kroku je možné využít výsledků následníka. Jelikož posuv je vždy na následníka následníka, tak se krok vždy zdvojnásobuje s každou iterací, proto stačí použít pouze $log_2(n)$ kroků tedy časová náročnost je $O(log_2(n))$. Poslední krok algoritmu Suffix Sum je korekční, kdy se přičte všem členům hodnota poslední hrany Eulerovy cesty, která byla na začátku hrazena hodnotou 0, což máš konstantní časovou náročnost $O(c)$. Celkově tedy algoritmus Suffix Sum má časovou náročnost $t(n) = O(c) * O(log_2(n)) * O(c) = O(log_2(n))$.

Poslední krok algoritmu pro výpočet úrovně vrcholu je korekce výsledku, která přičte ke všem hodnotám vah jedničku a výsledkem toho je úroveň v daném binárním stromě.

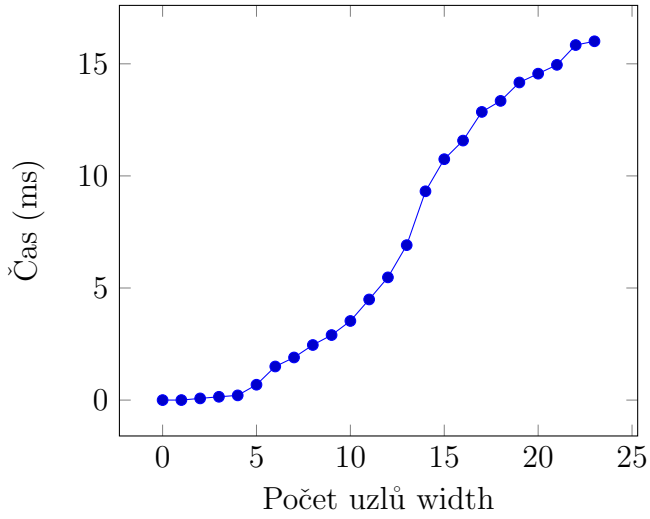
$$\begin{aligned} \text{Časová složitost: } t(n) &= O(c) + O(log_2 n) + O(c) = O(log_2(n)) \\ \text{Prostorová složitost: } p(n) &= O(n)) \\ \text{Celková cena: } c(n) &= O(n) \cdot O(log_2(n)) = O(n \cdot log_2(n)) \end{aligned} \tag{1}$$

2 Implementace

Program je implementován v jazyce C++ s využitím knihovny Open MPI. V samotné implementaci programu se jako první operace inicializuje MPI komunikace a zjistí se informace o tom, kolik procesorů běží, přičemž si každý procesor zjistí svoje identifikační číslo (id). Nejprve si každý procesor (reprezentující hranu) alokuje místo pro pole následníků `succ` a pole hodnot `val`. Dále si každý procesor inicializuje v poli následníků pouze index svého následníka a následníka svého následníka (nalezení následníka je popsáno v kapitole 1) a v poli hodnot inicializuje pouze svoji hodnotu (váhu pro zpětnou hranu 1 nebo -1 pro dopřednou). Před začátkem výpočtu samotné sumy suffixů se zavolá z každého procesoru funkce `MPI_Allgather` z knihovny MPI, která zašle broadcast své hodnoty `val[id]` všem ostatním vláknům a uloží je do pole `val` podle indexů procesorů. Dále přichází na řadu cyklus samotného výpočtu sumy suffixů. Cyklus je proveden $log_2(n)$–krát a v každém kroce cyklu se k hodně `val[id]` přičte hodnota jejího následníka a následník `succ[id]` se změní na následníka jeho následníka tedy `succ[succ[id]]`. Na konci každé iterace se zavolá ještě funkce `MPI_Allgather`, pro synchronizaci hodnot v polích. Při dokončení výpočtu sumy suffixů je v poli `val` na indexu `id` výsledná hodnota úrovně uzlu na něž hrana (procesor) s `id` míří, která ještě musí být zkorigována přičtením jedničky zanedbané ve výpočtu Suffix Sum (jako korekční krok) a ještě jedné jedničky jako korekce samotného algoritmu pro výpočet úrovně vrcholu (ve výsledku tedy ke každé hodnotě +2). Kořen má úroveň implicitně rovnu 0. Na závěr se ještě zašlou zkorigované výsledky do procesoru s `id` rovnu 0 za pomoci funkce `MPI_Gather` a ten vypíše výsledky na standardní výstup (stdout).

3 Experimenty

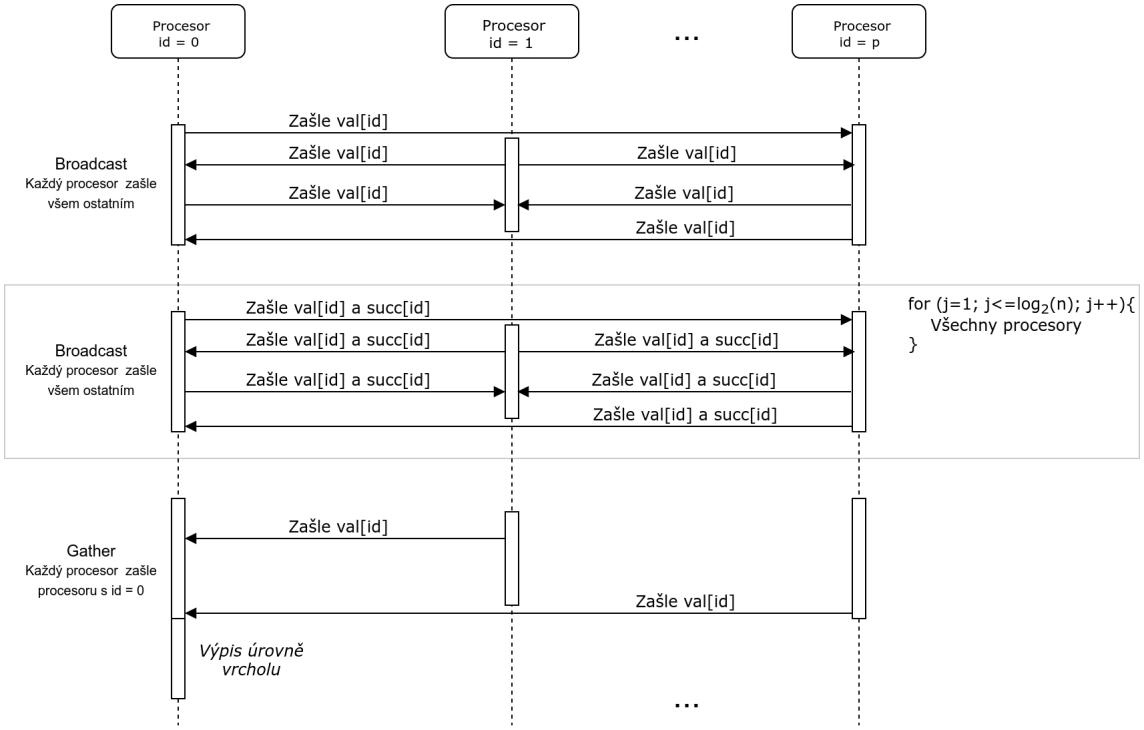
Nad implementovaným algoritmem byly prováděny experimenty v podobě opakovaného spouštění algoritmu s přidávání jednoho uzlu navíc s každým dalším krokem a byla měřena doba, za níž algoritmus dokončí svůj výpočet. Časové hodnoty byly získány za pomoci funkce přímo z knihovny Open MPI, která vrací čas v sekundách, ale kvůli lepší vizualizaci jsou hodnoty v grafu převedeny na milisekundy. Časové údaje byly při experimentu měřeny pouze pro samotný algoritmus, tedy byl z měření vynechán konečný tisk výsledku na standartní výstup.



Obrázek 1: Graf výsledku experimentů časové náročnosti

4 Komunikační protokol

Komunikační protokol je znázorněný pomocí sekvenčního diagramu. Ten popisuje komunikaci mezi jednotlivými procesory, které jsou zobrazeny za pomoci tří zástupců. Diagram byl takto navržen, aby popis mohl být obecný pro libovolný počet procesorů s výjimkou použití jednoho procesoru. To je z důvodu toho, že při použití pouze jednoho procesoru neprobíhá žádná komunikace a výsledek je rovnou uložen v paměti kořenového procesoru. Proměnné použité v diagramu jsou definovány v kapitole 1.



Obrázek 2: Sekvenční diagram komunikace procesorů

5 Závěr

V kapitole experimentů z grafu plyne, že algoritmus má sice rychle rostoucí tendenci při aplikaci na malém počtu uzlů ale při aplikaci na více uzlech se časový průběh postupně měnil na logaritmický, což potvrzuje odhad časové složitosti z rozboru a analýzy algoritmu, který byl určen jako $O(log_2(n))$. Experimenty byly prováděny na školním serveru Merlin s operačním systémem CentOS.