Paralelní a distribuované algoritmy
Přiřazení pořadí preorder vrcholům

Nikola Valešová
xvales02@stud.fit.vutbr.cz

28. dubna 2018

1 Úvod

Cílem projektu je implementace algoritmu přiřazení pořadí preorder vrcholům v jazyce C/C++ s využitím knihovny Open MPI a následná analýza časové a prostorové složitosti daného algoritmu.

2 Vstup programu

Vstupem programu je libovolná posloupnost ASCII znaků, jejíž znaky reprezentují hodnoty uzlů binárního stromu zapsaného v poli. Indexování začíná od 1 a každý i-tý uzel má svou hodnotu na i-tém místě pole. Na indexu $2 \cdot i$ se nachází hodnota jeho levého potomka a na indexu $(2 \cdot i) + 1$ je uložena hodnota jeho pravého potomka.

3 Výstup programu

Výstupem programu jsou stejné znaky, které byly na vstupu, avšak v pořadí, které odpovídá pořadí získanému preorder průchodem stromu, který vznikl výše uvedeným způsobem.

4 Popis algoritmu

Binární strom je na vstupu v podobě řetězu reprezentujícího hodnoty uzlů ve stromu. Cílem algoritmu je tento strom zkonstruovat a provést jím preorder průchod s vypsáním správně seřazených hodnot uzlů.

Pokud budeme uvažovat vstupní řetěz o n znacích, algoritmus ke své činnosti vyžaduje $(n-1) \cdot 2$ procesorů. Z každé hrany v grafu tedy uděláme dvě orientované hrany (jednu dopřednou a druhou zpětnou) a každý procesor poté představuje jednu nově vzniklou hranu.

Nejprve proběhne předzpracování, kdy je ze vstupní stromové struktury vytvořen seznam sousednosti (*adjacency list*), ve které záznam o každé hraně nese informace o pořadovém čísle hrany, o hraně k ní opačné (dopředné/zpětné mezi stejnými dvěma uzly) a o hraně následující vycházející ze stejného vrcholu.

Po předzpracování může začít samotný algoritmus průchodu. Prvním krokem je spočtení Eulerovy cesty, které má na vstupu seznam sousednosti a jeho výstupem je pole *Etour*. Toto pole udává pro každou hranu, jaká hrana po ní následuje v Eulerově cestě daným stromem. Pro určení následné hrany byl implementován algoritmus popsaný v přednášce.

V dalším kroku je paralelně každé hraně přiřazena váha. Jestliže se jedná o hranu dopřednou, získá váhu 1, hrany zpětné obdrží váhu 0. Nad polem vah a Eulerovou cestou je vypočtena suma suffixů. V každém procesoru jsou k váze postupně přičítány hodnoty vah hran, které se v Eulerově cestě nachází až za hranou aktuální. Takto získáme pro každou hranu počet dopředných hran, které za ní v Eulerově cestě následují.

V posledním kroku algoritmu se provede korekce výsledku a s využitím vztahu $preorder(v) = n - weight(e) + 1$ (kde e je hrana vedoucí z uzlu u do uzlu v a n je celkový počet uzlů ve stromu) je získaná váha hrany převedena na hodnotu udávající pořadí uzlu při preorder průchodu stromem.

5 Analýza složitosti algoritmu

Počet procesorů lze vyjádřit vztahem:

$$p(n) = (n - 1) \cdot 2,$$

kde n je rovno délce vstupní posloupnosti.

První krok, kterým je zpracování vstupu a vytvoření seznamu sousednosti (*adjacency list*), má kvadratickou časovou složitost $O(n^2)$. V poslední fázi probíhá sekvenční vypsání seřazené posloupnosti, které má složitost $O(n)$.

Analýza časové složitosti samotného preorder průchodu:

- výpočet Eulerovy cesty má při paralelní implementaci konstantní časovou složitost $O(c)$,
- inicializace váhy hrany má také konstantní časovou složitost $O(c)$,
- spočtení sumy suffixů při paralelním zpracování dosahuje logaritmické složitosti $O(log\, n)$,
- korekce výsledku a určení pořadí hrany ve výsledném seznamu má opět konstantní časovou složitost $O(c)$.

Celková teoretická časová složitost lze popsat vztahem:

$$t(n) = O(log\, n).$$

Z počtu procesorů a časové složitosti jsme schopni odvodit celkovou cenu algoritmu:

$$c(n) = t(n) \cdot p(n) = O(n \cdot log\, n).$$

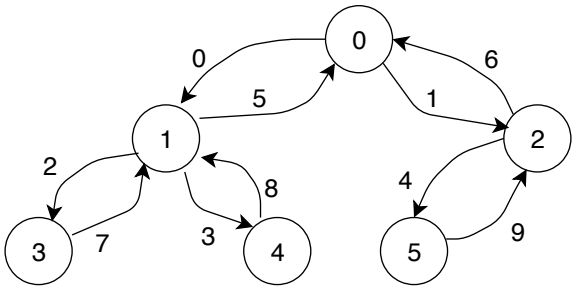
Algoritmus tedy není optimální.

6 Implementace

Pro implementaci byl zvolen jazyk C/C++ a knihovna Open MPI.

Každý procesor dostane na vstupu kompletní vstupní posloupnost, jehož délka udává počet uzlů binárního stromu. Nejprve zkontroluje, zda strom obsahuje více než 1 uzel. Pokud ne, vypíše svou hodnotu na výstup a program skončí. V opačném případě je vstupní řetěz zpracován a je z něj vytvořen seznam sousednosti. Na něj je aplikován algoritmus na vytvoření Eulerovy cesty. Následně je vypočtena suma suffixů a vytvořeno pole seřazených hodnot.

Vizualizace implementovaného způsobu číslování hran je znázorněna na obrázku 1. Nejprve jsou číslovány dopředné hrany od kořene směrem k listům, poté ve stejném pořadí hrany zpětné, přičemž i-tý procesor reprezentuje i-tou hranu. Tento způsob číslování byl zvolen za účelem snadného výpočtu indexů opačných hran, přilehlých uzlů apod.



Obrázek 1: Způsob číslování uzlů a hran ve stromu

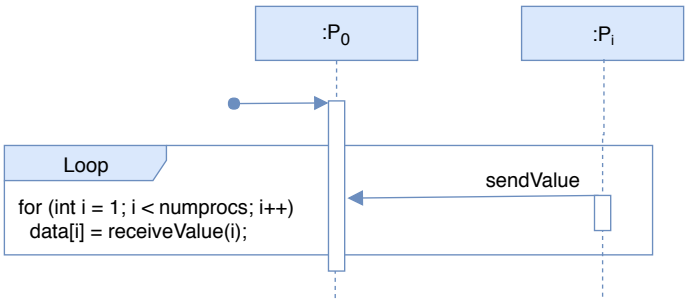
Procesor s identifikačním číslem 0 má zvláštní postavení, provádí výpis seřazených hodnot na výstup a také při vzájemné komunikaci více procesorů je právě on tím, kdo přijímá hodnoty od ostatních procesorů a postupně plní celé pole. Po dokončení přijímání zpráv začne rozesílat sestavené pole pomocí algoritmu *broadcast*, který slouží k simulaci simultánního čtení, když není umožněn přímý paralelní přístup.

Vzájemná komunikace mezi procesory je implementována ve funkci `exchangeInformation`. Ta pro procesor s identifikačním číslem 0 zavolá funkci `receiveAndBroadcast`, ve které procesor postupně přijímá hodnoty od ostatních procesorů a poté iniciuje *broadcast* komunikaci přeposláním

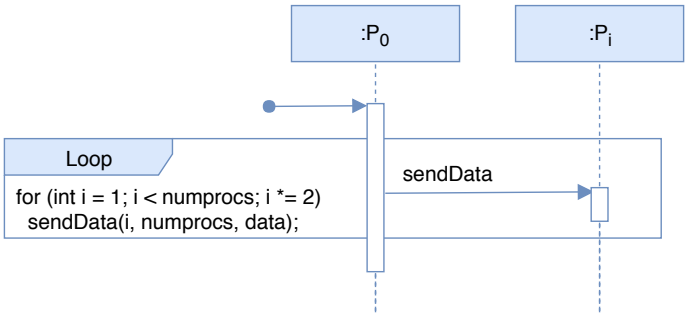
naplněného vektoru procesoru s identifikačním číslem 1. Pro ostatní procesory s identifikačním číslem $i \neq 0$ se z funkce `exchangeInformation` zavolá funkce `sendAndBroadcast`, v níž procesor nejprve pošle svou hodnotu procesoru 0, poté čeká na příjem naplněného pole od procesoru s identifikačním číslem $i - 2^{trunc(log_2(i))}$ a následně započne sdílení pole dalším procesorům podle algoritmu *broadcast*.

Funkce `exchangeInformation` je použita pro sdílení informací například při vytváření sumy suffixů nebo při výpočtu Eulerovy cesty.

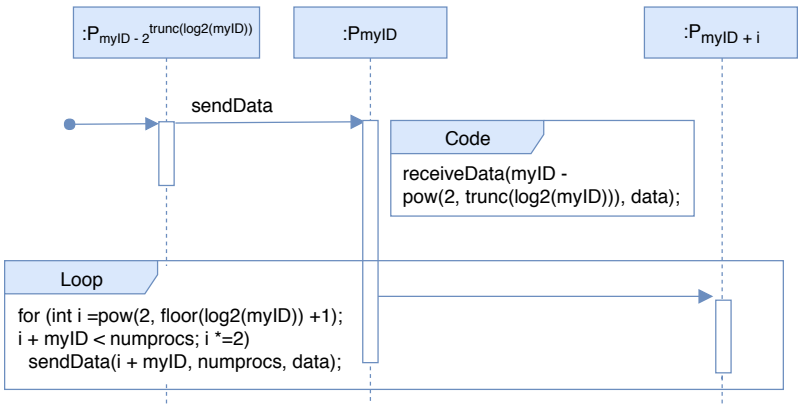
Komunikační protokol znázorňující způsob a následnost zasílání zpráv mezi jednotlivými procesory je znázorněn na obrázku 2.



(a) Schema komunikace pro procesor s identifikačním číslem 0 při plnění pole hodnot od jednotlivých procesorů



(b) Schema komunikace pro procesor s identifikačním číslem 0 při rozesílání naplněného pole dat s využitím algoritmu broadcast



(c) Schema komunikace procesorů s identifikačním číslem větším než 0

Obrázek 2: Komunikační protokol zasílání zpráv

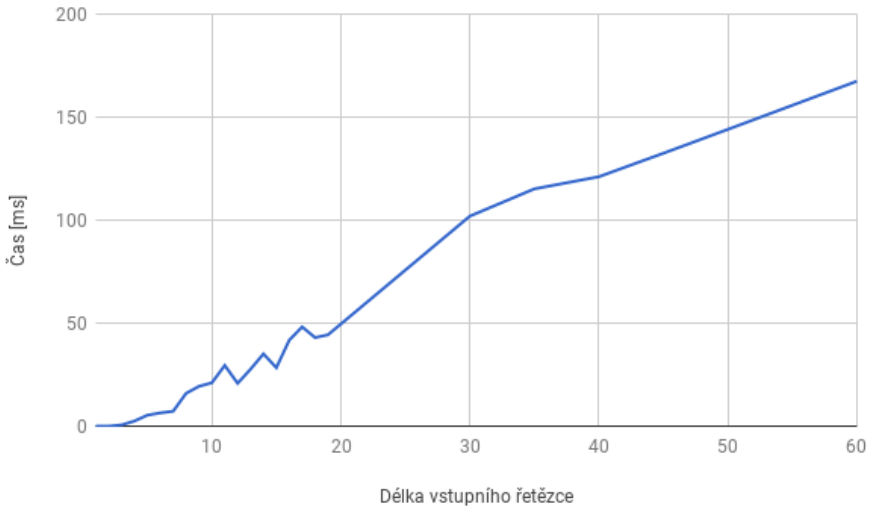
7 Experimenty

Pro praktické ověření časové složitosti algoritmu byla vytvořena sada 25 testovacích vstupů. Délka vstupního řetězce se v nich pohybovala v rozmezí od 1 znaku do 60-ti znaků. Pro každou délku vstupního řetězce bylo měření spuštěno celkem třicetkrát a do grafu byl poté zanesen průměr získaných hodnot. Aby se do statistiky výrazněji nepromítaly extrémní hodnoty, byla nejprve od-

straněna minimální a maximální hodnota a teprve poté proběhlo samotné průměrování.

Měření času výpočtu prováděl vždy procesor 0. Pro získání doby běhu byly využity příkazy `now` a `duration` z knihovny *chrono*. Měření bylo započato před začátkem výpočtu Eulerovy cesty a zastaveno po získání kompletního pole *Rank*, tedy těsně před započetím výpisu hodnot na výstup.

Na obrázku 3 je zobrazena křivka představující experimentálně získanou závislost doby běhu programu na počtu řazených prvků. Teoretická časová složitost algoritmu je $t(n) = O(log\, n)$. Získaná křivka má zpočátku lineární, dále však spíše logaritmický průběh. Pro malé délky vstupního řetězce (konkrétně v rozmezí mezi 10 a 20 znaky) si můžeme povšimnout výrazného kolísání, které je pravděpodobně způsobeno režií při simulování procesorů.



Obrázek 3: Závislost doby běhu programu na délce vstupního řetězce

8 Závěr

V rámci projektu byl implementován paralelní algoritmus přiřazení pořadí preorder vrcholům a pomocí experimentů byla ověřena jeho časová složitost. Experimentálně získaná časová složitost odpovídala očekávané logaritmické složitosti, a to především pro větší délky vstupního řetězce, kdy se do času výpočtu již tolik nepromítá inicializace a režie simulování procesorů.