1 Rozbor a analýza algoritmu

1.1 Časová složitost

Vzhledem k tomu, že algoritmus *Euler tour* popsaný ve studijních materiálech nebere v potaz prvotní distribuci hran mezi procesory (stejně tak jako distribuci vah po skončení *parallel suffix sum*), nebudu v rozboru analýzy algoritmu tyto části zohledňovat, aby bylo možné algoritmus teoreticky a prakticky porovnat. Počet procesorů se rovná počtu zpětných a dopředných hran ve stromě, což pro n prvků znamená $2 \cdot n - 2$ procesorů.

Každý procesor paralelně na základě *adjecency list* a funkcí *next* zjistí následníka v *Euler tour*. Časová složitost je konstantní, tedy

$$O(1).$$

Následně proběhne algoritmus *parallel suffix sum*, který obsahuje pro každý procesor sekvenční cyklus. První prvek v seznamu se na konec seznamu dostane v logaritmickém čase, složitost je tedy

$$O(\log(2 \cdot n - 2)).$$

Poté proběhne korekce výsledků v konstantním čase. Celkově tedy algoritmus *přiřazení pořadí preorder vrcholů* proběhne v logaritmickém čase, tedy

$$O(\log(2 \cdot n - 2)).$$

1.2 Prostorová složitost

Na začástku algoritmu je třeba uchovat pole všech hran a pole následníků *succ*, dále v případě *Euler tour* navíc *adjecency list*, který počítá s polem o dvou prvcích (a případně s ukazatelem do dalšího pole) pro každou dopřednou i zpětnou hranu. Nakonec u algoritmu *suffix sum* je třeba vytvořit nové pole pro všechny hrany. Každá z těchto složitostí je lineární, celkově má tedy algoritmus lineární prostorovou složitost, a to

$$O(2 \cdot n - 2).$$

1.3 Celková cena

Cena algoritmu se spočítá jako časová složitost krát počet procesorů, v případě *přiřazení pořadí preorder vrcholů* tedy

$$O((\log(2 \cdot n - 2)) \cdot (2 \cdot n - 2)).$$

Zda je algoritmus optimální se určí porovnáním s časovou složitostí ideálního sekvenčního algoritmu pro *preorder* průchod stromem. Vzhledem k tomu, že pro průchod *preorder* existuje algoritmus s lineární časovou složitostí $O(n)$, je jasné, že tento algoritmus nelze označit za optimální.

2 Implementace

Vzhledem k tomu, že po spuštění programu má každý procesor přístup ke vstupnímu stromu, si každý sám, tedy paralelně, přiřadí jednu hranu stromu. Při tomto přiřazení si také každý procesor uchovává data o tom, který procesor získal kterou hranu. Tato informace je uložena ve struktuře `std::map`.

Dalším krokem je zjištění další hrany v *Euler tour*. Jelikož ve studijních materiálech je k tomuto kroku využita sdílená paměť, kterou ale ve své implementaci nepoužívám, byl tento krok implementován jinak (spojuje dohromady funkce *adjecency list* a *next*). Jelikož má každý procesor přístup ke vstupnímu stromu, dokáže si další hranu v cestě spočítat bez komunikace se sdílenou pamětí, respektive bez komunikace s ostatními procesory. Tímto je pro tento krok správně docíleno konstantní časové složitosti.

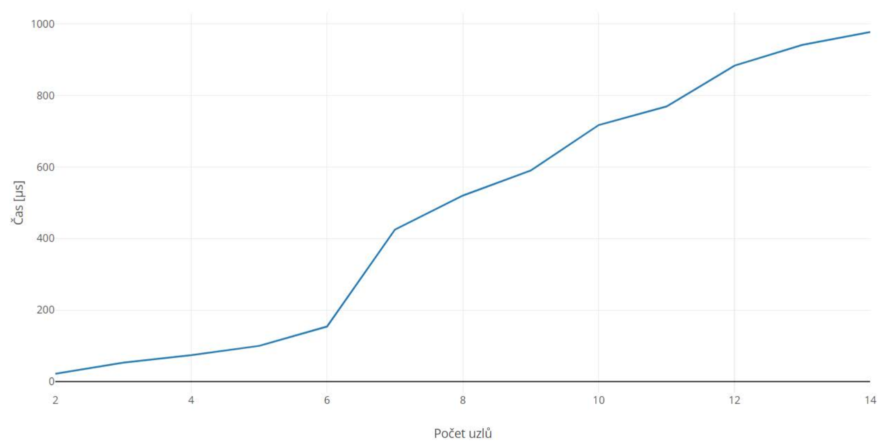
K implementaci *suffix sum* ale má implementace počítá s tím, že že každý procesor musí znát i vlastníka následující hrany v *Euler tour* - zde je využito uložení informace o tom, který procesor spravuje kterou hranu. Každý procesor se tedy podívá do `std::map` a přiřadí následníka v *Euler tour* s *rankem* procesoru. Nalezení v `std::map` má dle standardu `C++` logaritmickou časovou složitost. Posledním předzpracováním před samotným *suffix sum* je otočení pořadí procesorů v cestě - každý procesor zašle svůj *rank* svému následníku a čeká na zprávu od předchůdce. Tím je zajištěno, že poslední procesor v cestě bude nést *rank* 0.

Následně proběhně algoritmus *suffix sum* ve stejné formě, jako ve studijních materiálech - s tím rozdílem, že poslední procesor (v mém případě *rank* 0) neposlouchá a ani nic neposílá. Tedy jakmile kterýkoliv procesor ukazuje na *rank* 0, nepožaduje žádná data a rovnou si přičte váhu 0. Jedině tak lze u tohoto algoritmu dosáhnout logaritmické časové složitosti bez sdílené paměti, jelikož pokud by měl procesor s *rankem* 0 poslouchat a odpovídat na každý požadavek o váhu, časová složitost by rostla exponenciálně (první cyklus poslouchá jedenkrát, druhý dvakrát, třetí cyklus čtyřikrát atd.).

3 Experimenty

Testování proběhlo na systému *merlin*. Čas byl měřen od konce distribuce hran až po ukončení *suffix sum* (tak, aby složitost odpovídala teoreticky popsané složitosti výše), přičemž pro každý počet prvků bylo provedeno pět měření, která byla do výsledného grafu zprůměrována. K měření času byl použit rozdíl vestavěné funkce knihovny *MPI* - `MPI_Wtime()`.

Očekávanou logaritmickou složitost se mi nepodařilo naměřit, pravděpodobně kvůli vytížení systému *merlin* a počtu fyzických jader. Každopádně od určitého bodu graf logaritmus trochu připomínat začal.



4 Komunikační protokol

Komunikační protokol mezi procesory znázorňuje níže přiložený sekvenční diagram, který ukazuje obecnou komunikaci mezi procesory v průběhu algoritmu *suffix sum*. Tedy pokud určitý procesor požaduje váhu svého následníka, zašle mu požadavek (zároveň se speciálním flagem *doNotListen*, který říká, zda má daný procesor další cyklus naslouchat svému předchůdci ohledně své váhy) a očekává odpověď v podobě váhy a dalšího následníka.

5 Závěr

Jak již bylo zmíněno, výsledky algoritmu úplně nedokazují jeho logaritmickou časovou složitost, pravděpodobně díky fyzickým vlastnostem testovacího systému a zároveň jeho vytížení. Teoreticky by však má implementace logaritmické časové složitosti dosahovat měla.