# Dokumentácia k projektu z predmetu PRL : *Algoritmus viditeľnosti*

Anton Firc (xfirca00)

19. apríla 2020

## 1 Úvod

Cieľom projektu je implementácia algoritmu viditeľnosti použitím knižnice Open MPI a zmerať a odôvodniť jeho časovú zložitosť.

## 2 Rozbor a analýza algoritmu

Algoritmus viditeľnosť je paralelný algoritmus pre zistenie viditeľnosti výškových bodov na priamke z určitého miesta pozorovateľa.

Algoritmus najprv distribuuje N vstupných hodnôt medzi M procesorov kde ($M \leq N$). Každý procesor obdrží práve dve hodnoty ktoré v ďalšom uvažovaní reprezentujú samostatné procesy. Distribúcia hodnôt je vykonaná postupne podľa ID bežiacich procesov. Po vyčerpaní vstupných hodnôt sú procesom distribuované hodnoty `NO_VAL` alebo `INACTIVE_CPU` postupne značiace, že proces nedostal žiadnu hodnotu, prípadne, že procesor nemusí byť aktívny. Význam a spôsob prideľovania týchto hodnôt je popísaný ďalej v časti zaoberajúcej sa implementáciou.

Po distribúcii hodnôt sú zastavené nepotrebné procesy a následne každý proces vypočíta uhol odpovedajúci výškovému bodu jemu priradenému. Po výpočte uhlov, nadchádza fáza *max-prescan* ktorá sa skladá z troch častí:

1. *upSweep* V tejto časti dochádza v imaginárnej stromovej štruktúre k výmene hodnôt tak, že rodičovský uzol vždy obdrží väčšiu z hodnôt jeho synovských uzlov. Po ukončení tohoto kroku sa v pomyselnom koreňovom uzle nachádza maximum z hodnôt zo všetkých procesov.

2. *clear* Vloží do pomyselného koreňového uzla neutrálnu hodnotu. V tomto prípade sme za neutrálnu hodnotu zvolili konštantu `-DBL_MAX` ktorá značí najmenšie možné číslo reprezentovateľné typom `double`. Takto je zabezpečené, že pri porovnaní neutrálnej hodnoty s ľubovoľnou hodnotou vždy dostaneme druhú hodnotu.

3. *downSweep* V poslednej časti dochádza k distribúcii doposiaľ najväčšej dosiahnutej hodnoty každému procesu. Táto distribúcia je dosiahnutá v $log_2(N)$ krokoch kde si pomyselné rodičovské uzly vymieňajú hodnoty so synovskými uzlami. Rodičovský uzol si ponechá väčšiu z hodnôt, synovský uzol obdrží hodnotu z rodičovského uzlu.

Po ukončení fázy *max-prescan* vypočíta každý proces, či je jemu pridelený bod viditeľný. Dochádza k porovnaniu vypočítaného uhlu bodu a najväčšieho doposiaľ videného uhlu získaného pomocou *max-prescan*. Po výpočte viditeľnosti dochádza k distribúcii viditeľností do procesu s ID 0 ktorý následne obsluhuje finálny výpis.

### 2.1 Zložitosť

Asymptotická časová zložitosť algoritmu viditeľnosti je určená ako $O(n/N + log(N))$, kde $n$ značí počet bodov a $N$ počet procesorov. Výpočet uhlov a viditeľnosti je vykonávaný paralelne na všetkých procesoroch, čo odpovedá zložitosti $n/N$. Obe časti *upSweep* a *downSweep* prebiehajú v maximálne $log(N)$ krokoch, čo odpovedá zložitosti $log(N)$. Takže jednoduchým sčítaním týchto zložitostí dostávame zložitosť $n/N + log(N)$, ktorú následne overíme experimentami.

# 3 Implementácia

Algoritmus je implementovaný v jazyku C++, s využitím knižnice Open MPI podporujúcej paralelné výpočty. Beh programu začína inicalizáciou knižnice Open MPI volaním funkcie `MPI_Init()`. Následne sú prostredníctvom funkcií `MPI_Comm_size()` a `MPI_Comm_rank()` získané počet dostupných (bežiacich) procesov a vlastný identifikátor (ID) každého procesu. Hlavný (riadiaci) proces má ID 0, pre ostatné procesy je ID nenulové kladné celé číslo. Procesor s ID 0, následne načíta vstup zo *STDIN* volaním funkcie `loadData()`, vypočíta počet procesov potrebných pre zostrojenie binárneho stromu a každému procesu odošle jemu priradený bod volaním funkcie `MPI_Send()`. Každý fyzický procesor v našom poňatí reprezentuje 2 procesy tvoriace uzly binárneho stromu. Počet procesov potrebných pre zostrojenie binárneho stromu pre zadaný počet vstupných hodnôt je teda $2^{log_2(n)}$ kde hodnota logaritmu je zaokrúhlená nahor. Procesom ktoré sú aktívne, ale už pre ne nie je priraditeľná vstupná hodnota odošle hodnotu `NO_VAL`. Procesom ktoré sú mimo rozsah potrebný pre zostrojenie binárneho stromu odošle hodnotu `INACTIVE_CPU`. Okrem hodnôt bodov sú fyzickým procesorom ešte odoslané výška bodu pozorovateľa a počet aktívnych procesov.

Po ukončení distribúcie si každý proces prevezme jemu priradené hodnoty. Hodnota v premennej `point1` reprezentuje prvý a hodnota `point2` druhý pomyselný proces bežiaci na jednom fyzickom procesore. Procesory ktoré obdržali ako hodnoty `point1` a `point2` hodnotu `INACTIVE_CPU` sú ukončené. Následne dochádza k výpočtu uhlu medzi bodom pozorovateľa a výškou daného bodu pre každý proces a tieto hodnoty sú uložené do premenných `angleP1` a `angleP2`. Proces s ID 0 si do premennej `angleP1` uloží hodnotu `-DBL_MAX` nakoľko nemá zmysel vypočítavať uhol pre miesto pozorovateľa. Sú vytvorené premenné `maxPrevAngle1 = angleP1` a `maxPrevAngle2 = angleP2` Následne je volaná funkcia `upSweep()` ktorej sú predané ukazatele na premenné `maxPrevAngle1` a `maxPrevAngle2`. Táto funkcia rekurzívne porovnáva hodnoty uhlov pomyselného ľavého syna a rodičovského uzlu zlúčeného s pravým synom a vloží do rodičovského uzlu väčšiu z týchto hodnôt. Funkcia končí keď zostávajú aktívne len dva procesory, a vtedy vloží do koreňového uzlu, teda posledného procesoru do premennej `maxPrevAngle2` najväčšiu zo všetkých hodnôt. Ďalej pokračuje beh programu volaním funkcie `clear()` ktorá do koreňového uzlu pomyselného binárneho stromu vloží neutrálny prvok, teda `-DBL_MAX` v našom prípade. Posledne je volaná funkcia `downSweep()` ktorá rozdistribuuje medzi všetky procesy doposiaľ najväčší pozorovaný uhol. Tel funkcie tvorí for cyklus bežiaci maximálne $log_2(N)$ kde N značí počet aktívnych procesov. V každom behu si ľavý syn vymení hodnotu s rodičovským uzlom zlúčeným s pravým synom. Rodičovský uzol si ponechá väčšiu z hodnôt jeho alebo jeho ľavého syna. Ľavý syn obdrží hodnotu rodiča. Či je uzol v danom opakovaní rodič alebo syn rozhodujú funkcie `parentNode()` a `childNode()` ktoré na základe ID procesoru a aktuálnej iterácie rozhodujú či sa jedná o rodiča, syna alebo ani jednu z týchto možností.

Po skončení funkcií reprezentujúcich fázu *max-prescan* vypočíta každý proces, či je jemu priradený bod viditeľný. Výpočet prebieha porovnaním uhlu bodu priradeného procesu `angleP1/2` a doposiaľ najväčšiemu videnému uhlu `maxPrevAngle1/2`. Vypočítaná viditeľnosť je následne odoslaná procesu s ID 0.

Procesor s ID 0, najprv vloží do vektoru výslednej viditeľnosti svoje hodnoty a následne si prevzeme hodnoty od všetkých ostatných procesorov. Hodnoty následne vypíše na *STDOUT*. Posledne každý procesor volá funkciu `MPI_Finalize()` ktorá ukončí prostredie paralelného behu a skončí úspechom.

# 4 Experimenty

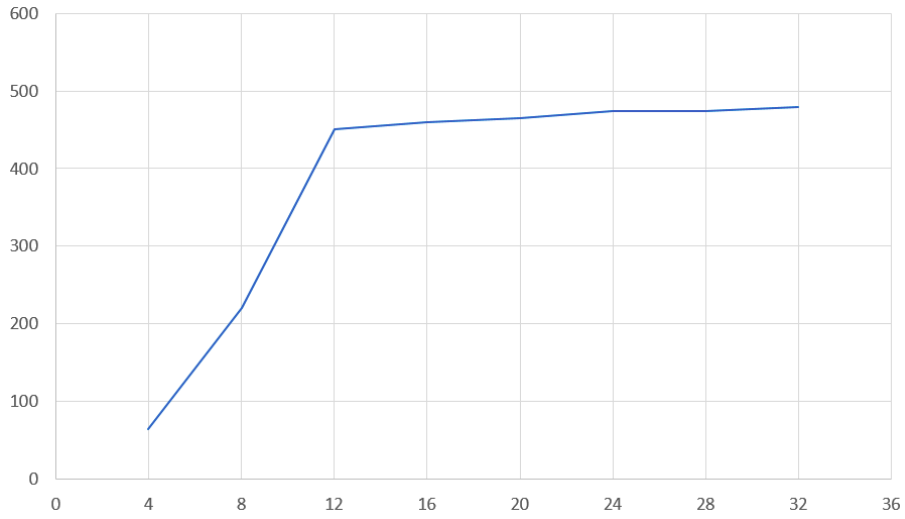
Implementácia algoritmu bola podrobená experimentom s rôzne veľkými vstupmi za účelom overenia teoretickej časovej zložitosti. Experimenty prebiehali na superpočítači *Anselm*, so vstupmi o veľkosti 4, 8, 12, 16, 20, 24, 28 a 32 hodnôt. Pre vstupy každej veľkosti bol meraný čas behu algoritmu.

Meranie času behu algoritmu prebiehalo použitím C++ knižnice `chrono` a bariéry z knižnice `OpenMPI`. Bariéra zaisťovala, že meranie času bolo spustené a ukončené práve vtedy keď sa na dané miesto v programe dostali všetky procesy naraz. Zaznamenanie času začiatku prebiehalo tesne pred výpočtom uhlov. Zaznamenanie času konca zase prebiehalo okamžite po výpočte viditeľnosti

jednotlivých bodov. Po výpočte dĺžky behu algoritmu bola táto hodnota vypísaná na *STDOUT* v $\mu s$.

Meranie pre každú veľkosť vstupu prebehlo 100 krát a do grafu bola následne zaznamenaná priemerná nameraná hodnota. Ako môžeme z grafu vidieť, je potvrdená logaritmická časová zložitosť. Takisto vidíme, že najväčšie rozdiely spôsobuje počet potrebných procesorov pre výpočet(4,8,16). Zvyšovanie počtu hodnôt pri rovnakom počte procesorov (20 - 32) nemá zásadný vplyv na čas.

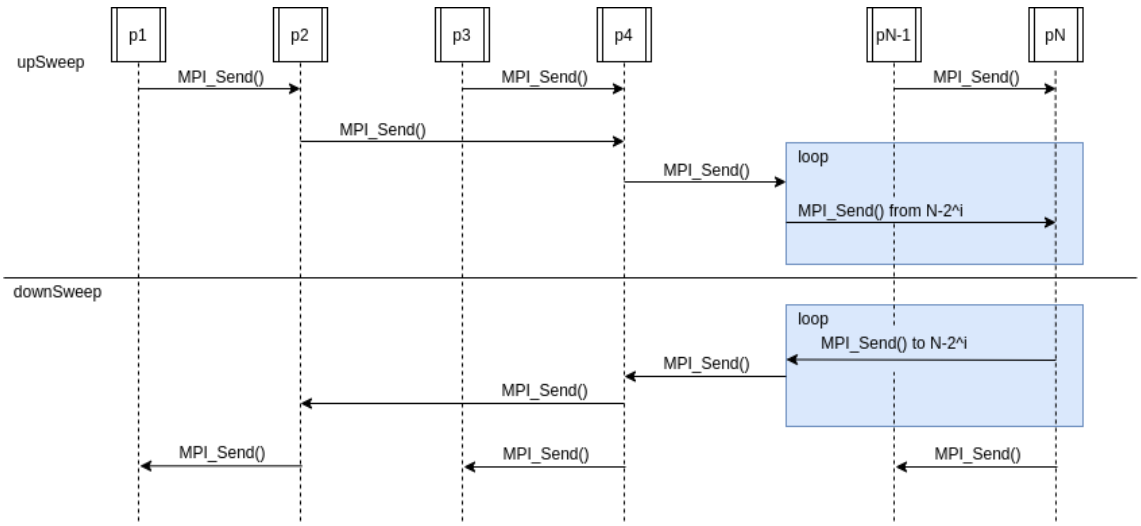
Režim merania výkonu je možné v programe zapnúť definovaním makra `BENCHMARK`.




Obr. 1: Graf znázorňujúci priemernú dĺžku behu algoritmu($\mu s$) v závislosti na veľkosti vstupu

# 5 Komunikačný protokol

Komunikačný protokol obsahuje iba fázy algoritmu *upSweep* a *downSweep*. Ostatné časti buď totižto priamo nie sú súčasťou algoritmu ako napríklad distibúcia hodnôt alebo ich spätné získanie, alebo tieto časti neobsahujú žiadnu medziporcesorovú komunikáciu ako napríklad výpočet uhlov, viditeľnosti alebo fáza *clear*.




Obr. 2: Sekvenčný diagram komunikácie medzi procesmi

# 6 Záver

Experimenty so vstupmi rôznych veľkostí potvrdili teoretickú zložitosť algoritmu. Z grafov je vidno, že čas behu algoritmu logaritmicky rastie s počtom vstupných hodnôt. Mierne odchýlky v nameraných hodnotách mohli byť spôsobené číslom použitých procesorov ktoré bolo získané prepočtom dĺžky vstupu na mocniny dvojky.