# Rozbor

V projektu byl implementován program na řešení problému viditelnosti (line of sight problem). Algoritmus pracuje tak, že každému procesu je přiřazena dvojice hodnot, se kterou pracuje. Teoretická složitost algoritmu se odvíjí od několika faktorů: Každý procesor nejprve vypočte pro oba své listy velikost úhlu mezi daným bodem a pozorovatelem. Velikost úhlu lze získat jako: $actan((vyskaBodu - vyskaPozorovatele)/index)$. Jelikož se jedná o dva jednoduché výpočty a procesory je počítají paralelně, je složitost tohoto kroku konstantní – $O(c)$, kde $c$ je konstanta odpovídající složitosti daného výpočtu.

Následuje výpočet operace maxPrescan nad hodnotami úhlů. Zde je nejprve proveden průchod vzhůru (upsweep), hodnota posledního prvku (kořenu) je nastavena na neutrální prvek a poté je proveden průchod dolů (downsweep). Počet iterací v obou průchodech se odvíjí od počtu úrovní binárnáho stromu tvořeného procesy, obecně tedy $2 \times O(log_2(n)) + O(c)$ pro oba průchody. Po vypočtení maxPrescan každý z procesů porovná úhel svých listů s nejvyšším předcházejícím úhlem – $O(c)$ výsledky jsou vytisknuty.

Z analýzy vyplývá, že očekávaná třída složitosti je $O(log(n))$ a počet procesorů pro výpočet je roven polovině délky vstupu. Výsledné složitosti obecného algoritmu jsou tedy následující:

$$t(n) = O(log_2(n)) \tag{1}$$

$$p(n) = n/2 \tag{2}$$

$$c(n) = t(n) * p(n) = log_2(n) * (n/2) = O(n * log_2(n)) \tag{3}$$

Cena $c(n) = O(n * log_2(n))$ není optimální výsledek, protože optimální sekvenční algoritmus má složitost lineární, tedy lepší, než tento algoritmus.

Výše zmíněné výpočty složitosti platí v případě, že máme k dispozici dostatečně velký počet procesorů (což je i případ v tomto projektu). Pokud by dostupný počet procesorů byl nižší ($N$ fyzických procesorů), rozdělila by se vstupní sekvence výšek na několik částí o délce $n/N$ a každé této sekvenci by byl přiřazen procesor, který by optimálním sekvenčním algoritmem provedl operaci prescan a výsledky by se zpracovaly pomocí paralelní operace prescan. Časová složitost by se pak skládala ze dvou komponent – sekvenční algoritmus a paralelní algoritmus, tedy $t(n) = O(n/N + log_2(n))$. V případě, že by si obě komponenty složitosti byly rovny, byla by cena algoritmu optimální, tedy $O(n)$.

# Implementace

Pro samotný program byl vytvořen spouštěcí skript (test.sh), který z velikosti vstupu odvodí počet procesorů jako polovinu následující mocninu dvou délky vstupu ($p = dalsiMocnina2(n)/2$) a program spustí s tímto argumentem.

V programu je nejprve zpracován vstupní řetězec. Zpracování provádí proces číslo 0 a zasílá ostatním procesům jejich hodnoty. Jelikož velikost vstupu nemusí odpovídat mocnině dvou, je pro daný vstup zjištěna nejbližší následující mocnina dvou a pole vstupů je rozšířeno na tuto velikost s hodnotami výšky 0.

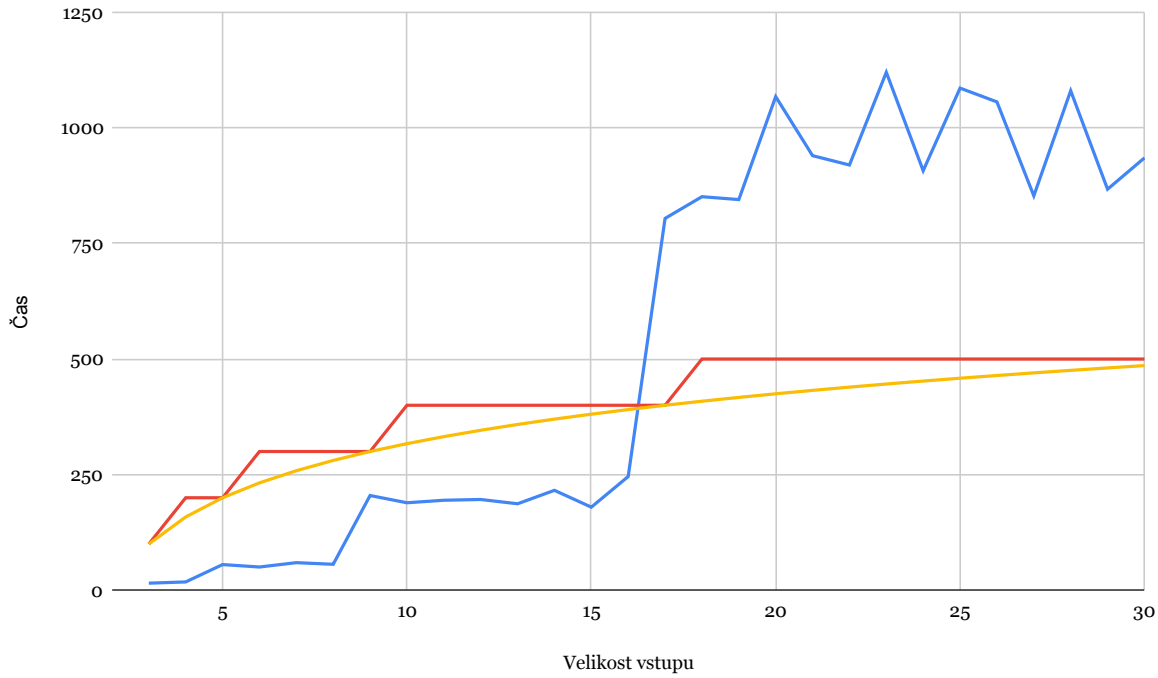
Proces 0 dále rozešle informace o celkové velikosti vstupu a výšce pozorovatele. Po vypočtení úhlů svých listů se počítají procesy hodnoty maxPrescan. Zde byla jako neutrální prvek zvolena hodnota $-1.57079633$ $(-\pi/2)$, což zhruba odpovídá velikosti úhlu, který by směřoval od pozorovatele přímo dolů. Při hledání maximálního prvku se tak jedná o neutrální prvek. Při samotném výpočtu maxPrescan je synchronizace zaručena předáváním zpráv s hodnotami, nemůže se tak stát, že by proces přistoupil do paměti předčasně.

Následně každý z procesů porovná zjištěný úhel s maximálním předcházejícím úhlem a zašle procesu 0 informaci o viditelnosti. Proces 0 se pak také stará o výstup programu. Celá výpočetní sekvence je zobrazena v diagramu 2 na konci dokumentu.

# Měření

Měření algoritmu proběhlo na severu Merlin. Program byl modifikován tak, aby pomocí funkcí `high_resolution_score::now()` z knihovny `<chrono>` zaznamenal čas před a za algoritmem, konkrétně před sekcí `downsweep` a za sekcí `upsweep`. Před začátkem měření byly procesy synchronizovány pomocí `MPI_Barrier()`.

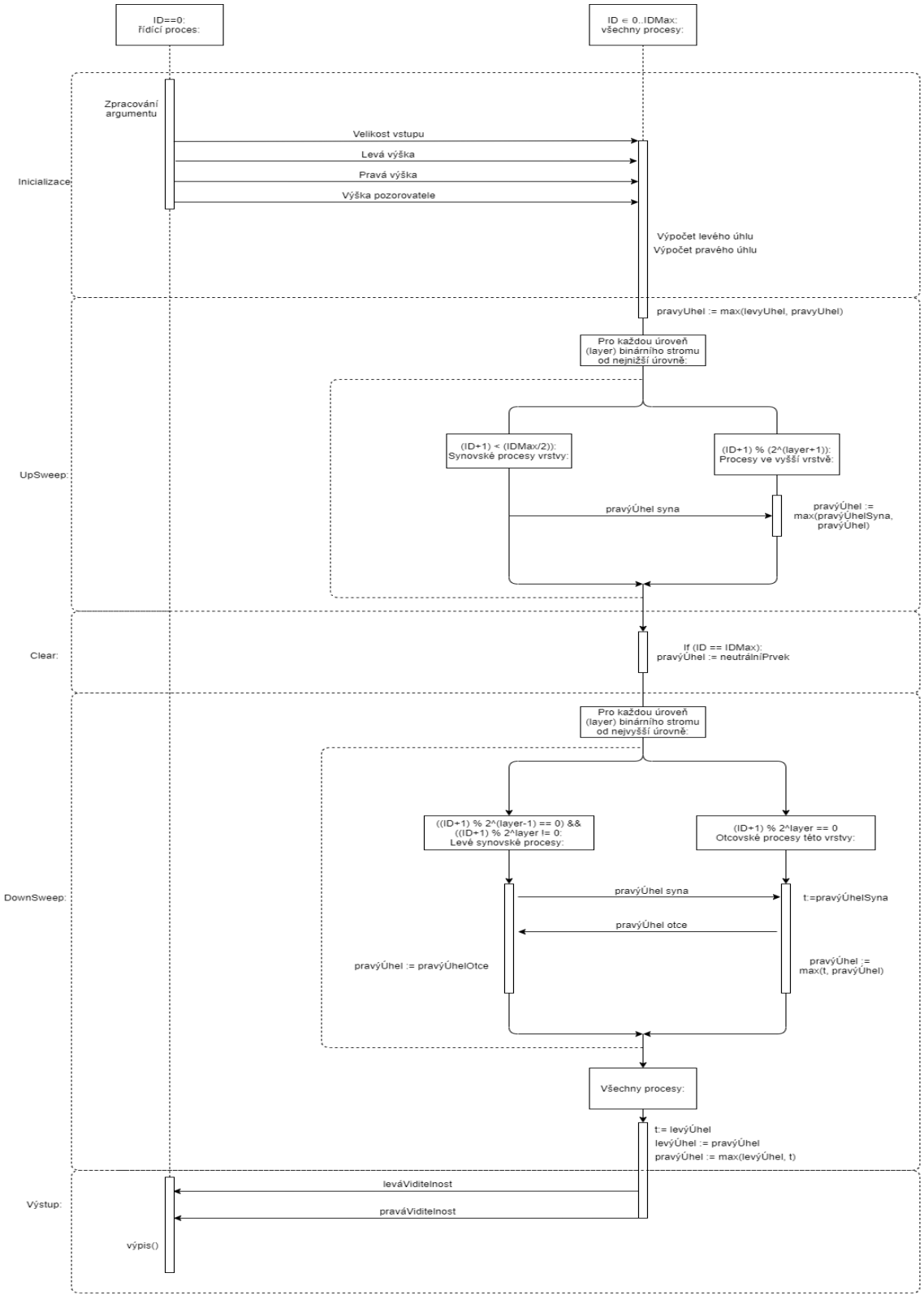
Samotné měření bylo provedeno nad vstupy o velikosti 2 až 30 výšek, pro každou délku vstupu bylo měření opakováno padesátkrát. Předpokládaným výsledkem byla schodová logaritmická závislost času na velikosti vstupu. Výsledky jsou zobrazeny v grafu 1.




Obrázek 1: Závislost času výpočtu (v milisekundách) na velikosti vstupu. Modře jsou zobrazeny naměřené hodnoty, červeně jsou zobrazeny očekávané výsledky pro tuto implementaci – schodově logaritmická závislost, žlutě pak čistě logaritmická závislost odhadovaná v rozboru algoritmu. Schodovitost grafu je způsobená rozšířením pole vstupu na následující mocninu dvou.

# Výsledky a závěr

Z naměřených dat vyplývá, že při každém zdvojnásobení délky vstupu je spotřebován na výpočet asi trojnásobný počet času, což neodpovídá logaritmické časové složitosti algoritmu. Předpokládaná časová složitost algoritmu se tedy testováním nepotvrdila. Na vině může být zaneprázdnění serveru Merlin. Nepřesnosti mohl do měření také zanést fakt, že režie při přepínání procesorů v tomto případě příliš zpomaluje daný algoritmus. Je velmi pravděpodobné, že při testování s větším počtem dostupných fyzických/virtuálních procesorů a s větším vstupem by se potvrdila očekávaná složitost.




Obrázek 2: Diagram výpočty programu. Jednotlivé procesy (ID) jsou číslovány od nuly, každý pracuje se dvěma hodnotami (levýÚhel, pravýÚhel). Hodnota IDMax odpovídá indexu posledního procesoru. Zprávy v sekci Inicializace jsou vždy zasílány od procesu s ID 0 všem ostatním procesům, v sekci Výstup pak opačně. V ostatních sekcích jsou pak zprávy vždy poslány mezi synovskými a otcovskými procesy dané vrstvy.