

# Viditeľnosť

Matej Karas

## Rozbor

Algoritmus pracuje nad vektorom prvkov, ktoré reprezentujú nadmorskú výšku v teréne a počiatočný bod pozorovateľa, resp. jeho nadmorskú výšku.

Algoritmus má 2 kroky:

1. Vytvorenie vektoru uhlov z vektoru výšok.
2. Pomocou prefixového max scanu sa vytvorí vektor maximálnych uhlov, ktoré sa budú následne porovnávať s výškami.

### Teoretická zložitosť:

Časová	$O(n/N + \log N)$
Priestorová	$O(2n) = O(n)$
Celková cena	$O(n^2/N + n \log N)$

Časová zložitosť je odvodená z faktu, že fáza algoritmus *up-sweep* potrebuje  $n - 1$  krokov a dá sa paralelizovať  $N$  procesormi.

Priestorová zložitosť je odvodená z faktu, že sa musí uchovať vektor výšok, resp. uhlov a zároveň vektor prefixových maxím. V asymptotickej zložitosti je to teda  $O(n)$ .

Celková cena je potom *priestorová* \* *časová* zložitosť a teda  $O(n^2/N + n \log N)$ . Avšak pri maximálnom paralelizme sa bude táto cena rovnať  $O(n)$ , a teda cena algoritmu bude optimálna.

## Implementácia

Úlohu som implementoval v jazyku C++17. Skript *test.sh* alokuje toľko procesov, koľko je dostupných fyzických jadier (nepoužitie parametru *-np* s predpokladom, že daný systém má správne nastavené OpenMPI).

Na začiatku vytvorí tzv. *root* proces zo vstupného reťazca pole floatov (výšok) a rozošle počiatočnú výšku všetkým procesom funkciou **MPI\_Broadcast**. Jednotlivé časti poľa výšok následne rozošle pomocou funkcie **MPI\_Scatterv** všetkým procesom, ktoré si ho stransformujú na pole uhlov.

Následne sa vykoná fáza down-sweep, tak ako bola popísaná na prednáškach, s tým rozdielom, že v mojej implementácii je ošetrovaná, aby fungovala aj na množstvách prvkov, ktoré nie sú mocnina dvojky. Poslední kroky fázy sa vykonávajú zasielaním správ.

Potom nasleduje fáza down-sweep, ktorá je taktiež ošetrovaná, aby fungovala správne s ľubovoľným počtom prvkov.

Následne prebehne samotný algoritmus viditeľnosti, kde sa porovnávajú uhly jednotlivých výšok s výsledkom prefixovej sumy.

Výsledky jednotlivých procesov sú zozbierané *root* procesom pomocou funkcie **MPI\_Gatherv** a následne vypísané na štandardný výstup.

Pre účely testovania je pridané makro **BENCHMARK**, ktorého nastavením v skripte *test.sh* sa preloží variant s meraním času, v ktorej sa výšky generujú v programe funkciou **generateNumbers**.

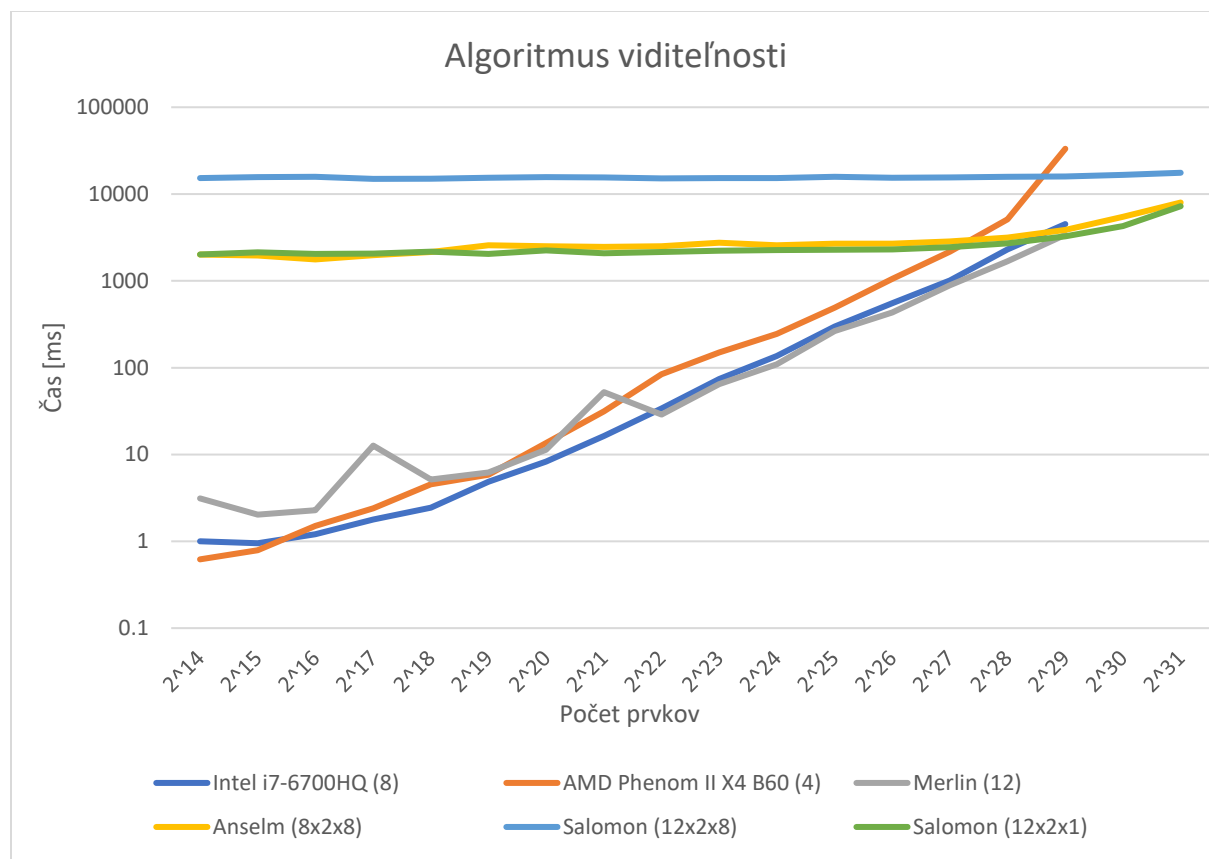
## Experimenty

Experimenty boli uskutočnené na piatich typoch strojov, z toho 2 boli na Ostravskom superpočítači Salomon a Anselm. Program bol testovaný na veľkom vstupe (až po plné využitie dostupnej pamäte RAM).

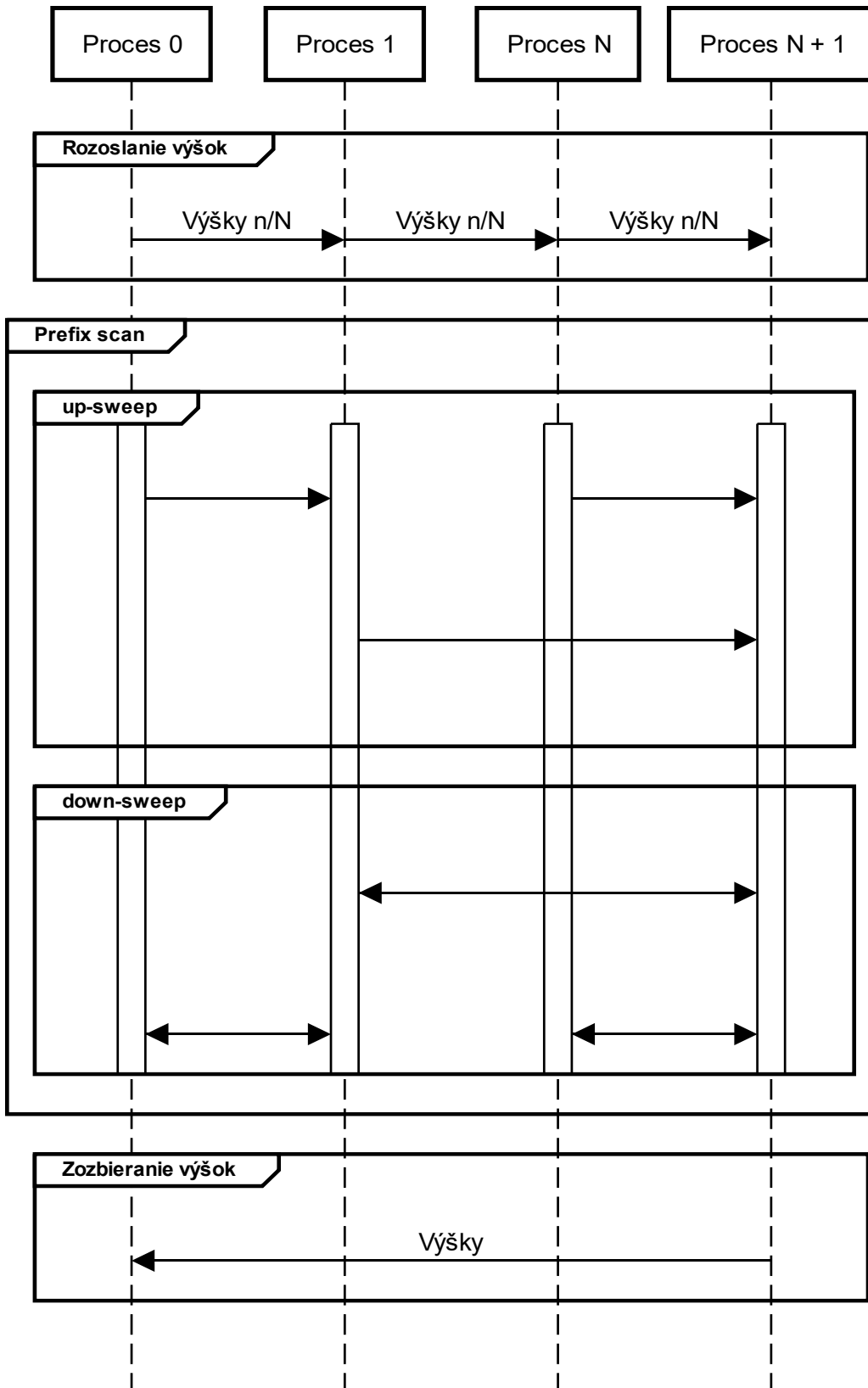
Z grafu vidieť, že testovanie na školskom serveri Merlin nie je smerodajné, pretože jeho nepredikovateľná záťaž ovplyvňuje výsledky.

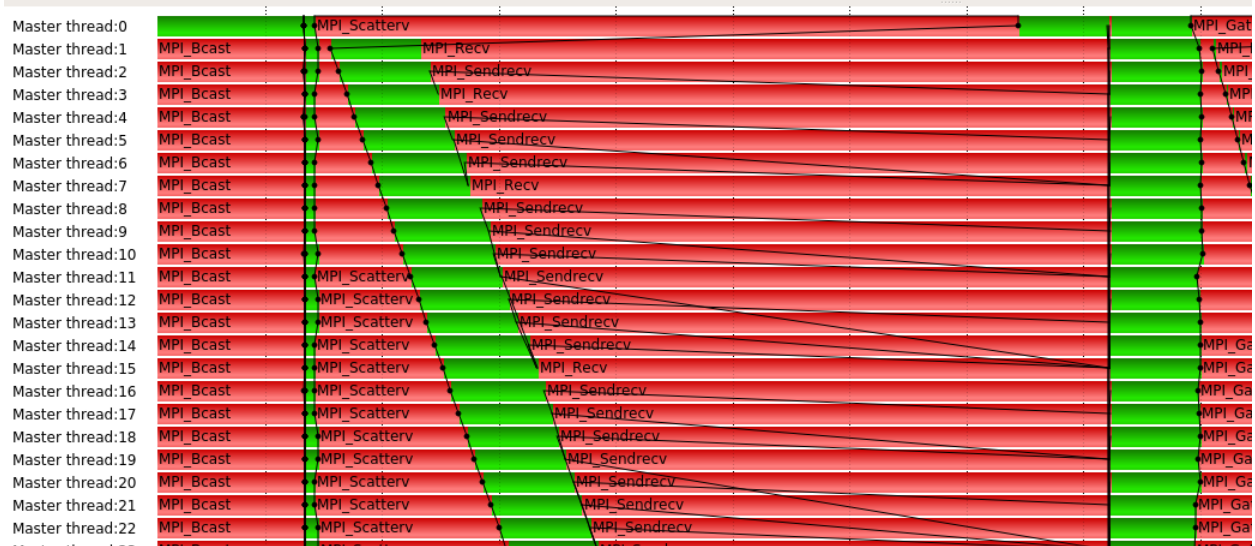
Taktiež, tým že algoritmus nie je vôbec výpočetne náročný je jeho použitie na superpočítači celkom neefektívne – komunikácia medzi jednotlivými uzlami je príliš zdĺhavá, čo je vidieť na grafe (Salomon so 8 uzlami vs. Salomon s 1 uzlom), resp. Obrázok 1.

Pridaním procesorov (v UMA doméne), sa výpočet algoritmu urýchli, pretože zasielanie správ pomocou zdieľanej pamäte je efektívne.



## Komunikačný protokol





Obrázok 1 - Komunikačný graf, pri použití 8 uzlov (192 jadier).

## Záver

Experimentami sa potvrdila lineárna časová zložitosť algoritmu. Algoritmus je neefektívny na clustri, pretože algoritmus je výpočetne nenáročný a komunikácia medzi jednotlivými uzlami je príliš drahá.