Paralelní a distribuované algoritmy
3. projekt

Viditelnost

Peter Horňák
xhorna14@stud.fit.vutbr.cz

1. Analýza algoritmu

Algoritmus na základe matice terénu, v podobe nadmorských výšiek vyhodnocuje, či sú jednotlivé body matice viditeľné z pozorovacieho bodu. Bod je viditeľný z pozorovacieho bodu, práve, vtedy, ak žiadny bod medzi nimi nemá väčší vertikálny uhol. Pozostáva z 3 fáz:

1. Výpočet vertikálneho uhlu pre každý bod v pozorovacej línií.
2. Operácia Reduce.
3. Operácia DownSweep.

Časová zložitosť algoritmu môžeme vyjadriť ako sumu operácií Reduce a DownSweep. Reduce má časovú zložitosť $t(n) = \mathcal{O}(\log_2 n)$, keďže algoritmus vykoná paralelne operácie na každej úrovni stromu. To však v prípade, že máme pre každý pár prvkov na vstupe vólny procesor. V opačnom prípade, každý procesor vykoná sekvenčnú operáciu Reduce pre svoju časť o dĺžke n/N. Z toho vyplýva zložitosť $t(n) = \mathcal{O}(n/N)$.

Pamäťová zložitosť algoritmu v prípade, že N>=n je $p(n) = \mathcal{O}(n/2)$ pretože pre každý pár čísel, je potrebný jeden procesor. Inak je možné použiť maximálny možný počet procesorov teda $p(n) = \mathcal{O}(N)$.

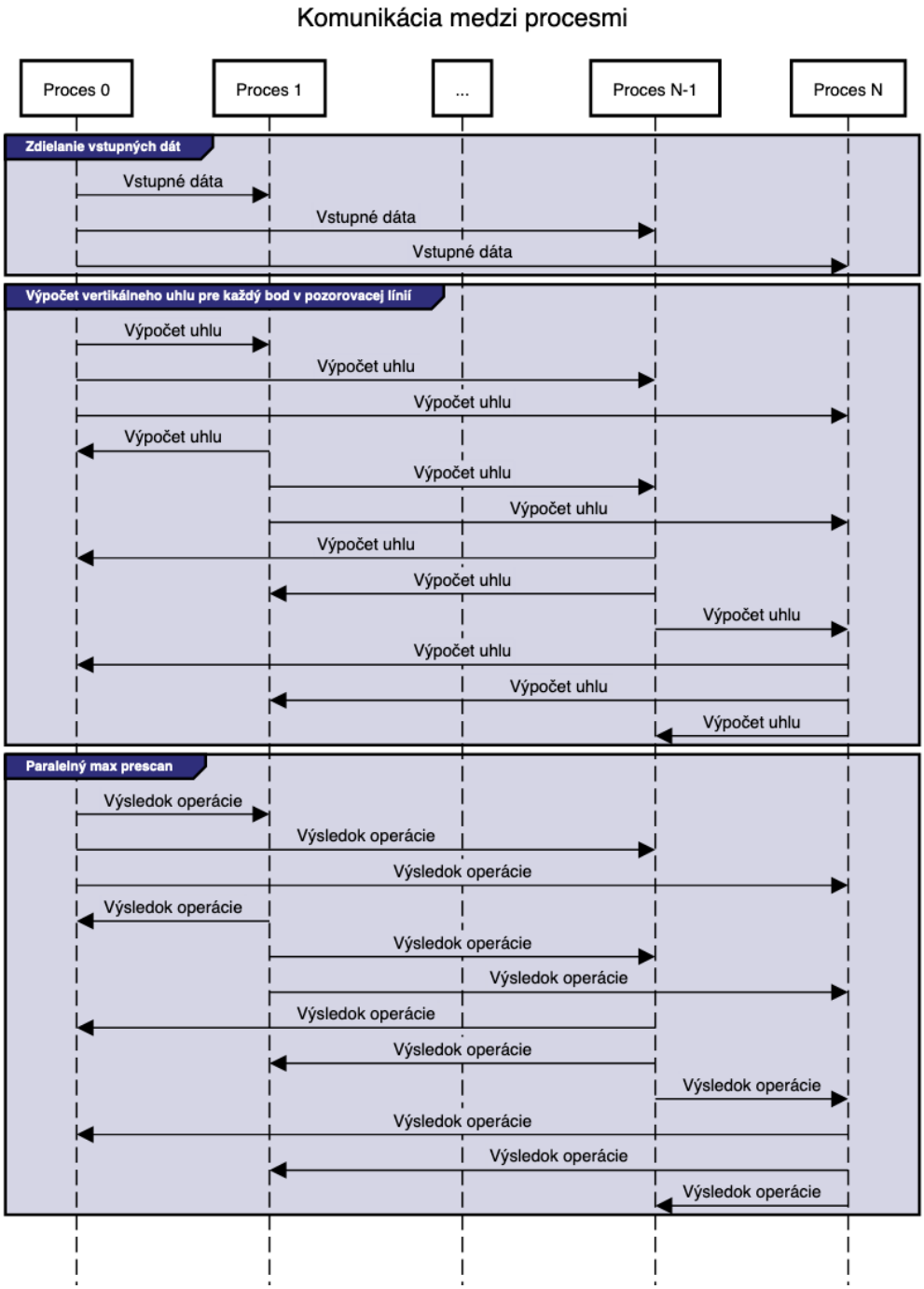
Výslednú cenu teda môžeme odvodiť nasledovne:

- Ak N>=n výsledná cena $p(n) = \mathcal{O}(\log_2 n) * n/2 = \mathcal{O}(n * \log_2 n)$. Čo nie je je optimálna cena v porovnaní s sekvenčným algoritmom.
- Inak je výsledná cena $p(n) = \mathcal{O}(n/N) * N = \mathcal{O}(n)$. Táto cena je rovnaká ako v sekvenčnom algoritme a preto hu môžeme označiť za optimálnu.

2. Implementácia

Projekt je implementovaný v jazyku C++. Pre jeho správne spustenie je k zdrojovému kódu priložený skript *test.sh*, ktorý prijíma jeden povinný parameter a to reťazec reálnych čísiel oddelených znakom ‘,’. Tento reťazec je uložený do súboru *numbers*, ktorý je následne presmerovaný do súboru. Následne sa spustí program, kde proces s id=0, načíta vstup z *stdin* a následne rozpošle všetkým procesom tieto čísla uložené ako pole pomocou funckie *MPI_Bcast()*. Následne každý proces vypočíta vertikálny uhol medzi pozorovacím bodom. Následne prebehne operácia Reduce, kde každý dostupný proces si zoberie nasledujúci pár čísiel a vykoná túto operáciu. Počas toho procesy medzi

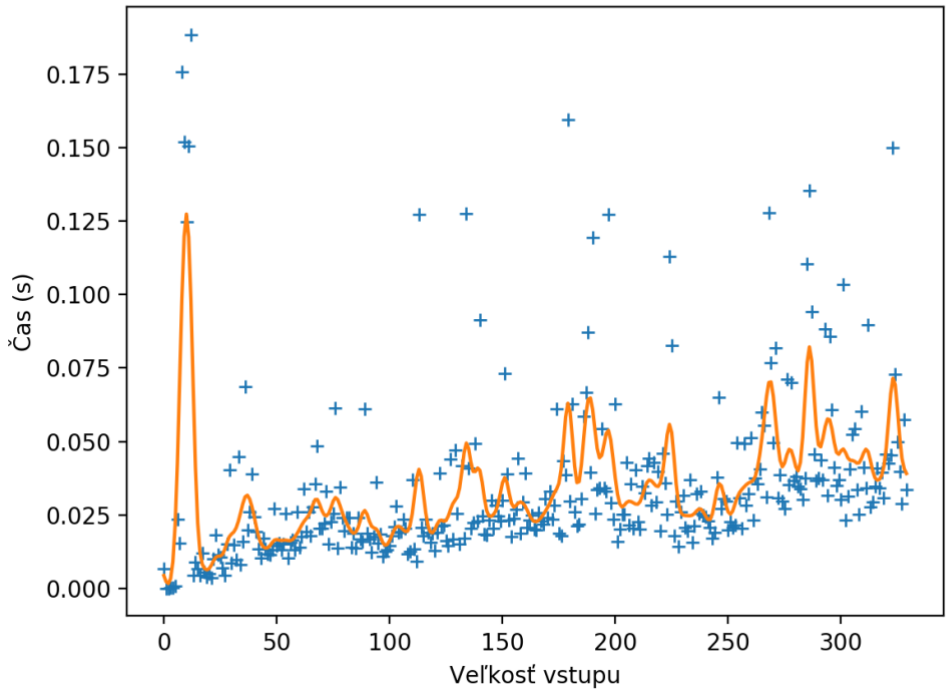
sebou zdieľajú, výpočty pomocou funkcie *MPI_Bcast()*. Rovnaký postup sa následne vykoná aj pre operáciu DownSweep. Nakoniec proces s id=0 vypíše na *stdout* výstup programu.



Obr. 1: Sekvenčný diagram komunikácie

3. Experimenty

Experimenty boli vykonávané na školskom servery Merlin. Na meranie času bola použitá štandardná knižnica C++ *std::chrono*, konkrétne funkcia *std::chrono::high_resolution_clock::now()*. V grafe je možné vidieť presné výsledky a krivku vytvorenú pomocou techniky *Kernel regression*.



Obr. 2: Výsledky experimentov.

4. Záver

Z experimentov je vidieť, že časová zložitosť s jemnými odchýlkami potvrdzuje, že časový nárast sa zhoduje s našou teoretickou analýzou. V prvých 50 vstupoch, sa používalo n/2 procesov, čo vyjadruje logaritmický nárast. Ďalej, museli niektoré procesy obsluhovať viac vstupov sériovo. Odchýlky, vyjadrujú vyťaženie serveru Merlin, réžia medzi procesovej komunikácie a prepínania procesov.