PRL – Viditelnost

Alena Tesařová (xtesar36), duben 2020

1 Úvod

Zadáním projektu bylo naimplementovat a zdokumentovat algoritmus **Viditelnost** tak, jak je uveden na přednášce z PRL[1]. Problém spočívá v tom, že je zadané pole nadmořských výšek a pozorovací bod X (místo pozorovatele) a úkolem je zjistit, které body podél paprsku vycházejícího z místa X jsou viditelné. V sekci 2 bude podrobněji zanalyzována časová složitost algoritmu. Popis implementace se nachází v sekci 3. Pro lepší názornost obsahuje zpráva i komunikační prokol (sekce 5) a průběh časové složitosti pro různě velké vstupy (sekce 4).

2 Rozbor a analýza algoritmu

Na vstupu se nachází pole nadmořských výšek a pozorovací bod X v tomto formátu $[X, a^0, a^1, a^2, ...a^n]$, kde $a^0..a^n$ jsou nadmořské výšky terénu. Platí, že bod je viditelný, pokud žádný bod mezi pozorovatelem a jím nemá větší vertikální úhel. Řešení problému obsahuje 3 fáze:

1. vytvoří se vektor výšek bodů podél pozorovacího paprsku
2. vektor výšek se přepočítá na vektor úhlů
3. pomocí **max_prescan** se spočte vektor maximálních úhlů, pro zjištění viditelnosti bodu pak stačí určit jeho úhel a porovnat ho s maximem

Pojďme nyní rozebrat časovou složitost jednotlivých kroků algoritmu. V prvním kroku se vytvoří vektor výšek bodů – tento krok umíme vykonat v konstantním čase – v případě, že každý proces provede načtení jednoho prvku, stejně tak i druhý krok vytvoření vektoru úhlů (každý proces provede jeden výpočet). Co se týká sumy prefixů, tam je to trochu složitější. Pokud máme po ruce dostatek procesorů a nezáleží nám na ceně algoritmu, tak bude mít tato část logaritmickou složitost, tzn. $t(n) = O(logn)$. Důvodem je implementace pomocí binárního stromu, kde má strom výšku log(n), pak pro každý pár prvků je nutný 1 procesor ($p(n) = n/2$), tedy cena bude $c(n) = O(n * log(n))$. Cena není v tomto případě optimální, jelikož sekvenční algoritmus měl složitost lineární. Cenu je možné vylepšit tím, že ubereme na počtu procesorů. V tomto případě by se jeden proces starou o skupinu prvků daného počtu a na ně aplikoval operaci sekvenčně (optimálním sekvenčním algoritmem). Výsledky jednotlivých skupin by se pak spočítaly paralelně. V tomto případě bude mít algoritmus optimální cenu, ale zhorší se časová složitost na lineární $O(n/N + logN)$. V mé implementace byl použit strom s počtem procesů zarovnaný na mocniny dvojky a časová složitost je logaritmická.

3 Implementace

Implementace vychází z pseudo-algoritmu probíraném na přednášce PRL. Počet procesů je vypočítán v pomocném skriptu test, který je přiložen k řešení. Na základě informací z přednášky a ze zadání byl zvolen počet procesorů zarovnaný na mocniny dvojky vydělený dvěma. Například dvěma prvkům na vstupu bude odpovídat 1 procesor, čtyřem prvkům dva procesory, pěti – čtyři procesory (5 zarovnáno na mocninu dvou je 8 a polovina z 8 jsou 4) atd. Pokud by byl zvolen stejný počet procesorů jako prvků, může se stát, že bude zadáno více jak 30 nadmořských výšek a systém na referenčním serveru (merlin) by nám nedovolil vytvořit tolik synovských procesů.

V první části kontroluji, jestli je vstup správně zadaný – jestli obsahuje pouze jeden argument a jestli obsahuje pouze přirozená čísla bez nuly. V případě úspěchu, je zaslaná broadcastem zpráva s

[1] https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FPRL-IT%2Flectures%2FPDA06_PRAM_MNG.pdf

celkovým počtem nadmořských výšek a nadmořská výška pozorovatele. V případě neúspěchu se všechny procesy ukončí a vypíší chybovou zprávu.

Počítání výšek je implementováno paralelně, kde každému procesu jsou zaslány hodnoty pro dva výpočty úhlů (výpočet pomocí arctan). Výsledky se hned porovnají (operace max) a větší úhel se uloží na vyšší index. Výsledky si pak procesory uchovávají pro další operace. Index pole, kam proces zapisuje, se počítá jako $processId * 2$ (jelikož má každý proces dvě hodnoty, se kterými manipuluje).

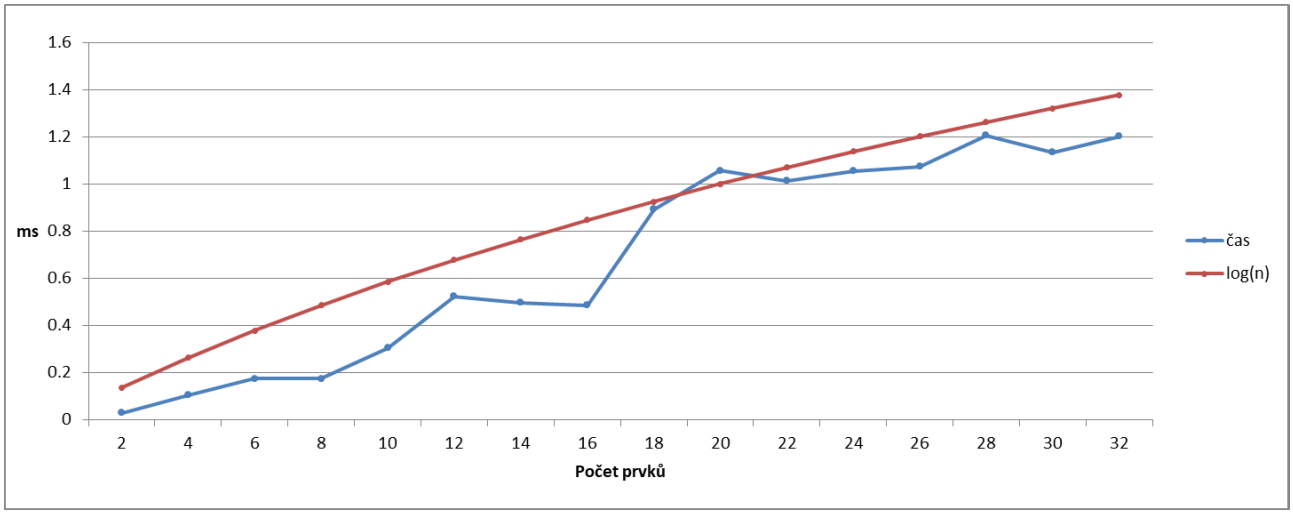
Max_prescan je již trochu zajímavější. Výpočet je uskutečněný pomocí stromu. Ve fázi *upsweep* procházíme stromem od listů ke kořeni tak, že se porovnají hodnoty na dané úrovni. Funguje to tak, že proces vždy dostane aktualizovanou hodnotu, porovná ji se svojí aktuální a výsledek si uloží na index, který je vypočítaný jako $processId * 2 + 1$. Ve fázi *clear* se vymaže poslední prvek v poli. Mazání je implementováno uložením nejnižší hodnoty, kterou může arctan nabývat tedy -90 stupňů $\doteq$ -1,6 rad $\doteq$ -2 rad. Fáze *downsweep* je podobná fází *upsweep* s rozdílem, že se postupuje od kořene k listům (tedy od prvků s vyšší indexem k prvkům s menším). Každý proces si opět ukládá hodnoty do svých indexů.

V poslední fází každý proces vykoná dvě porovnání. V případě, že jeho úhel na indexu $processId * 2$ menší než původní vypočítaný úhel, nastaví do pole $result[processId * 2]$ znak *'v'*, v opačném případě znak *'u'*. Stejné porovnání provede pro $processId * 2 + 1$ a výsledky pošle rootu. Root tímto způsobem obdrží výsledky od všech procesů a nakonec vrátí hodnoty ve správném formátu na výstup. Při implementaci byly použity funkce: *MPI_Recv, MPI_Send, MPI_Bcast, MPI_Barrier* a další funkce pro inicializaci a ukončení MPI.

4 Experimenty

Experimenty byly naměřeny pro 2 až 32 prvků tzn. pro počty procesů 1,2,4,8 a 16. Výpočet času pro stejné počty prvků se prováděl 50× a z každého měření se vzal nejpomalejší proces. Výsledný čas se vypočítal jako medián jednotlivých měření a výsledek lze vidět v grafu 1. Měřila se pouze část vykonávání algoritmu Viditelnosti. Na grafu si lze povšimnout skok z 16 na 18 prvků. V tomto případě dojde k nárůstu procesů z 8 na 16 a výpočet se zpomalí. Dále lze vidět, že křivka roste logaritmicky a tedy jsme ověřili, že časová složitost je skutečně logaritmická $O(log(n))$.

Pro měření času byla použita funkce *MPI_Wtime*[2], která poskytuje vysoké rozlišení při měření času. Funkce byla volána před a po fází řazení a vypočítal se rozdíl těchto dvou hodnot.

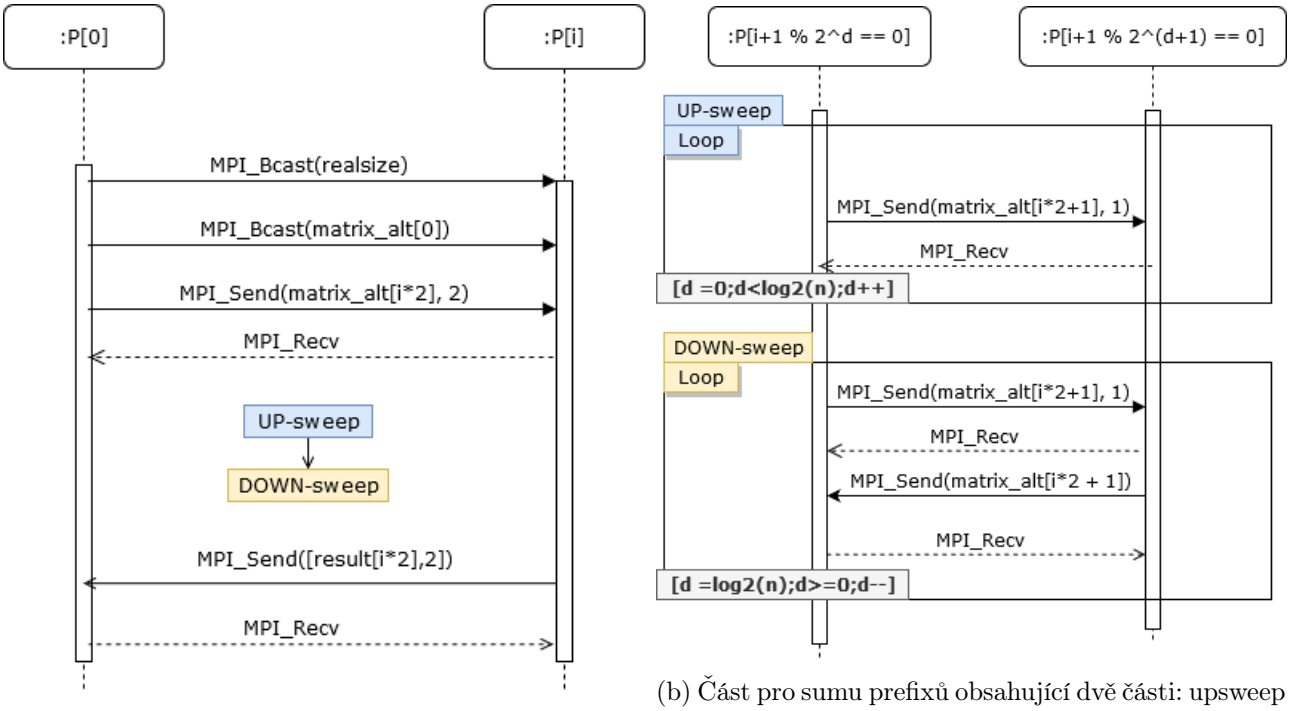


Obrázek 1: Graf zachycuje trvání algoritmu pro různě velké vstupy

[2]https://www.mpich.org/static/docs/v3.2/www3/MPI_Wtime.html

5 Komunikační protokol

Na obrázku 2 je znázorněný komunikační protokol mezi procesory. Jedná se o zjednodušený model pro n procesů. Pro lepší přehlednost je zobrazen na dvou obrázcích. Lze zde vidět, že na začátku root rozešle hodnoty všem procesům, které je zpracují, posílají si navzájem výsledky a nakonec root přijme všechny vypočítané hodnoty od procesů a vypíše je na výstup.



(a) Hlavní část vykonávání programu.

(b) Část pro sumu prefixů obsahující dvě části: upsweep (modře) a downsweep (žlutě).

Obrázek 2: Sekvenční graf zobrazující komunikaci mezi procesy během vykonávání programu.

6 Závěr

V této zprávě se podařilo podrobně popsat algoritmus Viditelnosti a jeho implementace. Program funguje podle očekávání a jeho časová složitost byla ověřena na základě sady experimentů.