Vysoké učení technické v Brně
Fakulta informačních technologií

Paralelní a distribuované algoritmy
3. Projekt – Viditelnost

Dominik Harmim (xharmi00)
xharmi00@stud.fit.vutbr.cz
3. dubna 2020

1 Rozbor algoritmu

Implementovaný algoritmus řeší problém tzv. *viditelnosti*, anglicky zvaný *Line-of-Sight*. Tento problém je definován následovně. Je dána dvou-rozměrná matice terénu ve formě nadmořských výšek a pozorovací bod (*místo pozorovatele*). Je třeba zjistit, které body podél paprsku vycházejícího z místa pozorovatele určitým směrem jsou viditelné. V rámci tohoto projektu je problém zjednodušen tím způsobem, že je dána pouze *sekvence bodů*, která reprezentuje nadmořské výšky terénu podél pozorovacího paprsku. První bod v této sekvenci je nadmořská výška místa pozorovatele.

Algoritmus, který řeší tento problém, označí body podél pozorovacího paprsku jako viditelné, pokud žádný bod mezi pozorovatelem a těmito body nemá větší *vertikální úhel*. Nejdříve se vektor výšek bodů podél pozorovacího paprsku přepočítá na vektor vertikálních úhlů. Následně se spočítá *vektor maximálních úhlů* pomocí operace *prescan* s operátorem *maximum*. Pro zjištění viditelnosti bodu stačí porovnat jeho vertikální úhel s patřičným maximem.

1.1 Analýza algoritmu

Algoritmus provádí operaci *prescan*, která předpokládá takový počet procesů, který je *mocninou čísla 2*. Zároveň každý proces pracuje se dvěma body. Počet prvků je vždy zaokrouhlen na *nejbližší vyšší* mocninu čísla 2. V nejlepším případě, kdy počet bodů n podél pozorovacího paprsku je mocninou čísla 2, je *počet procesů pro výpočet* $p(n) = \frac{n}{2}$. V nejhorším případě je $p(n) = n - 1$.

Prostorová složitost je $s(n) = p(n) \cdot 2$, protože každý proces pracuje se dvěma body. Pokud je počet bodů mocninou čísla 2, potom je $s(n) = n$ a všechen prostor je maximálně využit. V opačném případě je ale určitý prostor vyplněn *neutrálními prvky*.

V algoritmu každý proces paralelně provádí výpočet vertikálních úhlů svých bodů a porovnání vypočítaných úhlů s patřičnými maximálními úhly. Toto se provede v *konstantním čase*. Výpočet maximálních úhlů se provádí operací *prescan*, která má *logaritmickou* časovou složitost. *Celková časová složitost* je tedy $t(n) = \mathcal{O}(\log p(n))$.

Celková cena tohoto algoritmu je následující: $c(n) = t(n) \cdot p(n) = \mathcal{O}(\log p(n)) \cdot p(n) = \mathcal{O}(\log(p(n)) \cdot p(n))$. Tato cena *není optimální*, protože $\mathcal{O}(\log(p(n)) \cdot p(n)) \neq \mathcal{O}(n)$, kde $\mathcal{O}(n)$ je cena *optimálního sekvenčního řešení* tohoto algoritmu. Algoritmus by byl optimální, pokud by se snížil počet procesů a každý proces by zpracovával několik bodů sekvenčně a pokud by platilo, že $\log p(n) < \frac{n}{p(n)}$.

Zrychlení tohoto algoritmu oproti optimálnímu sekvenčnímu řešení, které má časovou složitost $t(n) = \mathcal{O}(n)$, je následující: $\frac{\mathcal{O}(n)}{\mathcal{O}(\log p(n))}$.

2 Implementace

Algoritmus je implementován v programovacím jazyce C++. Je využito knihovny *Open MPI* pro zasílání zpráv mezi procesy. Implementace se nachází v souboru `vid.cpp`. Po spuštění programu se hned po inicializaci knihovny Open MPI zjistí číslo procesu, která mu knihovna přidělila. Podle tohoto čísla se potom rozhoduje, jak se bude daný proces chovat a se kterými procesy bude komunikovat. Procesy se číslují přirozenými čísly včetně čísla 0. Proces s číslem 0, tzv. *hlavní proces*, je proces, který provádí vstupní a výstupní operace a který iniciuje komunikaci s ostatními procesy a celou komunikaci řídí. Program je rozdělen do několika částí, které budou popsány v následujících odstavcích.

Před samotným spuštěním programu je nejdříve potřeba říct, kolik procesů bude pro výpočet potřeba. Při spouštění programu připraveným skriptem `test.sh` je tento počet automaticky spočítán. Skript nejdříve zkontroluje validitu vstupní sekvence bodů. Pokud je tato validní, vypočítá se potřebný počet procesů následujícím výrazem $\frac{2^{ceil(\frac{\log n}{\log 2})}}{2}$, kde n je počet bodů podél pozorovacího paprsku a $ceil$ je funkce, která zaokrouhlí dané číslo na nejbližší vyšší celé číslo. Jinými slovy, počet procesů je polovina počtu bodů podél pozorovacího paprsku zaokrouhlená na *nejbližší vyšší* mocninu čísla 2.

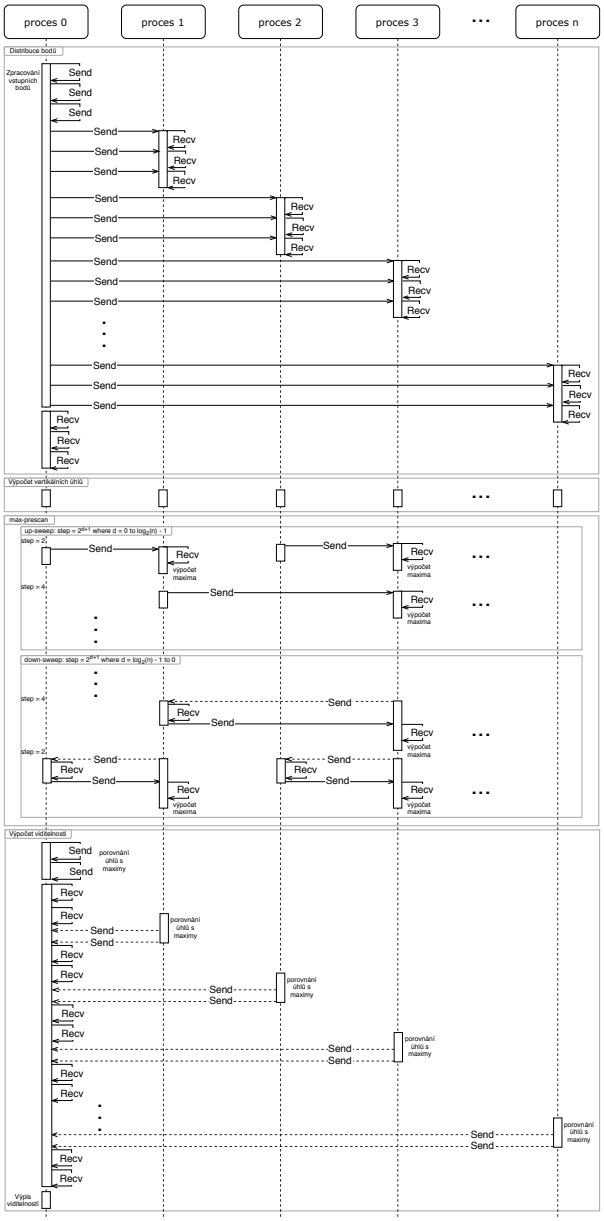
V první části programu hlavní proces provádí funkci `send_alts`, kde se znovu provede validace vstupní sekvence bodů a kontrola, zda byl program spuštěn se správným počet procesů (pro případ, že by program nebyl spuštěn připraveným skriptem). Následně se provede rozeslání vstupních bodů všem procesům. Každý z procesů obdrží

dva po sobě jdoucí body a bod místa pozorovatele. Protože počet procesů byl zaokrouhlen na nejbližší vyšší mocninu čísla 2, sekvence vstupních bodů je doplněna speciálními prázdnými hodnotami tak, aby každý proces obdržel dva body. Všechny procesy následně provádí funkci `receive_alts`, která přijímá patřičné vstupní body od hlavního procesu.

V další části programu nejdříve všechny procesy paralelně volají funkci `calculate_angles`, která počítá *vertikální úhly* daných bodů od místa pozorovatele. Dále se volá funkce `max_prescan`, která provádí operaci *prescan* s binárním asociativním operátorem *maximum* a s *neutrálním prvkem*, který je roven nejmenší možné hodnotě úhlu (konkrétně v C++ je to `numeric_limits<double>::lowest`). Jedná se o částečný výpočet *sumy prefixů*. Provádí se zde operace *up-sweep*, která vypočítá maximální úhel a zároveň jsou ukládány mezivýsledky. Nejdříve je paralelně vypočítáno maximum každých dvou úhlů, dále každých čtyřech úhlů atd. Výpočet končí po $\log_2 n$ krocích, kde n je počet úhlů. Následně se provádí operace *down-sweep*, která nejprve do posledního úhlu uloží neutrální prvek a dále se paralelně zpracovávají každé $\log_2 n$ úhly, následně každé $\log_2(n) - 1$ úhly atd. Výpočet končí po $\log_2 n$ krocích. V každé úrovni zpracování pošle proces svůj úhel procesu vlevo od něj, se kterým komunikuje a dále vypočítá maximum z původního levého úhlu a ze svého úhlu.

Konečně se paralelně vypočítají viditelnosti bodů podél pozorovacího paprsku porovnáním svých úhlů s patřičným vypočítaným maximum a výsledná viditelnost se pošle hlavnímu procesu. Toto se provádí ve funkci `send_visibility`. Hlavní proces potom už jen volá funkci `receive_visibility`, která přijímá vypočítané viditelnosti všech bodů a potom volá funkci `print_visibility`, která tyto výsledné viditelnosti vypisuje na standardní výstup.

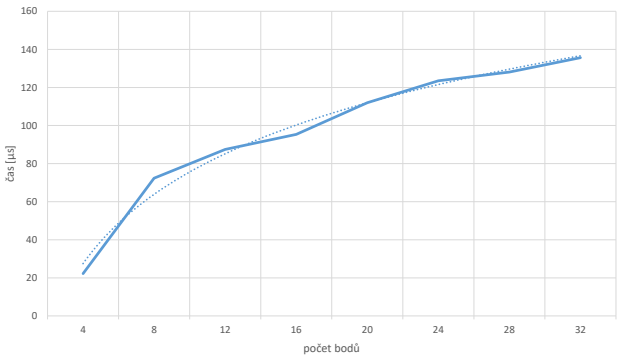
Na obrázku 1 je sekvenční diagram, který znázorňuje výše popsanou komunikaci mezi procesy.



Obrázek 1: Komunikační protokol procesů

3 Experimenty

Pro ověření *teoretické časové složitosti* byly provedeny experimenty, kde byl měřen *reálný čas* provádění algoritmu funkcí `std::chrono::high_resolution_clock::now`. Měření bylo prováděno na superpočítači Salomon pro různý počet bodů od 4 až do 32 bodů. Každé měření pro určitý počet bodů bylo prováděno několikrát a výsledky byly průměrovány, aby nebyly vidět případné odchylky. Výsledky tohoto měření jsou k vidění na obrázku 2. Je zřejmé, že reálná časová složitost roste přibližně *logaritmicky*, stejně jako teoretická.



Obrázek 2: Výsledky měření experimentů

4 Závěr

V rámci tohoto projektu byla úspěšně vytvořena implementace algoritmu, který řeší problém *viditelnosti*. Teoretická časová složitost tohoto algoritmu byla ověřena reálnými experimenty.