Dokumentácia k 2. projektu z PRL
Implementácia algoritmu Viditeľnosti

Tomáš Lapšanský (xlapsa00)

Apríl 2020

1 Úvod

Projekt sa zaoberá implementáciou algoritmu *viditeľnosti* v jazyku C/C++ pomocou knižnice OpenMPI. Vrámci projektu bolo potrebné vytvoriť súbor test.sh, ktorý zabezpečuje spustenie testovacieho procesu algoritmu, a taktiež teoreticky popísať a prakticky zmerať časovú zložitosť algoritmu.

2 Rozbor a analýza algoritmu

Algoritmus je navrhnutý a implementovaný tak, aby každý procesor spracovával 2 hodnoty. Pre N vstupných hodnôt je teda potrebných aspoň *N/2* procesorov. Procesory simulujú stromovú štruktúru algoritmu a každý vykonáva *log(N)* krokov, vždy teda daný procesor komunikuje s procesorom na základe indexu jeho iterácie a chová sa buď ako rodičovský procesor (v tomto prípade aj ako pravý potomok), alebo jeho ľavý potomok.

Algoritmus prvotne rozdistribuuje hodnoty poľa procesorom, každému procesoru pridelí dve hodnoty, teda procesoru s indexom X pridelí hodnotu s indexom $X/2$ (celočíselné delenie) a s indexom $X/2 + 1$ zo vstupného poľa. Hodnoty zo vstupného poľa prideľuje procesorom, ktorých index je maximálne $log_2(N)$ po zaokrúhlení nahor. Po vyčerpaní týchto hodnôt im prideľuje takzvanú $DEFVALUE$, ktorá je menšia ako každá hodnota zo vstupného poľa (v tomto prípade -1), aby bolo možné tieto procesory použiť, no nijako nenarušovali výpočetný chod. Ostatným procesom sú rozdistribuované hodnoty tak, aby vedeli že nebudú použiteľné v danom behu, a teda sú okamžite po obdržaní hodnôt vypnuté.

Následne každý procesor vykonáva kalkuláciu svojho uhlu v závislosti na počiatočnom bode poľa a vzdialenosti týchto bodov. Nasleduje vykonanie algoritmu *max_prescan*, ten sa skladá z 3 hlavných etáp, *upSweep*, *clear*, *downSweep*. Vrámci algoritmu *upSweep* každý aktívny proces v prvej iterácií vyberie maximum zo svojich 2 hodnôt a uloží túto hodnotu do druhej hodnoty. Následne sa

jednotlivé procesory podľa iterácie delia na prijímateľov a odosielateľov hodnôt podľa svojho indexu. Odosielatelia simulujú ľavého syna a prijímatelia uzol podľa aktuálnej iterácie a pravého syna. Prijímateľ prevezme hodnotu od odosielateľa a uloží do hodnotu z pravého a ľavého syna uzlu. Tento algoritmus prebieha až do dosiahnutia vrcholu stromu. Následne aktívny proces s najvyšším indexom (kde sa nachádza hodnota vrcholu stromu) nastaví hodnotu vrcholu stromu na neutrálnu hodnotu (*clear*) a začína algoritmus *downSweep*. V tomto algoritme vykoná každý proces $log_2(N)$ krokov, v každom kroku sa overí, či je procesor rodičovským uzlom, jeho ľavým synom alebo aktuálne neaktívnym uzlom. Ak je procesor rodičovským uzlom, distribuuje svoju hodnotu ľavému synovi a do hodnoty pravého syna (svojej hodnoty) uloží maximum zo svojej hodnoty a pôvodnej hodnoty ľavého syna. Odosielateľ distribuuje svoju hodnotu svojmu rodičovskému procesoru. Ak procesor dôjde k listom stromu, urobí rovnakú operáciu nad svojimi dvoma hodnotami.

Následne každý procesor overuje hodnotu svojho pôvodného uzlu a hodnoty maxima ktoré dosiahol prostredníctvom algoritmu *max_prescan*, ak je táto hodnota väčšia, hodnota na jeho indexe by mala byť viditeľná, v opačnom prípade neviditeľná. Tieto hodnoty pre obe svoje hodnoty distribuuje hlavnému procesu, ktorý následne rieši dalšie výpisy na štandardný výstup.

2.1 Zložitosť

Asymptotická časová zložitosť algoritmu *Viditeľnosť* je *O(n/N + log(N))*, kde n je počet hodnôt a N je počet procesorov. Ak by sme si algoritmus detailnejšie rozobrali, zistíme nasledovné. Na každom procesore sa vykoná výpočet uhlov podľa počtu jemu pridelených hodnôt, dostávame teda zložitosť n/N. Následne sa vykonávajú algoritmy *upSweep* a *downSweep*, tie oba vykonávajú maximálne *log(N)* krokov. Po jednoduchom výpočte dostaneme *O(n/N + log(N))*.

3 Implementácia

Algoritmus je implementovaný v jazyku C++ s využitím knižnice OpenMPI. Na začiatku je potrebná inicializácia paralelného prostredia pomocou volania MPI_Init() a ďalších funkcií pre identifikáciu jednotlivých procesorov. Následne už program rozdeľuje svoj chod podľa toho, či ho vykonáva hlavný procesor (s *id 0*) alebo ostatné procesory. Hlavný proces vstupuje do funkcie *parseInput* kde distribuuje jednotlivým procesorom hodnoty zo vstupu. Každý procesor obdrží 2 hodnoty. Počet aktívnych procesorov je vždy zaokrúhlený na vyššiu mocninu dvojky (to znamená že ak je na vstupe 6 hodnôt, aktívne budú 4 procesory, nie iba 3). Aktívnym procesorom ktoré neobdržia hodnotu zo vstupu je poslaná hodnota *DEFVALUE*, ktoré je programom inicializovaná na hodnotu -1, keďže je táto hodnota nižšia ako môžeme pre štandardné vstupy očakávať, a teda uhly pre ňu vypočítané nezasahujú do chodu programu. Ostatným procesorom je pridelená hodnota *DEFINACTIVEVALUE*, a ukončujú svoju činnosť

po obdržaní týchto hodnôt. Procesorom je taktiež distribuovaná hodnota *baseValue*, ktorá reprezentuje prvú hodnotu vstupu pre výpočet uzlov, a hodnota *activeProc*, ktorá reprezentuje počet aktívnych procesorov.

Každý procesor následne vykalkuluje hodnotu jeho uhlu pomocou funkcie *calculateAngle* a túto hodnotu si ukladá, následne spúšťa vykonávanie behu funkcie *upSweep*. V prvej iterácií funkcie *upSweep* funkcia uloží hodnotu maxima z 2 hodnôt uhlov do druhého uhlu, následne už bude pracovať iba s týmto druhým uhlom, keďže to od neho vyžaduje stromová štruktúra algoritmu. Procesory sa podľa výpočtovej podmienky delia na tie, ktoré sú pre daný počet aktívnych procesorov v iterácií prijímatelia alebo odosielatelia. Odosielať odosiela svojmu "otcovskému uzlu" svoju hodnotu, a prijímateľ ako otcovský uzol túto hodnotu príma a uloží do svojej hodnoty (ktorá reprezentuje aj jeho pravého syna) hodnotu maxima z týchto dvoch hodnôt. Každý procesor volá rekurzívne funkciu *upSweep* s polovičným počtom aktívny procesorov, až do doby keď ostanú aktívne iba 2 (posledná iterácia) alebo je procesor vypočítaný ako neaktívny. Následne procesor s najvyšším aktívnym indexom resetuje vrchol stromu na hodnotu *-DBL_MAX* a pokračuje volaním funkcie *downSweep* (ostatné procesory túto funkciu samozrejme volajú tiež). Funkcia *downSweep* vykonáva vždy *log(N)* krokov. V každom kroku pomocou funkcií *isParent* a *isChild* overuje pomocou indexu procesoru a čísla iterácie, či by procesor mal byť v aktuálnom kroku ľavým synom, otcovským uzlom alebo aktuálne neaktívny. Ak je procesor otcovským procesorom v danej iterácií, príjme hodnotu ľavého syna pomocou *MPI_Recv*, odošle mu svoju hodnotu pomocou *MPI_Send* a uloží do svojej hodnoty (hodnoty pravého syna) hodnotu maxima zo svojej a obdržanej hodnoty. Ak je procesor ľavým synom, odošle svoju hodnotu uzla a príjme novú. Po ukončení základným iterácií spraví každý procesor tento mechanizmus nad svojimi internými hodnota, teda porovná hodnotu *angle2* s ktorou doteraz pracoval s hodnotou *angle1*, do *angle1* je uložená hodnota *angle2* a do *angle2* ukladá maximum z týchto hodnôt.

Následne každý procesor porovnáva hodnoty *angle1* a *angle2* s hodnotami pôvodných uhlov ktoré si predtým uchoval, ak je hodnota pôvodného uzlu väčšia, než hodnota na danom indexe, je prehlásený za viditeľný, v opačnom prípade nie. Tieto hodnoty sú následne distribuované hlavnému procesu pre výpis na *STDOUT*.

4 Popis experimentov

Implementovaný algoritmus bol testovaný experimentálne. Bol vytvorený *Python script*, ktorý generoval pre rôzne dĺžky vstupov náhodné hodnoty vstupu. Takýto test bol spustený 100x pre jednu dĺžku vstupu a ich výsledná časová hodnota bola spriemerovaná. Keďže podstatný bol pohľad na algoritmus samotný a nie na réžiu celého programu, rozhodli sme sa implementovať jednoduché meranie času. Meranie času je možné aktivovať zmenou premennej $MEASURE$ na hodnotu *true*. Pre meranie algoritmu sme sa rozhodli využiť bariéru pomocou *MPI_Barrier*, bariéra je využitá po rozdistribuovaní hodnôt pre zmeranie času

začiatku hlavného procesu a následne po výpočte hodnôt *visibility* jednotlivých procesorov, kde znova ukladá hlavný procesor čas.

Testovanie prebehlo na superpočítači *Salomon*, testované bolo prostredie, kde sme prispôsobovali veľkosť vstupu počtu pridelených procesorov tak, aby každý procesor obsahoval 2 hodnoty. Keďže algoritmus ráta so stromovou štruktúrou, bolo potrebné zaokrúhliť počet procesorov najbližšej mocnine dvojky, teda napríklad pre 20 vstupov bolo potrebných 16 procesorov, nie 10 ako by sa mohlo zdať.

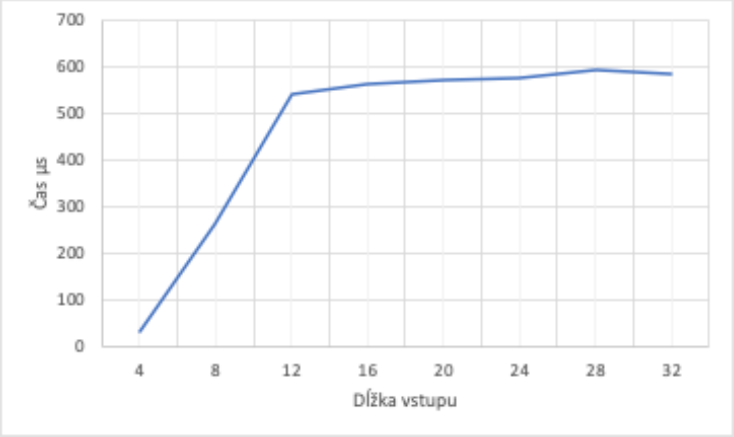
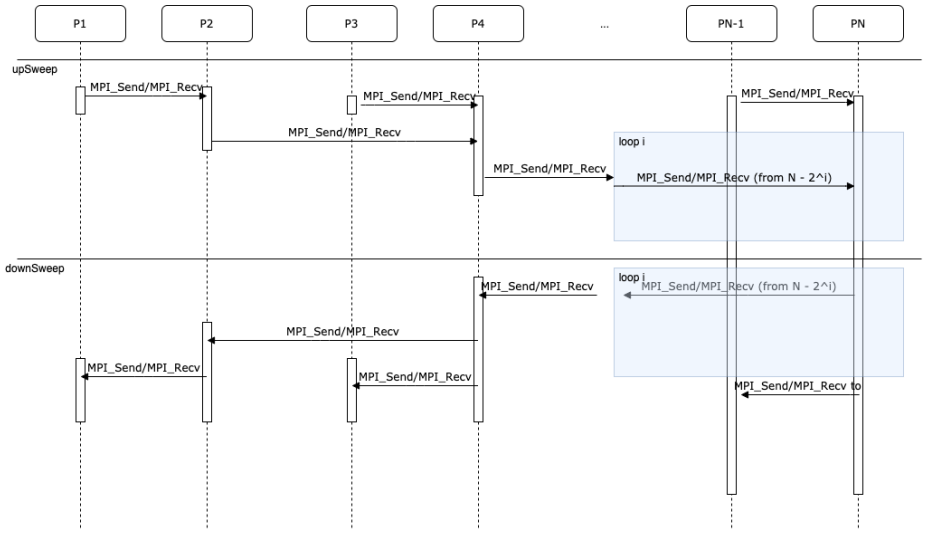


Figure 1: Graf časovej závislosti na dĺžke vstupu

Na dosiahnutých výsledkoch je vidieť naznačenie logaritmickej časovej zložitosti algoritmu, keďže je závislý na počte procesorov ktoré ho obsluhujú, preto je napríklad rozdiel medzi hodnotami pre dĺžku vstupu 20 a 24 zanedbateľný, no pri viditeľný pri hodnotách kde sa mení počet potrebných procesorov.

5 Komunikačný protokol

Komunikačný protokol znázorňuje komunikácia medzi procesormi *p1 .. pn*, pomocou MPI, konkrétne MPI_Send a MPI_Recv (každé prijatie MPI_Send je prijaté pomocou MPI_Recv). Protokol neznázorňuje distribúciu hodnôt medzi procesormi, ale iba komunikáciu pri vykonávaní algoritmu.



6 Záver

Testy vykonané na algoritme nasvedčujú predpokladanú zložitosť algoritmu. Jemné odchýlky v nameraných hodnotách môžu byť spôsobené počtom procesorov, ktoré program využíva, a spôsob zaokrúhlenia ich počtu na mocniny čísla dva, takže graf nemá čisté logaritmické stúpanie.