Semestrální projekt MI-PPR 2015/16

Paralelní algoritmus pro řešení

Steinerova stromu

Libor Vytlačil, Vladimír Mlázovský

Magisterské studium, FIT-ČVUT, Kolejní 550/2, 160 00 Praha 6

Prosinec 2015



# Definice problému a popis sekvenčního řešení

## Vstupní data

 = přirozené číslo představující počet uzlů grafu ,   
 = přirozené číslo představující počet hran grafu   
 = přirozené číslo v řádu jednotek představující průměrný stupeň uzlu grafu ,   
 = jednoduchý neorientovaný neohodnocený souvislý graf o uzlech,  hranách a průměrném stupni   
 = množina o velikosti , , počet náhodně vybraných uzlů z grafu

## Doporučení pro generování G

Použijte generátor grafu s typem grafu -t AD, který vygeneruje souvislý neorientovaný neohodnocený graf.

## Definice

Steinerův strom  v grafu  je podgraf , který je stromem a obsahuje všechny uzly z množiny .

## Úkol

Zkonstruujte minimální Steinerův strom  v grafu (tedy steinerův strom s minimálním počtem uzlů). Řešení existuje vždy, ale nemusí být jednoznačné.

## Výstup algoritmu

Počet uzlů minimálního Steinerova strom a jeho výpis formou matice incidence.

## Sekvenční algoritmus:

Sekvenční algoritmus je tedy typu BB-DFS s hloubkou prohledávaného prostoru omezenou na . Cena, kterou minimalizujeme, je počet přidaných uzlů. Algoritmus končí, když je nalezena cena rovna triviální spodní mezi (tj. ), jinak algoritmus prohledává celý stavový prostor.

1. Zkusíme ověřit, zda zadaná množina neodpovídá řešení - není potřeba přidávat k žádný uzel. Ano? Vypíšeme graf.
2. Na zásobník se přidají první stavy (tj. uzly, které nepatří do z )
3. Stavy se postupně rozvíjí a zpracovávají
   1. Zkusíme přidat uzel do
      1. Pokud už máme řešení v menší hloubce, tuto větev výpočtu zařízneme.
      2. Pokud nalezneme řešení, uložíme si nové nejlepší řešení (nastavíme menší maximální hloubku) a vrátíme se.
      3. Pokud řešení nenalezneme, rozvíjíme dál.
4. Po souběžném projití a prořezání stavového prostoru vypíšeme na výstup nejlepší nalezené řešení.

Stavy rozvíjíme tak, že vždy přidáváme uzel s vyšším pořadovým číslem, než je aktuální zpracovávaný uzel, a zároveň uzel nepatří do . Tím zajistíme, že se do testu dostanou jen unikátní množiny uzlů.

Rozvinutý stavový prostor pro množinu uzlů .

### Ověření

Ověřování probíhá tak, že se vybere podgraf z  obsahující všechny vybrané vrcholy s hranami mezi nimi a zkontroluje se souvislost.

# Popis paralelního algoritmu a jeho implementace v MPI

## Práce jednotlivých procesorů

Každý procesor má úplnou informaci o zadání, počtu procesorů.

Práce je odlišná pro nultý procesor, který ověřuje triviální řešení a rozešle první expanze zásobníku.

Poslední procesor navíc sbírá informaci o nejlepším dosaženém výsledku.

Běžný pracující procesor si řeší pro sebe přidělenou práci, dokud ji má. Občas zkontroluje, zda ho někdo nežádá o práci. Pokud je žádán o práci, rozdělí svou na dvě částí a rozdělí se. Svoji práci si dodělá a žádá o další. Když na něj i po mnoha dotazech žádná práce nedostane, končí.

Pokud je procesorů příliš mnoho, některé se vůbec nevyužijí.

## Stavy procesorů

Každý procesor se nachází v jednom ze stavů

* INIT – inicializace
* WORKING – pracuji
* IDLE – nepracuji
* IDLE\_EXPECTING – nepracuji, ale požádal jsem si o práci
* DONE – nepracuji, a očekávám konec výpočtu
* EXIT – konec

Procesory se chovají jako jednoduchý stavový automat.

## Komunikace

Při jakékoliv komunikaci si procesory vymění informaci o nejlepším dosaženém řešení.

Typy zpráv:

* Žádost o práci
* Odeslání práce
* Není práce
* Token
* Konec
* Žádost o nejlepší řešení
* Odeslání nejlepšího řešení

Mezi procesory se dokola podle [Dijkstrova algoritmu](http://people.cs.aau.dk/~adavid/teaching/MVP-10/17-Termination-lect14.pdf) posílá token. Ten obíhá všechny procesory. Pouze neaktivní procesor posílá token dál. Token je obarven na procesoru, který zaslal práci proti směru posílání tokenu. Token je na nultém procesoru odbarven a obíhá znova, dokud se nevrátí do nultého procesoru neobarvený. Díky tomu může být výpočet ukončen ještě dříve, než by všechny procesory vzdaly žádání o práci a ukončily se samy.

## Další zrychlení

Nastavili jsme jistou minimální řeznou výšku, aby se zbytečně nepředávalo málo práce a nezdržovalo se komunikací. Pokud je se má tedy odeslat zásobníkový rámec (stav), který se už nedá příliš rozvíjet, je raději rozvinut na stejném procesoru než, aby se odesílal dál.

Procesory si jednou za mnoho cyklů vyžádají délku nejlepšího řešení z posledního procesoru. Tím se může velice efektivně prořezávat.

# Naměřené výsledky a vyhodnocení

K měření jsme připravili 3 grafy. Na těch se se zvětšením počtu procesorů snižuje délka výpočtu. Měřili jsme čas od sesynchronizování před přečtením vstupu po ukončení činnosti všech procesorů.

## Grafy pro měření

3 testované grafy mají následující podobu

* Nejhorší možný případ o 32 uzlech
  + Je nutné projít celý stavový prostor k nalezení řešení
* Náhodný řídký graf se 40 uzly
* Náhodný řídký graf se 42 uzly

## Naměřené hodnoty

Čas byl měřen pro hledání řešení ve třech výše uvedených grafech. Čas je v sekundách.

|  |  |  |  |
| --- | --- | --- | --- |
| # CPU | Čas "Nejhorší případ 32" | Čas "Náhodný 40" | Čas "Náhodný 42" |
| **1** | 243 | 247,9 | 793 |
| **2** | 124,8 | 248,8 | 172,8 |
| **4** | 62,7 | 248,7 | 173,6 |
| **5** | - | 248,4 | 7,206 |
| **6** | - | 249,1 | 7,203 |
| **7** | - | 14,04 | 7,361 |
| **8** | 31,17 | 0,0159 | 7,185 |
| **12** | 31,17 | - | 7,229 |
| **16** | 19,84 | 0,081 | 8,98 |
| **24** | 17,52 | 0,2868 | 0,4711 |

*Celkový čas nutný ke zpracování.*

*Cena (čas \* počet procesorů) (log)*

*Zrychlení na procesor (log)*

*Zrychlení na procesor pro jednotlivá zadání.*

## Zrychlení

U zadání nejhoršího případu se zrychlení blíží lineárnímu. Z grafu je vidět, že komunikace a režie s ní spojená výpočet brzdí.

U zadání náhodného grafu se čtyřiceti uzly dlouho ke zrychlení nedochází. Až při sedmi procesorech dochází k průlomu. Pak je zrychlení naopak extrémní, protože je velmi brzy nalezeno *nějaké řešení*. Díky známému řešení je možné začít prořezávat stavový prostor. Problém se tak začne extrémně rychle zmenšovat. Při větším počtu procesorů už začíná narůstat režie a zrychlení není tak extrémní.

Graf zrychlení třetí zadání obsahuje tři *schody*. Pro 2, 5 a 24 procesorů se problém rozdělí tak, že je pak brzy nalezeno nějaké dobré řešení a pak je zbytek procházení stavového prostoru otázkou prořezávání.

## Hledání dárce

Pokud dojde na jednom procesoru práce, zažádá si o další. Žádá ve smyčce od nejbližšího souseda s vyšším ID až po poslední procesor a pak znova od nultého. S dotazy končí když mu dorazí zpátky práce nebo pokud i po stanoveném množství dotazů žádnou práci nedostane, tak sám skončí.

## Dělení zásobníku a dynamické vyvažování zátěže

((( prosím doplnit )))

Zátěž se tedy rozloží právě ve chvíli, když si procesor zažádá o další práci

## Meze pro řešení problému

počet vrcholů grafu

počet hran grafu

Nad složitostmi uvažuji asymptoticky, zanedbávám konstanty.

Náš algoritmus není nijak zvlášť paměťově náročný. Největší je samotný graf, který v paměti zabírá . Paměť není to, co nám dělá starosti.

Ověření souvislosti grafu, pomocí které probíhá ověření řešení, probíhá ve . Je potřeba ověřit (v nejhorším případě) možných kombinací vrcholů. Časová složitost nalezení řešení je tedy exponenciální. To nám starosti dělá.

Empiricky jsme zjistili, že nemá smysl předávat práci s hloubkou stavového prostoru méně než 15. To nám udává nejmenší problém, který už nemá cenu více rozmělňovat.

# Závěr

Já, Libor Vytlačil, jsem (((doplnit, prosím))).

Já, Vladimír Mlázovský, jsem si oživil programování v C++ a zamyslel jsem se hlouběji nad paralelizací problému. Nenapadlo by mě, že až takový rozdíl programovat více vláknovou aplikaci a paralelní stroj.