Semestralni projekt PAR 2009/2010:

Paralelni algoritmus pro reseni problemu bipartitniho podgrafu

Petr Smejkal
David Vavrousek

5. rocnik, obor pocitace, K336 FEL CVUT, Karlovo nam. 13, 121 35 Praha 2

May 10, 2010

1 Definice problemu a popis sekvencniho algoritmu

1.1 Definice problemu BPG: bipartitni podgraf

1.1.1 Vstupni data

n = prirozene cislo predstavujici pocet uzlu grafu, n 5
k = prirozene cislo radu jednotek predstavujici maximalni stupen uzlu grafu
G(V,E) = jednoduchy neorientovany neohodnoceny souvisly graf o n uzlech a stupni nejvyse k
Doporuceni pro algoritmus generovani G:

Pouzijte generator grafu Generator1 s typem grafu "-t 1", ktery vygeneruje souvisly neorientovany neohodnoceny graf.

1.1.2 Definice

Graf G(V,E) je bipartitni jestlize muzeme rozdelit mnozinu uzlu na disjunktni pomnoziny U a W tak, ze kazda hrana v G spojuje uzel z U s uzlem z W. Bipartitni graf je mozne uzlove obarvit 2 barvami.

1.1.3 Ukol

Graf G(V,E) je definovan mnozinou uzlu V a mnozinou hran E. Ukolem je nalezt maximalni podmnozinu hran F takovou, ze graf G(V,F) je bipartitni (viz obrazek).

1.2 Popis sekvencniho algoritmu

Zakladnim algoritmem bylo prochazeni stavoveho prostoru do hloubky. Hloubka stavoveho stromu je omezena na |E|. Pro reprezentaci stavu jsme si vytvorili tridu State, ktera obsahuje matici sousednosti grafu. Pripustny mezistav je podmnozina hran F, ktera tvori bipartitni podgraf G(V,F). Cena, kterou maximalizujeme, je pocet hran v F. Pokud je graf G bipartitni, pak trivialne F=E, jinak F je podmnozinou E. Vstupni graf nejprve otestujeme, zda je bipartitni (linearni algoritmus) a pokud ne, pouzijeme algoritmus prohledavani do hloubky. Zakladni myslenkou algoritmu je postupne odebirani hran. Na zasobnik se vlozi pocatecni stav, reprezentujici zadany graf. Dale se postupuje podle algoritmu prohledavani do hloubky, pri expanzi se na zasobnik vlozi 2 nove stavy, jeden s hranou n a jeden bez ni. Celkovy pocet hran n urcuje maximalni hloubku stromu reprezentujiciho graf stavoveho prostoru. Pro orezavani stavoveho prostoru vyuzivame techniky branch and bounds, cely algoritmus je tedy DFS-BB. Jedna se o uplne prohledavani stavoveho prostoru do hloubky |E|-|V|. Prohledavani se muze navracet v mezistavech s |F|=|V|-1 (strom je trivialne bipartitni, proto musi existovat reseni s |F|=|V|-1).Trivialni dolni mez je |F|=|V|-1.Tesna horni mez neni znama. Trivialni horni mez je |F|=|E|.

!!!!!!!!!!!TABULKA SEKVENCNICH CASU !!!!!!!!!!!

2 Popis paralelniho algoritmu a jeho implementace v MPI

Zakladem paralelniho algoritmu je sekvencni algoritmus. S tim rozdilem, ze stavovy prostor se disjunktne rozdeli mezi vice procesoru, ktere ho samostatne zpracovaji. Kvuli omezujicim podminkam algoritmu DFS-BB si procesory mezi sebou posilaji nalezena nova nejlepsi reseni. Jednotlive procesory maji svuj vlastni lokalni zasobnik

a pouziva se dynamicke vyvazovani vypocetni zateze (load balancing). Vyvazovani zateze je implementovano pomoci 2 algoritmu: algoritmu pro deleni zasobniku (ADZ) a algoritmu pro hledani darce (AHD), kterymi necinny procesor s prazdnym lokalnim zasobnikem (stav I=idle) ziskava praci od vhodneho darce, coz je aktivni procesor s neprazdnym zasobnikem (stav A=active), jehoz zasobnik odpovida dostatecne velkemu stavovemu podprostoru. Procesor si o praci zada procesor s cislem o jedno vetsi, pokud ten praci nema, zvysi se citac o jednicku, takto cyklicky zada procesory dokud neobdrzi praci nebo pozadavek na ukonceni. Na pocatku procesor P1 zna pocet procesoru p, na kterych se uloha bude resit. Provede dostatecny pocet expanzi pocatecniho stavu, rozdeli svuj zasobnik na p casti a rozesle jednotlive casti ostatnim procesorum. Vsechny procesory zacnou prohledavat svuj prideleny podprostor a pritom realizuji programovou smycku naznacenou na tomto obrazku.

!!!!!!!!!!! OBRAZEK algoritmu !!!!!!!!!!!!

Aktivni procesor, ktery vycerpa svuj prideleny dil prace, se stane necinny a pomoci algoritmu pro hledani darce AHD vybere darce a posle mu zadost o praci. Pokud mu darce praci posle, prepne se zpet do stavu aktivni. Pokud se mu vrati odmitnuti, vse se opakuje. Procesor ve stavu necinny musi take periodicky kontrolovat, zda nema ve fronte zadosti o praci, na ktere odpovida odmitnutim, nebo zpravy o nalezeni reseni jinym procesorem. V pripade, ze se jedna o optimalni reseni, jsou to v podstate zadosti o ukonceni vypoctu. Aktivni procesor provadi fixni objem prace nad lokalnim zasobnikem (expanduje urcity dany pocet stavu na lokalnim zasobniku) a pak kontroluje frontu zprav. V te mohou byt zadosti o praci, informace o nalezeni reseni jinym procesorem a zadosti o ukonceni vypoctu. Aktivni procesor zpracovava zadosti o praci nasledujicim zpusobem: Pokud je velikost lokalniho zasobniku vetsi nez nastavena minimalni mez, tak se zasobnik rozdeli na polovinu a jedna cast se zasle zadateli o praci. Pokud neni, tak procesor pozadavek odmitne.

3 Analyza slozitosti paralelniho reseni

3.1 Analyza sekvencniho reseni

Slozitost sekvencniho reseni lze rozdelit na dve casti. Nejprve na to, kolik stav; se projde, cili jak je velky stavovy prostor a pro kazdy stav se pak jeste provadi algoritmus BFS pro testovani bipartitnosti. Orezavani algoritmem Branch and Bound se neda odhadnout, proto ho neuvazujeme. Vyska stromu stavoveho prostoru odpovida poctu hran. V kazdem kroku jednu hranu bud odeberu, nebo necham.

Pocet prohledavanych stavu je tedy:

$\sum_{i=0}^{|E|} 2^i = 2^{|E|} + 2^{|E|-1} + ... + 2^0 = O(2^{|E|})$

Test bipartitnosti se provadi prochazenim do sirky. Slozitost tohoto alogirtmus je :

$O(|E| + |V|)$

Celkova slozitost je tedy:

$SU(|E|, |V|) = O(|E| + |V|) \times O(2^{|E|}) = O(2^{|E|} \times (|E| + |V|))$

3.2 Analyza paralelniho reseni

Pri paralelnim algoritmu se nejprve expanduje prvnich p stavu, ktere se rozeslou, potom si p procesoru rozdeli stejnou praci, jakou mel sekvencni algoritmus. Toto tvori vypocetni slozku slozitosti. Vedle te jeste musime zapocist slozku komunikacni, ktera vyjadruje komunikac mezi procesory a predavani prace. Tato slozka je uzce zavisla na rovnomernem rozdeleni prace a poloze hledaneho reseni ve stavovem prostoru. Konkretne tuto slozku tedy vyjadrit nemuzeme, nicmene rozhodne neni zanedbatelna, proto ji ve vzorci uvadime symbolicky.

Paralelni cas tedy je:

$T(|E|, |V|, p) = T_v(|E|, |V|, p) + T_k(|E|, |V|, p) = O(p + \frac{2^{|E|} \times (|E|+|V|)}{p}) + T_k(|E|, |V|, p)$

Teoreticka paraleln cena:

$$C(|E|,|V|,p) = p * T(|E|,|V|,p) = O(p^2 + (2^{|E|} \times (|E| + |V|)) + T_k(|E|,|V|,p) \times p$$

Efektivnost:

$$E(|E|,|V|,p) = \frac{SU(|E|,|V|)}{C(|E|,|V|,p)} = \frac{O(2^{|E|} \times (|E|+|V|)}{O(p^2+(2^{|E|} \times (|E|+|V|)))+T_k(|E|,|V|,p) \times p}$$

A zrychleni:

$$S(|E|,|V|,p) = \frac{SU(|E|,|V|)}{T(|E|,|V|,p)} = \frac{O(2^{|E|} \times (|E|+|V|))}{O(p+\frac{2^{|E|} \times (|E|+|V|)}{p})+T_k(|E|,|V|,p)} = O(p + \frac{2^{|E|} \times (|E|+|V|)}{p}) + T_k(|E|,|V|,p)$$

4 Namerene vysledky a vyhodnoceni

1. Zvolte vice instanci problemu s takovou velikosti vstupnich dat, pro ktere ma sekvencni algoritmus casovou slozitost radu nejmene jednotek minut a nejvyse nekolik malo hodin. Pro mereni cas potrebny na cteni dat z disku a ulozeni na disk neuvazujte a zakomentujte ladici tisky, zpravy a vystupy.
2. Merte paralelni cas pri pouziti $i = 2, \ldots, 16$ procesu.
3. Pri mereni kazde instance problemu na dany pocet procesoru spoctete pro vas algoritmus dynamicke delby prace celkovy pocet odeslanych zadosti o praci, prumer na 1 procesor a jejich uspesnost.
4. Mereni pro dany pocet procesoru a instanci problemu provedte 3x a pouzijte prumerne hodnoty.
5. Z namerenych dat sestavte grafy zrychleni $S(n, p)$. Zjistete, zda a za jakych podminek doslo k superlinearnimu zrychleni a pokuste se je zduvodnit.
6. Vyhodnodte komunikacni slozitost dynamickeho vyvazovani zateze a posudte vhodnost vami implementovaneho algoritmu pro hledani darce a deleni zaosbniku pri reseni vaseho problemu. Posudte efektivnost a skalovatelnost algoritmu. Popiste nedostatky vasi implementace a navrhnete zlepseni.

5 Literatura

A Navod pro vkladani grafu a obrazku do Texu

Nejjednodussi zpusob vytvoreni obrazku je pouzit sunovsky graficky editor xfig, ze ktrereho lze exportovat latex formaty (v poradi prosty latex, latex s macry epic, eepic, eepicemu) a postscript formaty, uvedene poradi odpovida rustu komplikovanosti obrazku (postscript umi jakykoliv obrazek, prosta latex macra pouze jednoduche, epic makra neco mezi, je treba vyzkouset). Nasleduji priklady pro vsechny pripady.

Obrazek v postscriptu, vycentrovany a na celou sirku stranky, s popisem a cislem. Vsimnete si, jak ridit velikost obrazku.

Figure 1: Popis vaseho obrazku

Obrazek pouze vlozeny mezi radky textu, bez popisu a cislovani.

Texovske obrazky maji pripony *.latex, *.epic, *.eepic, a *.eepicemu, respective. Vypustenim zavorek `figure`

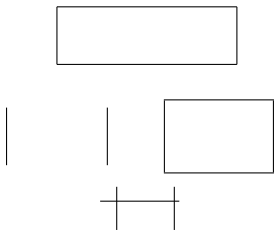

Figure 2: Popis vaseho obrazku

dostanete opet pouze ramecek v textu bez cisla a popisu.

Takhle jednoduse muzete poskladat obrazky vedle sebe.

Ridit velikost texovskych obrazku lze prikazem

```
\setlength{\unitlength}{0.1mm}
```

ktere meni meritko rastru obrazku, Tyto prikazy je ale soucasne nutne vyhodit ze souboru, ktery xfig vygeneroval.

Pro vytvareni grafu lze pouzit program gnuplot, ktery umi generovat postscriptovy soubor, ktery vlozite do Texu vyse uvedenym zpusobem.