Semestrální projekt MI-PDP 2016/2017:

Paralelní algoritmus pro řešení problému maximálního bipartitiního podgrafu

Martin Melka

magisterské studium, FIT ČVUT, Thákurova 9, 160 00 Praha 6

May 3, 2017

1 Definice problému a popis sekvenčního algoritmu

1.1 Definice problému

Řešení problému maximálního bipartitního podgrafu (MBG) znamená nalézt takový podgraf H daného grafu G, který je bipartitní a jeho počet hran je maximální, tj. neexistuje žádný další bipartitní podgraf grafu G, který by měl větší počet hran než H. Množinu hran grafu H označme F.

Graf je bipartitní právě tehdy, když lze všechny jeho uzly obarvit dvěma barvami.

Jelikož je vstupní graf souvislý, lze určit spodní mez počtu hran řešení. Tou bude $|F| = n - 1$, kde n je počet hran grafu. Toto řešení je dáno tím, že pro každý souvislý graf lze najít kostru, tedy takový podgraf, který je strom. Kostru pak stačí "zakořenit", uzly uspořádat do úrovní, jak je u stromů zvykem, a každou úroveň obarvit jinou barvou než tu předchozí.

1.2 Formát vstupních dat

Formát vstupních dat je daný zadáním:

- Na prvním řádku je číslo n, určující počet hran grafu G
- Na následujících n řádcích je vždy n číslic 0 nebo 1, které reprezentují matici sousednosti grafu – 1 znamená, že odpovídající uzly mezi sebou mají hranu, 0 že nemají.

1.3 Formát výstupních dat

Formát výstupních dat má následující strukturu:

```
Best bipartite graph edge count:
================
==   25 edges ==
================
Bipartite subsets:
0) 0  4  6  7  9  14  1  11  3
1) 2  8  10  12  13  15  5

Edges in bipartite subgraph:
0 <-> 2
```

```
0 <-> 8
(...)


Computation time:
13.0867
```

Kde:

- Ohraničená hodnota udává počet hran nalezeného grafu,
- množiny čísel pod `Bipartite subsets` ukazují, do které ze dvou množin ten který uzel patří (tj. kterými barvami jsou uzly obarveny),
- seznam dvojic čísel udává, mezi kterými uzly jsou ve výsledném grafu hrany,
- a nakonec je uveden čas v sekundách, jež byl potřeba pro dokončení výpočtu.

1.4 Popis sekvenčního algoritmu

Algoritmus řešící problém MBG sekvenčně využívá prohledávání do hloubky, případně do šířky. Detailní popis těchto metod není předmětem tohoto předmětu, jelikož už byly popsány dříve (BI-PA2, BI-ZUM, BI-EFA, BI-GRA).

K reprezentaci uzlů prohledávaného stavového prostoru jsem zvolil matici sousednosti. Z hlediska implementace to znamená to, že každý stav je reprezentován třídou `Graph`, jež si uchovává vlastní kopii dvourozměrného pole – matici sousednosti. Kromě toho má některé pomocné metody, například pro ověření, zda je tento graf bipartitní, spojitý, či nespojitý (metoda vrací hodnoty 1, 0, resp. -1 jako příznak těchto vlastností).

1.4.1 Generování stavů

V každém kroku DFS je nutné vygenerovat potomky řešeného stavu (grafu), které se vloží do zásobníku a budou dále zpracovávat. Algoritmus, kterým jsem se rozhodl tyto sousedy-potomky generovat zajišťuje, že množina vygenerovaných podgrafů bude disjunktní – tedy že žádné dva grafy nebudou totožné, což by v případě jednoduchého odebírání hran nastávalo.

Algoritmus generování funguje tak, že ze zadaného grafu postupně odebírá hrany, čímž generuje podgrafy. Hrany jsou odebírány následujícím způsobem – každý graf má indexy `startI` a `startJ` do matice sousednosti. Ty určují, od které pozice v matici dále budou odebírány hrany (tj. měněny jedičky na nuly). Na předchozí hrany algoritmus nesmí sahat.

Vygenerování množiny potomků stavu s grafem `G` pak znamená:

```
given graph G:

neighbors = []
foreach (i,j)>(startI,startJ):
        if edge(i,j) is present:
                new_graph = G.remove_edge(i,j)
                new_graph.startI = i
```

```
                new_graph.startJ = j
                neighbors.append(new_graph)
```

Tímto způsobem generuji stavy, které může mít dále smysl prohledávat.

1.4.2 Ořezávání stavového prostoru

Před samotným zpracováním vygenerovaných stavů je vhodné provést kontrolu, zda to má vůbec smysl:

- Pokud je graf bipartitní, nemá smysl prohledávat jeho potomky – budou mít určitě menší počet hran a tento graf bude tedy zaručeně lepší.
- Pokud má graf stejný nebo menší počet hran než dosud nejlepší nalezený graf, nemá smysl ho dále prohledávat.
- Pokud je vygenerovaný graf nespojitý, nemá smysl se s ním dále zabývat - zajímají mě pouze spojité grafy. Test spojitosti provádím v rámci testu bipartity, viz popis reprezentace uzlu výše.
- Pokud má graf méně hran než $|V| - 1$, nemá smysl se s ním zabývat, jelikož toto je spodní mez řešení.

1.5 Naměřené časy

Časy naměřené na vzorových datech na lokálním stroji (frekvence CPU 2.4GHz) jsou následující:

- `graph10_5.txt` – 1020 ms
- `graph17_3.txt` – 10 ms
- `graph20_3.txt` – 120 ms
- `graph25_3.txt` – 130 ms
- `graph14_4.txt` – 1650 ms

2 Popis paralelního algoritmu a jeho implementace v OpenMP

Paralelní algoritmus v OpenMP, tj. řešení se sdílenou pamětí, jsem řešil dvěma způsoby – pomocí task paralelismu a datového paralelismu.

Task paralelismus jsem řešil jednoduše vytvářením nového `OMP Task` při každém rekurzivním volání DFS funkce. Až na toto tvoření tasků a uzamykání aktualizace nejlepšího řešení do kritické sekce, bylo toto řešení totožné se sekvenčním.

Datový paralelismus byl implementačně o něco náročnější než *Task paralelismus*. Pro jeho korektní běh bylo nejprve nutné vygenerovat dostatečně velkou množinu počátečních (disjunktních) stavů, nad kterou poté budou OpenMP vlákna iterovat. Generování počátečních stavů jsem extrahoval do samostatné funkce, která ale vnitřně funguje velmi podobně běžnému BFS algoritmu. Prohledávání probíhá stejně jako u "ostrého" řešení s tím, že ve chvíli, kdy velikost fronty grafů čekajících na zpracování odpovídá zadanému parametru, algoritmus na její konec vloží nedozpracovaný graf (abych nepřišel o žádný ze stavů) a celou frontu vrátí jako výsledek

funkce. Nad touto frontou grafů pak standardně pomocí `#pragma omp parallel for` spustím požadovaný počet vláken.

Tím, že vlákna mají disjunktní množinu počátečních stavů, které jsou na sobě nezávislé, nemusí na sebe nijak čekat (vyjma kritické sekce při nalezení nového maxima).

Počet generovaných stavů jsem empiricky stanovil na 200násobek počtu vláken, ale ideální hodnota tohoto parametru silně závisela na vstupních datech.

2.1 Příkazová řádka

Kompilace a spuštění programu v příkazové řádce se provede následující sekvencí příkazů:

```
$ cd project
$ make clean && make
$ ./solver <input-graph> [seq|omp|mpi] [num_threads]
```

Druhý parametr programu udává:

- **seq** – sekvenční běh.
- **omp** – běh pouze v OpenMP (nepoužívají se volání MPI).
- **mpi** – běh OpenMP + MPI (viz dále).

Popiste paralelni algoritmus, opet vyjdete ze zadani a presne vymezte odchylky, zvlaste u algoritmu pro vyvazovani zateze, hledani darce, ci ukonceni vypoctu. Popiste a vysvetlete strukturu celkoveho paralelniho algoritmu na urovni procesuu v OpenMP a strukturu kodu jednotlivych procesu. Napr. jak je naimplemtovana smycka pro cinnost procesu v aktivnim stavu i v stavu necinnosti. Jake jste zvolili konstanty a parametry pro skalovani algoritmu. Struktura a semantika prikazove radky pro spousteni programu.

3 Popis paralelniho algoritmu a jeho implementace v MPI

Popiste paralelni algoritmus, opet vyjdete ze zadani a presne vymezte odchylky, zvlaste u algoritmu pro vyvazovani zateze, hledani darce, ci ukonceni vypoctu. Popiste a vysvetlete strukturu celkoveho paralelniho algoritmu na urovni procesuu v MPI a strukturu kodu jednotlivych procesu. Napr. jak je naimplemtovana smycka pro cinnost procesu v aktivnim stavu i v stavu necinnosti. Jake jste zvolili konstanty a parametry pro skalovani algoritmu. Struktura a semantika prikazove radky pro spousteni programu.

4 Namerene vysledky a vyhodnoceni

1. Zvolte tri instance problemu s takovou velikosti vstupnich dat, pro ktere ma sekvencni algoritmus casovou slozitost kolem 5, 10 a 15 minut. Pro mereni cas potrebny na cteni dat z disku a ulozeni na disk neuvazujte a zakomentujte ladici tisky, logy, zpravy a vystupy.
2. Merte paralelni cas pri pouziti $i = 2, \cdot, 32$ procesoru na siti Ethernet.

3. Z namerenych dat sestavte grafy zrychleni $S(n, p)$. Zjistete, zda a za jakych podminek doslo k superlinearnimu zrychleni a pokuste se je zduvodnit.

4. Vyhodnodte komunikacni slozitost dynamickeho vyvazovani zateze a posudte vhodnost vami implementovaneho algoritmu pro hledani darce a deleni zasobniku pri reseni vaseho problemu. Posudte efektivnost a skalovatelnost algoritmu. Popiste nedostatky vasi implementace a navrhnete zlepseni.

5. Empiricky stanovte granularitu vasi implementace, tj., stupen paralelismu pro danou velikost reseneho problemu. Stanovte kriteria pro stanoveni mezi, za kterymi jiz neni ucinne rozkladat vypocet na mensi procesy, protoze by komunikacni naklady prevazily urychleni paralelnim vypoctem.

5 Zaver

Celkove zhodnoceni semestralni prace a zkusenosti ziskanych behem semestru.

6 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.

Vypustenim zavorek `figure` dostanete opet pouze ramecek v textu bez cisla a popisu.

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.