

Paralelní algoritmus pro řešení problému maximálního bipartitního podgrafu

Martin Melka

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

May 2, 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 – matice 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 odebrání 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 spojitě 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

Popiste paralelní algoritmus, opet vyjdete ze zadání a přesně vymezte odchylky, zvláště u algoritmu pro vyvazování zátěže, hledání darce, či ukončení výpočtu. Popiste a vysvětlíte strukturu celkového paralelního algoritmu na úrovni procesů v OpenMP a strukturu kódu jednotlivých procesů. Např. jak je naimplementována smyčka pro činnost procesu v aktivním stavu i v stavu nečinnosti. Jaké jste zvolili konstanty a parametry pro skalování algoritmu. Struktura a semantika příkazové řádky pro spouštění programu.

3 Popis paralelního algoritmu a jeho implementace v MPI

Popiste paralelní algoritmus, opet vyjdete ze zadání a přesně vymezte odchylky, zvláště u algoritmu pro vyvazování zátěže, hledání darce, či ukončení výpočtu. Popiste a vysvětlíte strukturu celkového paralelního algoritmu

na úrovni procesu v MPI a strukturu kódu jednotlivých procesů. Např. jak je naimplementována smyčka pro činnost procesu v aktivním stavu i v stavu nečinnosti. Jaké jste zvolili konstanty a parametry pro skalování algoritmu. Struktura a semantika příkazové řádky pro spuštění programu.

4 Namerené výsledky a vyhodnocení

1. Zvolte tři instance problému s takovou velikostí vstupních dat, pro které má sekvencní algoritmus časovou složitost kolem 5, 10 a 15 minut. Pro měření času potřebný na čtení dat z disku a uložení na disk neuvazujte a zakomentujte řádky tisky, logy, zprávy a výstupy.
2. Měřte paralelní čas při použití $i = 2, \cdot, 32$ procesorů na síti Ethernet.
3. Z naměřených dat sestavte grafy zrychlení $S(n, p)$. Zjistete, zda a za jakých podmínek došlo k superlineárnímu zrychlení a pokuste se je zdůvodnit.
4. Vyhodnoďte komunikační složitost dynamického vyvazování zátěže a posuďte vhodnost vami implementovaného algoritmu pro hledání darce a dělení zásobníku při řešení vašeho problému. Posuďte efektivnost a skalovatelnost algoritmu. Popište nedostatky vaší implementace a navrhněte zlepšení.
5. Empiricky stanovte granularitu vaší implementace, tj., stupeň paralelismu pro danou velikost řešeného problému. Stanovte kritéria pro stanovení mezí, za kterými již není účinné rozkládat výpočet na menší procesy, protože by komunikační náklady převážily urychlení paralelním výpočtem.

5 Závěr

Celkové zhodnocení semestrální práce a zkušenosti získaných během semestru.

6 Literatura

A Navod pro vkládání grafu a obrázku do Texu

Nejjednodušší způsob vytvoření obrázku je použití sunovského grafického editoru xfig, ze kterého lze exportovat latex formáty (v pořadí prostý latex, latex s makry epic, eepic, eepicemu) a postscript formáty, uvedené pořadí odpovídá rostoucí komplikovanosti obrázku (postscript umí jakýkoliv obrázek, prostá latex macro pouze jednoduché, epic makra něco mezi, je třeba vyzkoušet). Následují příklady pro všechny případy.

Obrázek v postscriptu, vycentrován a na celou šířku stránky, s popisem a číslem. Všimnete si, jak řídit velikost obrázku.

Vypuštěním závorek `figure` dostanete opět pouze rámeček v textu bez čísla a popisu.

kteří mají meritko rastru obrázku, Tyto příkazy je ale současně nutné vyhodit ze souboru, který xfig vygeneroval.

Pro vytváření grafu lze použít program gnuplot, který umí generovat postscriptový soubor, který vložíte do Texu výše uvedeným způsobem.