Semestrální projekt NI-PDP 2022/2023

Paralelní algoritmus pro hledání minimálního hranového řezu hranově ohodnoceného grafu

Bc. Luboš Zápotočný

FIT ČVUT
Thákurova 9, 160 00 Praha 6

15. května 2023

1 Definice pojmů a popis sekvenčního algoritmu

V této kapitole se budeme věnovat definici problému a popisu sekvenčního algoritmu pro řešení minimálního hranového řezu v hranově ohodnoceném grafu. Nejprve připomeneme základní pojmy týkající se grafů a hranového řezu a poté představíme konkrétní problém, který se v této práci řešil.

Problém minimálního hranového řezu v hranově ohodnoceném grafu spočívá v nalezení dvou disjunktních podmnožin uzlů, takzvaných X a Y, takových, že součet ohodnocení všech hran spojujících uzly z obou množin je minimální. Tento problém má mnoho praktických aplikací, například v oblasti sítí, kde se hledají nejlevnější cesty mezi uzly sítě.

Vstupní data obsahují číslo reprezentující počet vrcholů v grafu a následně reprezentaci ohodnocení hran pomocí matice sousednosti. Ukázka vstupních dat je vyobrazena v ukázce 1.

```
10
     0   112     0     0    98    80     0     0    91   102
   112     0    90     0     0     0     0   119    96     0
     0    90     0     0   104   111    82     0     0   107
     0     0     0     0     0   114    96     0     0     0
    98     0   104     0     0   118    80    88     0     0
    80     0   111   114   118     0   105   106     0   105
     0     0    82    96    80   105     0   109    93    99
     0   119     0     0    88   106   109     0     0    83
    91    96     0     0     0     0    93     0     0    95
   102     0   107     0     0   105    99    83    95     0
```

Kód 1: Vstupní data

Sekvenční algoritmus pro řešení tohoto problému je založen na postupném procházení některých (nikoli všech) kombinací podmnožin uzlů X a Y a hledání takové kombinace, která minimalizuje součet ohodnocení hran mezi nimi.

Některé kombinace algoritmus vynechává, jelikož nemůžou vést k lepšímu řešení, než je aktuálně nalezené minimum. Při průchodu stromem možných kombinací některé podstromy můžeme vynechat. Této metodě se říká Branch and Bound.

```
if (currentWeight >= bestWeight) {
    upperBoundCounter++;
    return;
}
```

Kód 2: Horní řez

```
int lowerBound = graph.cutLowerBound(cut, index);

if (currentWeight + lowerBound >= bestWeight) {
    lowerBoundCounter++;
    return;
}
```

Kód 3: Dolní řez

Princip prořezávání pomocí Branch and Bound spočívá v systematickém prohledávání stromu kombinací s cílem najít optimální řešení s co nejmenším počtem vyhodnocených kombinací. Algoritmus začíná v kořeni stromu a postupně prochází všechny jeho větve, přičemž se snaží minimalizovat horní hranici ceny řešení a zároveň maximalizovat dolní hranici ceny řešení. Toho se dosahuje pomocí prořezávání podstromů, které nemohou obsahovat optimální řešení.

Prořezávání probíhá pomocí dvou technik: dolních a horních odhadů. Dolní odhady se používají k identifikaci podstromů, které neobsahují optimální řešení, a mohou být tedy bezpečně prořezány. Horní odhady se používají k minimalizaci počtu vyhodnocených kombinací tím, že prořezávají podstromy, jejichž řešení by bylo horší než nejlepší nalezené řešení. Ukázka kódu 2 zobrazuje implementaci horního řezu. Ukázka kódu 3 znázorněna podmínka pro řez pomocí dolního odhadu.

Dolní odhad je spočítán tak, že pro každý zatím nepřiřazený uzel grafu v daném mezistavu s částečným řezem vypočteme, o kolik by se váha řezu zvýšila, pokud by tento uzel patřil do X, o kolik by se zvýšila, pokud by tento uzel patřil do Y a vezmeme menší z těchto dvou hodnot a tato minima posčítáme pro všechny nepřiřazené uzly.

Cílem prořezávání stromu kombinací pomocí Branch and Bound je minimalizovat počet vyhodnocených kombinací a najít optimální řešení. Tento algoritmus je velmi účinný pro řešení problémů, kde je prostor kombinací velký a výpočetní čas je omezený. [1]

Rekurzivní části algoritmu jsou vyobrazeny na ukázkách 4 a 5.

Za předpokladu, že je nalezeno lepší řešení, je tato hodnota uložena a rekurzivní prohledávání této cesty ve stromu kombinací je u konce. Kód 6 zobrazuje část rekurzivní funkce, která kontroluje

```
void DFS_BB(const Graph &graph, int maxPartitionSize,
            bool *cut, int count, int index,
            int currentWeight, int &bestWeight);
```

Kód 4: Prototyp rekurzivní funkce

```
// try with this vertex
cut[index] = true;

// try with this vertex (need to extend current cut)
DFS_BB(graph, maxPartitionSize,
       cut, count + 1, index + 1,
       currentWeight + graph.vertexWeight(cut, index, index), bestWeight);

// restore the status of the cut
cut[index] = false;

// try without this vertex (need to extend current cut)
DFS_BB(graph, maxPartitionSize,
       cut, count, index + 1,
       currentWeight + graph.vertexWeight(cut, index, index), bestWeight);
```

Kód 5: Rekurzivní volání

```
if (index == graph.size) {
    if (count != maxPartitionSize) {
        return;
    }

    if (currentWeight < bestWeight) {
        bestWeight = currentWeight;
    }

    return;
}
```

Kód 6: Prototyp rekurzivní funkce

aktuální váhu řezu a aktualizuje globální proměnnou v případě nalezení lepšího řešení.

Popiste paralelni algoritmus, opet vyjdete ze zadani a presne vymezte odchylky, ktere pri implementaci OpenMP pouzivate. 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.

2 Popis paralelniho algoritmu a jeho implementace v OpenMP - datovy paralelismus

Popiste paralelni algoritmus, opet vyjdete ze zadani a presne vymezte odchylky, ktere pri implementaci OpenMP pouzivate. 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 Master-Slave casti. 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 alespon nekolik minut - vice informaci na `http://courses.fit.cvut.cz` v sekci "Organizace cviceni". 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 $i = 2, \cdot, 60$ vypocetnich jader.
3. Tabulkova a pripadne graficky zpracovane namerene hodnoty casove slozitosti měernych instanci behu programu s popisem instanci dat. Z namerenych dat sestavte grafy zrychleni $S(n, p)$.
4. Analyza a hodnoceni vlastnosti paralelniho programu, zvlaste jeho efektivnosti a skalovatelnosti, pripadne popis zjisteneho superlinearniho zrychleni.

5 Zaver

Celkove zhodnoceni semestralni prace a zkusenosti ziskanych behem semestru.

6 Literatura

Reference

[1] Ph.D Ing. Michal Šoch. Ni-pdp: Prohledávání do hloubky, 4. 2. 2022.