Semestralni projekt NI-PDP 2021/2022
Paralelni algoritmus pro reseni problemu

Jaroslav Langer

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

May 2022

1 Definice problému a popis sekvenčního algoritmu

Tento projekt se zabývá řešením problému nalezení maximálního souvislého bipartitního podgrafu pro zadaný graf. Neboli najít podgraf zadaného grafu, který je bipartitní (všechny vrcholy lze obarvit dvěma barvami tak, že každé dva sousedící vrcholy mají různou barvu), souvislý (každý vrchol je dosažitelný z každého vrcholu) a z těch podgrafů které toto splňují, vybrat ten s nejvyšší vahou (nejvyšším součtem vah hran).

Sekvenční algoritmus dostane cestu k souboru s grafem jako první argument. Ten načte do pole struktur `EDGES* Edge`, kde `Edge = {.value, .begin, .end}`, `.value` je váha hrany, `.begin` je menší ze dvou indexů vrcholů, `.end` je index druhého vrcholu. Toto pole je seřazené podle vah hran, aby prořezávání fungovalo rychleji.

Po načtení grafu je otestováno, zda graf není bipartitní spojitý již na vstupu. DFS průchodem se navštíví všechny dosažitelné vrcholy a postupně se obarvují vždy opačnou barvou než soused. V případě, že nenastane konflikt (vrchol není možné obarvit jinou barvou, než sousedy) a v případě, že počet navštívených (obarvených) vrcholů je roven celkovému počtu vrcholů, tak algoritmus skončí a vypíše řešení - všechny hrany jsou řešení a váha tohoto řešení je součtem vah všech hran.

V případě, že řešení není triviální, se nejprve vytvoří stav `State`, který si pamatuje `.edges` - které hrany jsou jsou v tomto stavu obsaženy, `.vertices` - barvy jednotlivých vrcholů, `value` - součet vah hran, `.idx` - index v poli `EDGES`.

Řešením je globální spojový seznam struktur `Max`, která obsahuje `.state` a ukazatel na Max `.prev`. Tuto globální proměnou `MAX`, modifikuje rekurzivní funkce `solve(State* state)`, která nejprve zkontroluje, jestli předaný state není řešením (hodnota musí být větší nebo rovna `MAX->state.value`, a state musí být bipartitní a souvislý), potom zvýší ukazatel `.idx` a state třikrát zkopíruje, první kopii hranu z EDGES na index .idx vloží s barvami 0-1, ve druhé kopii vloží hranu s barvami 1-0, a ve třetí kopii hranu nepoužije. Funkce solve je zavolána pro tyto tři stavy. V případě, že `.idx` ukazuje za poslední hranu, funkce je ukončena. V případě, že by vložení hrany porušilo bipartitnost, funkce solve není volána. Dále je zkontrolováno, že je možné nalézt lepší řešení než doposud nejlepší. K tomuto účelu je předpočítané pole `VALUE_LEFT`, které na každém indexu obsahuje součty zbývající hran pro daný index. V případě, že hodnota stavu plus hodnota `VALUE_LEFT` na indexu toho stavu je menší než hodnota nejlepšího řešení, funkce je ukončena.

Na standardním výstupu program vypíše hodnoty řešení, tedy stejnou maximální hodnotu tolikrát, kolik je různých řešení pro tuto hodnotu. Samozřejmě jde program velmi snadno modifikovat aby ze struktury state nevypisoval jenom .value, ale i pole použítých hran .edges nebo dokonce rozdělení do jednotlivých partit .vertices.

Na chybovém výstupu program vypisuje délku běhu měřenou ve vteřinách. Měření času začíná po vytvoření `EDGES` a `VALUE_LEFT` a končí před uvolňováním zdrojů.

Graph	Sequential	Sequential -O3
10_3	0.000307	0.000131
10_5	0.008851	0.002794
10_6	0.173728	0.071894
10_7	0.290943	0.126110
12_3	0.000338	0.000193
12_5	0.058359	0.024575
12_6	0.271610	0.109427
12_9	592.944564	159.115643
15_4	0.029107	0.009958
15_5	1.062928	0.338359
15_6	17.073909	4.852974
15_8	3552.124058	899.376206

2 Popis paralelního algoritmu a jeho implementace v OpenMP - táskový paralelismus

Paralelní algoritmus vychází ze sekvenčního. V případě, že řešení není triviální, master vlákno odstartuje výpočet, poté každé zanoření `solve(state)`, není provedeno vláknem které ho zavolalo, ale místo toho, je toto volání nová úloha, kterou provede volné vlákno. Z důvodu, že kód že tato úloha není provedena hned, není možné uvolnit prostředky `state`, z toho důvodu je na konci každého solve použita pragma `taskwait`, která uvolní zdroje poté, co všechna (1-3) dceřiné procesy doběhnou. Zapisování do globální proměné `MAX` je v kritické sekci, neboli jenom jedno vlákno může modifikovat nejlepší řešení v jeden čas. Algoritmus má definovanou konstantu `PARALLEL_DEPTH`, která určuje kolik zanoření má být provedeno paralelně, testované grafy fungovali dobře pro překvapivě malé hodnoty. Například pro graf s 14 vrcholy a průměrným stupněm 8, neboli problém se 112 hranami pro hloubku zanoření 4 trval 119 sekund, pro zanoření 10 trval 117 sekund a pro zanoření 30 trval 413 sekund.

3 Popis paralelního algoritmu a jeho implementace v OpenMP - datový paralelismus

Datový paralelismus opět vycházel ze sekvenčního řešení, tentokrát, místo toho, aby funkce `solve` byla puštěna s počátečním stavem, byly stavy nejdřív přegenerované do pole `states` a potom v parallením for cyklu byla funkce `solve` volána s těmito předgenerovanými stavy. Stejně jako u táskového paralelismu bylo zapisování do globalní proměnné `MAX` v kritické sekci. Stejně tak byla definována konstanta `PARALLEL_DEPTH`, která v tomto případě nepřímo určovala kolik stavů bude nagenerováno. Narozdíl od táskového paralelismu zde není žádný taskwait, neboli žádné vlákno nemusí čekat, než jiné dokončí svou práci (s výjimkou kritické sekce).

4 Popis paralelního algoritmu a jeho implementace v MPI

Víceprocesový algoritmus vycházel z paralelního datového. Nejdříve hlavní proces rozešle všem pracujícím procesům kopii `EDGES`. Pracující procesy pošlou prázdné řešení hlavnímu procesu se značkou, že jsou připraveni. Hlavní proces mezitím použije předgenerování stavů z datového paralelismu. Místo toho, aby tyto stavy paralelizoval na úrovni vláken, tak místo toho stavy serializuje do bufferu a posílá připraveným procesům. Neboli iteruje nad předgenerovanými stavy a v každé iteraci for cyklu počká na nejlepší řešení od nějakého procesu a tomuto procesu pošle další stav. Aktualizuje své řešení pomocí příchozího. Pracující procesy používají datový paralelismus, na obdrženém stavu si předgenerují další stavy a ty řeší vícevláknově v paralelním cyklu.

Rozdíl ve spouštění tohoto kódu, je, že se pouští pomocí wrapperu mpirun a to s přpínačem -np který specifikuje kolik procesů má běžet. Je nutné zvolit číslo větší než dva aby algoritmus fungoval správně.

5 Naměřené výsledky a vyhodnocení

Grafy 14 8, 15 7 a 22 5 byly vygenerovány pomocí odkazovaného generátoru

```
./generator -t AD -n HRANY -k STUPEŇ -w80,120
```

Tabulka 1: 14 vrcholů, stupeň 8

Procesy	Vlákna	sequential	parallel task	parallel data	multiprocessing
1	1	307.1			
	2		353.1	307.2	
	4		171.8	168.7	
	8		98.0	119.8	
	16		122.8	224.0	
	20		197.0	255.8	
3	2				213.9
	4				198.9
	8				198.9
	16				200.7
	20				200.0
4	2				153.6
	4				159.5
	8				154.9
	16				151.6
	20				157.6

Všechny výsledky mají obdobný charakter. Paralelní řešení (datové i táskové) s malým počtem vláken je horší, než sekvenční řešení. Se zvyšujícím se počtem vláken se čas zkracuje až na třetinu sekvenčního času. Z nějakého důvodu pro více vláken, nejenže algoritmy nezrychlují, ale dokonce zpomalují a to výrazně. Víceprocesový algoritmus je vždy rychlejší oproti sekvenčnímu. Z nějakého důvodu počet vláken neovlivňuje rychlost algoritmu, ačkoli je to ten stejný kód, který v datovém paralelismu čas zrychluje.

Tabulka 2: 17 vrcholů, stupeň 6

Procesy	Vlákna	sequential	parallel task	parallel data	multiprocessing
1	1	264.9			
	2		445.6	227.5	
	4		162.4	142.6	
	8		120.8	78.4	
	16		149.9	137.2	
	20		186.4	155.2	
3	2				187.9
	4				186.7
	8				192.9
	16				183.2
	20				192.8
4	2				138.4
	4				139.0
	8				139.1
	16				136.2
	20				138.8

Tabulka 3: 22 vrcholů, stupeň 5

Procesy	Vlákna	sequential	parallel task	parallel data	multiprocessing
1	1	383.3			
	2		363.0	370.8	
	4		140.0	160.8	
	8		90.4	83.7	
	16		141.9	157.6	
	20		170.1	163.4	
3	2				324.1
	4				345.0
	8				342.8
	16				349.2
	20				321.2
4	2				274.5
	4				273.8
	8				290.7
	16				292.9
	20				260.0

6 Zaver

Práce popisuje jednotlivé algoritmy a výsledná měření. Ukazuje na místa, která by bylo možné více prozkoumat, například, proč více vláken nezrychluje víceprocesový algoritmus. Předmět je výbornou vstupenkou do světa vícevláknového a víceprocesového programování. Dlouho jsme nepoužíval jazyk C, takže to pro mě byl spíš těžší předmět, nicméně je dobré o tom, vědět, konkrétně datový parallelismus nabízí celkem přímočaré řešení pokud je potřeba, aby kód běžel rychleji.

7 Literatura

1. https://users.fit.cvut.cz/ soch/mi-par/graf_gen/
2. https://hal.inria.fr/hal-02383654/document