Semestrální projekt MI-PDP 2022/2023:

Paralelní algoritmus pro řešení problému

Anna Kapitánová

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

May 17, 2023

1 Definice problemu a popis sekvencniho algoritmu

V rámci semestrální práce jsem řešila problém Minimálního hranového řezu hranově orientovaného ohodnoceného grafu.

Zadání

Vstupní data:

- n = přirozené číslo představující počet uzlů grafu G, $150 > n \geq 10$,
- k = přirozené číslo představující průměrný stupeň uzlu grafu G $\frac{3n}{4} > k \geq 5$,
- G(V,E) = jednoduchý souvislý neorientovaný hranově ohodnocený graf o n uzlech a průměrném stupni k, váhy hran jsou z intervalu [80,120],
- a = přirozené číslo, $5 \leq a \leq \frac{n}{2}$

Úkol: Nalezněte rozdělení množiny uzlů V do dvou disjunktních podmnožin X, Y tak, že množina X obsahuje a uzlů, množina Y obsahuje n-a uzlů a součet ohodnocení všech hran u,v takových, že u je z X a v je z Y (čili velikost hranového řezu mezi X a Y), je minimální.

Moje implementace

Moje implementace algoritmu pro řešení problému minimálního hranového řezu v grafu je následující:

1. Začínám s třídou `Graph`, která reprezentuje vstupní graf. Tato třída obsahuje několik důležitých atributů, jako je počet vrcholů, průměrný stupeň vrcholu, počet vrcholů množiny X a matice váhy hran. Navíc třída `Graph` obsahuje metody pro získání nejlepšího řešení a jeho ceny, počtu rekurzivních volání a také metodu pro provedení minimálního hranového řezu.
2. Pro řešení jsem použila metodu Branch and Bound (B&B) s prohledáváním do hloubky (DFS). Tato metoda je implementována v rekurzivní funkci `solve`, která je volána na každém vrcholu grafu. Funkce se větví na dva "syny" - jeden s hodnotou 0 a druhý s hodnotou 1, což reprezentuje zařazení daného vrcholu do množiny X, nebo ne.
3. Abych zabránila prohledávání větví, které určitě nevedou k optimálnímu řešení, použila jsem ořezávání pomocí dolního odhadu váhy zbývajícího řezu. Dolní odhadu ceny řezu je vypočten pomocí funkce `check_lower_cost_estimation`. Pokud je odhadovaná cena vyšší než nejlepší dosud nalezené řešení, daná větev je zahozena.
4. Když prohledávání dorazí k poslednímu vrcholu (tj. všechny vrcholy byly prozkoumány), ověřím, zda je aktuální řešení lepší než dosud nejlepší. Pokud ano, aktuální řešení se stane novým nejlepším.
5. Na konci program vypíše nejlepší řešení, jeho cenu a počet rekurzivních volání.

Spuštění programu

Při spuštění programu jsou vstupní parametry následující:

1. Název souboru obsahujícího data o grafu.
2. Počet vrcholů množiny X (označeno jako `a`).

```
# Příklad spuštění programu:
./program graph_data.txt 5
```

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

Začátek paralelního procesu je definován v bloku `omp parallel shared`. Tento blok inicializuje paralelní sekci, kde jsou proměnné `best_price`, `best_sol`, `calls` a `X_vertices` sdíleny mezi vlákny.

V rámci tohoto bloku je sekce `omp single`, která zajistí, že funkce `solve` je zavolána pouze jednou. Tato funkce pak tvoří úkoly `omp task` rekurzivním zavoláním sama sebe, pokud se nachází pod maximální hladinou. Maximální hladina je stanovena podle složitosti úlohy jako `max_level = k * level_coef`, kde `k` je průměrný stupeň uzlu a `level_coef` je určen parametrem.

Pokud se funkce `solve` nachází nad maximální hladinou, je zavoláno sekvenční řešení pomocí funkce `solve_seq`, která byla použita v předchozí části implementace.

Spuštění programu

Při spuštění programu jsou vstupní parametry následující:

1. Název souboru obsahujícího data o grafu.
2. Počet vrcholů množiny X (označeno jako `a`).
3. Počet vláken použitých pro paralelizaci (`threads`).
4. Koeficient pro určení maximální hladiny paralelismu (označeno jako `level_coef`).

```
# Příklad spuštění programu:
./program graph_data.txt 5 10 2
```

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

Počáteční stavy jsou generovány pomocí BFS, kde každý vygenerovaný stav je vložen do fronty. Když je počet stavů ve frontě roven `max_initial_states`, generování stavů je zastaveno, stavy jsou přesunuty z fronty do vektoru a algoritmus přechází k paralelnímu zpracování těchto stavů.

V další části algoritmu je volán paralelní for cyklus `omp parallel for schedule(dynamic)`, který řeší stavy pomocí sekvenční funkce `solve_seq`, která byla popsána výše. V průběhu prohledávání stavového prostoru je postupně pro každý stav vytvářena jeho řešení a jsou porovnávány s aktuálním nejlepším řešením. Pokud je nové řešení lepší, stává se aktuálním nejlepším řešením. To je zajištěno použitím direktivy `#pragma omp critical`, která zajišťuje, že daný kód může být v daný okamžik vykonáván pouze jedním vláknem.

Spuštění programu

Při spuštění programu jsou vstupní parametry následující:

1. Název souboru s daty grafu.
2. Počet vrcholů množiny `X` (`a`).
3. Počet vláken použitých pro paralelizaci (`threads`).
4. Počet počátečních stavů vytvořených při inicializaci fronty stavů (`init_states_number`).

```
# Příklad spuštění programu
./program graph_file.txt 5 4 10
```

4 Popis paralelního algoritmu a jeho implementace v MPI

Hlavní (master) proces vygeneruje počáteční stavy algoritmem BFS (prohledávání do šířky). Těchto stavů generuje počet rovnající násobku koeficientu `master_states_coef` (zadaný parametrem) a proměnné `slave_n` (vypočtenné jako počet procesů minus jedna). Tyto počáteční stavy jsou zařazeny do fronty, představující počáteční úlohy pro slave procesy.

Master proces poté rozešle tyto úlohy slave procesům prostřednictvím funkce MPI_Send s označením zprávy (TAG_TASK). Následně master proces přijímá zprávy od slave procesů, dokud jsou nějaké dostupné. Pokud zpráva obsahuje označení úspěšně dokončené úlohy (TAG_DONE), je její výsledek porovnán s aktuálním globálním minimem. Pokud je nalezeno lepší řešení globální řešení je aktualizováno.

Pokud je fronta prázdná a nejsou další úlohy pro zpracování, master proces rozešle zprávu s označením pro ukončení slave procesu (TAG_KILL). Pokud však fronta prázdná není, je další úloha v frontě rozeslána slave procesu. Oproti základnímu řešení jsem navíc implementovala to, že master při posílání dalších stavů volným slave procesům posílá zároveň aktualizované globální řešení. Toto řešení si patřičné slave procesy nastaví jako svoje počáteční řešení.

Každý slave proces přijímá zprávy od master procesu prostřednictvím funkce MPI_Recv, a to v nekonečné smyčce. Pokud přijme zprávu s označením pro ukončení (TAG_KILL), proces se ukončí. Pokud však zpráva obsahuje úlohu, slave proces ji zpracuje pomocí datového paralelismu prostřednictvím OpenMP (jak bylo popsáno výše). Po dokončení úlohy slave proces rozešle zprávu s výsledkem zpět master procesu. Všechny zprávy jsou posílány jako struktura Message.

Spuštění programu

Při spuštění programu jsou vstupní parametry následující:

1. Název souboru s daty grafu.
2. Počet vrcholů množiny `X` (`a`).
3. Počet vláken použitých pro paralelizaci (`threads`).
4. Počet počátečních stavů ve slave procesech (`slave_states_number`).
5. Koeficient, použitý pro výpočet počtu počátečních stavů vytvořených při inicializaci fronty stavů (`master_states_coef`).

```
# Příklad spuštění programu
./program graph_file.txt 5 5 5 10
```

5 Naměřené výsledky a vyhodnocení

Následující sekce se věnuje výsledkům měření jednotlivých programů. Nejlépe vycházejí výsledky při použití implementace pomocí MPI. Paralelismus pomocí OpenMP vychází lépe v případě datového paralelismu.

5.1 Sekvenční řešení

Tabulka 1 zobrazuje naměřené časy sekvenčního řešení, tabulka obsahuje prvních 8 úloh, další úlohy už nebyly výpočetně proveditelné v daném časovém rozsahu – do 10 minut. Pro další měření byly vybrány soubory `graf_30_10 a=10`, `graf_30_10.txt a=15` a `graf_30_20.txt a=15`.

Soubor	Hodnota a	Sekvenční čas
graf_10_5.txt	5	0,014 s
graf_10_6b.txt	5	0,015 s
graf_20_7.txt	7	0,486 s
graf_20_7.txt	10	0,871 s
graf_20_12.txt	10	1,840 s
graf_30_10.txt	10	1 min 27,227 s
graf_30_10.txt	15	2 min 56,397 s
graf_30_20.txt	15	5 min 32,435 s

Table 1: Výsledky sekvenčního řešení

5.2 OpenMP

Tabulka 2 zobrazuje výsledky měření obou implementací v OpenMP, jak funkčního tak datového. Datový paralelismus vycházel lépe pro všechny vybrané soubory. Tendence naměřených hodnot metriky paralelního zrychlení je vidět na grafu 1. Vidíme, že s přibývajícími vlákny ze začátku roste zrychlení, s přibývajícím počtem vláken ovšem růst zrychlení začne zpomalovat. U náročnějších souborů by zřejmě došlo ještě k radikálnějšímu zpomalení, jak naznačují výsledky času běhu náročnějších programů. U těchto náročnějších programů ovšem nemáme srovnání se sekvenčním algoritmem, neboť nejsou v jeho výpočetní kompetenci, zde je tudíž neuvádím.

graf_30_10.txt, a=10						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	Funkční p.	Datový p.	Funkční p.	Datový p.	Funkční p.	Datový p.
2	68,380	23,618	1,276	3,691	0,638	1,847
4	48,381	14,279	1,803	6,108	0,451	1,533
6	36,342	13,790	2,399	6,326	0,400	1,258
8	33,384	11,866	2,614	7,347	0,327	1,155
12	27,759	12,819	3,144	6,805	0,262	0,571
16	27,541	12,866	3,168	6,772	0,198	0,536
20	26,993	13,141	3,232	6,638	0,162	0,333
graf_30_10.txt, a=15						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	Funkční p.	Datový p.	Funkční p.	Datový p.	Funkční p.	Datový p.
2	103,917	95,943	1,697	1,836	0,848	0,917
4	93,289	57,653	1,891	3,059	0,473	0,766
6	64,866	56,099	2,719	3,144	0,453	0,522
8	65,167	51,010	2,706	3,456	0,338	0,434
12	63,199	53,050	2,791	3.325	0,232	0.277
16	67,180	51,882	2,627	3.399	0,164	0.212
20	65,924	51,865	2,675	3.401	0,134	0.170
graf_30_20.txt, a=15						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	Funkční p.	Datový p.	Funkční p.	Datový p.	Funkční p.	Datový p.
2	205,677	37,667	1,616	8,818	0,808	4,420
4	167,742	23,055	1,982	14,416	0,495	3,611
6	119,306	20,921	2,787	15,898	0,465	2,816
8	108,963	19,561	3.050	16,957	0.381	2,129
12	94,044	20,865	3.534	15,937	0.294	1,413
16	88,896	20,521	3.739	16,195	0.233	1,016
20	90,452	20,616	3.675	16,128	0.183	0,805

Table 2: Výsledky OpenMP

5.3 MPI

Výsledky implementace MPI byla měřena počet procesů 3 (1x Master, 2x Slave) a č (1x Master, 3x Slave). Rovněž byly výsledky srovnávány pro různé počty vláken (2, 4, 6, 8, 12, 16, 20). Bylo měřeno paralelní zrychlení a paralelní efektivnost ve srovnání s výsledky sekvenčního algoritmu. Tabulka 3 zobrazuje naměřené výsledky implementace v MPI, je v ní vidět klesající charakter paralelní efektivnosti s přibývajícím počtem vláken. Z výsledků je vidět menší tendence změny rychlosti algoritmu se zvyšujícím se počtem vláken (u velkého množství vláken začne čas opět nepatrně narůstat). Domnívám se, že to může být zapříčiněno zvýšenou nutnou řežií při větším počtu vláken. Implementace pomocí knihovny MPI byla totiž zdaleka nejrychlejší ve srovnání s implementacemi s OpenMP a sekvenčním řešením, jak je mimo jiné vidět z naměřeného paralelního zrychlení. Pro první dvě úlohy na menším grafu vychází lépe řešení MPI 4, u posledního většího grafu však vychází nepatrně lépe řešení s 3 procesy.

6 Závěr

V rámci semestrální práce bylo implementováno několik přístupů k paralelizaci pomocí knihoven OpenMP a MPI nad úkolem nalezení minimálního hranového řezu hranově ohodnoceného grafu. Nejprve byla vytvořena implementace pomocí sekvenčního algoritmu. Naměřené výsledky tohoto algoritmu byly brány jako srovnávací baseline pro naměřené výsledky dalších algoritmů. Sekvenční řešení v některých zadáních nebylo schopno dopočítat řešení v nějakém měřitelném časovém úseku. Proto pro následující srovnání byly vybrány tři úlohy, které byly dopočítány sekvenčním algoritmem v čase mezi 1 minutou a 10 minutami. Následně byly implementovány dva způsoby paralelizace s pomocí knihovny OpenMP. Jednalo se o datový a taskový paralelizmus. Na základě měření bylo ukázána převaha implementace datového paralelizmu. Tato implementace byla tak použita v implementaci paralelizace pomocí knihovny MPI.

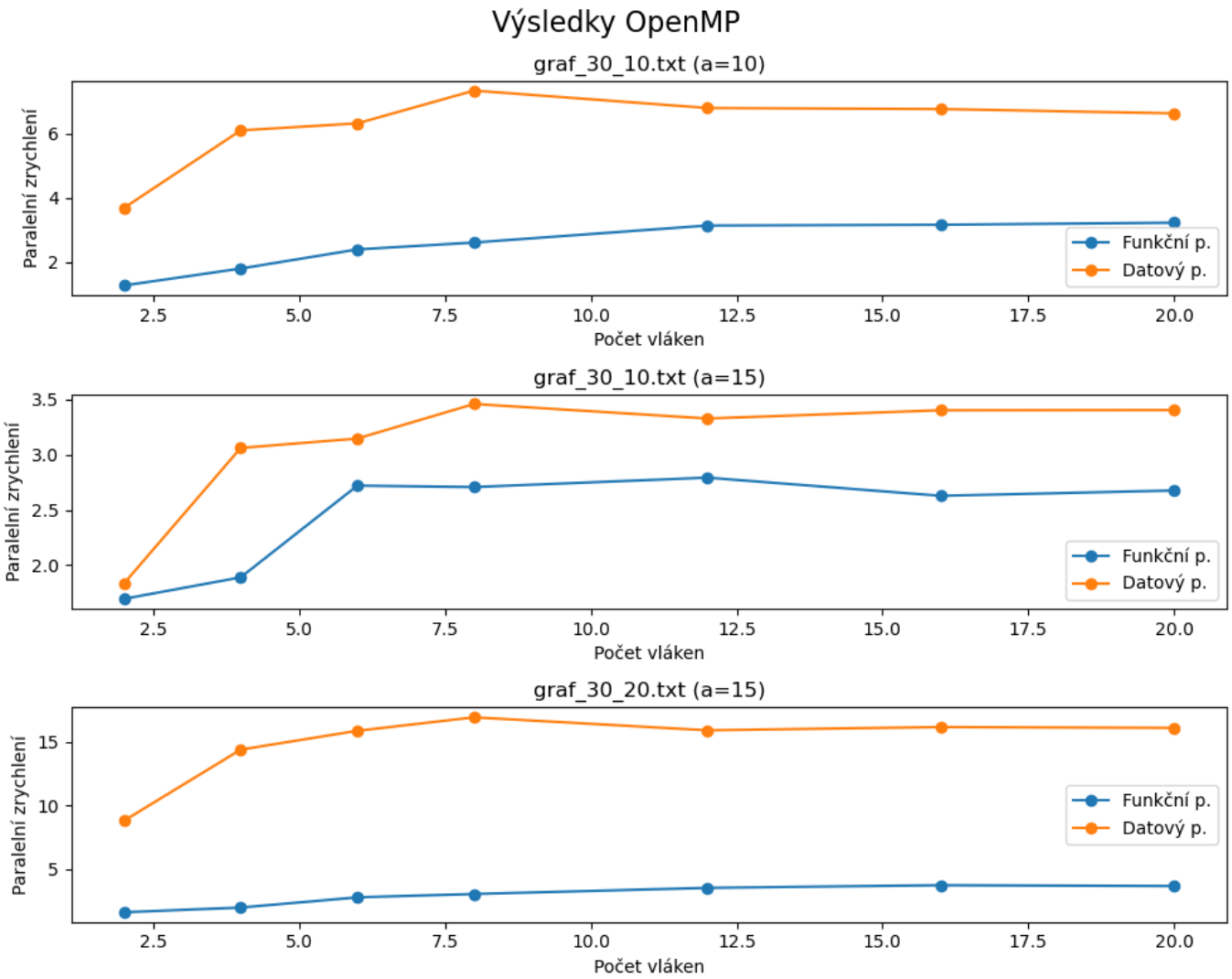

Figure 1: Srovnání paralelního zrychlení v OpenMP

graf_30_10.txt, a=10						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	MPI 3	MPI 4	MPI 3	MPI 4	MPI 3	MPI 4
2	3.339	2.552	26.13	34.16	13.08	17.08
4	2.898	3.422	30.08	25.48	7.54	6.37
6	3.788	4.778	23.00	18.23	3.83	3.04
8	4.688	5.423	18.58	16.07	2.32	2.01
12	2.644	2.683	32.97	32.50	2.30	2.71
16	2.642	2.586	32.99	33.78	2.06	2.11
20	3.113	2.628	27.97	33.15	1.40	1.66
graf_30_10.txt, a=15						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	MPI 3	MPI 4	MPI 3	MPI 4	MPI 3	MPI 4
2	7.003	6.121	25.17	28.83	12.59	14.42
4	5.984	6.672	29.45	26.43	7.36	6.61
6	7.085	6.828	24.88	25.83	4.14	4.31
8	7.356	8.167	23.97	21.61	2.99	2.70
12	5.366	5.775	32.84	30.55	2.29	2.54
16	5.420	5.191	32.53	33.92	1.35	2.12
20	5.577	5.233	31.65	33.71	1.26	1.68
graf_30_20.txt, a=15						
Metrika	Čas [s]		Paralelní zrychlení		Paralelní efektivnost	
Počet vláken	MPI 3	MPI 4	MPI 3	MPI 4	MPI 3	MPI 4
2	8.879	8.856	37.45	37.55	18.72	18.77
4	8.389	8.984	39.62	36.97	9.90	9.24
6	8.755	10.032	38.10	33.10	6.32	5.52
8	9.188	10.479	36.18	31.72	4.52	3.97
12	7.840	8.311	42.38	39.93	3.70	3.32
16	7.816	7.799	42.57	42.67	2.66	2.67
20	8.058	7.965	41.21	41.80	2.06	2.09

Table 3: Výsledky MPI