Semestrální projekt MI-PDP 2016/2017:

Paralelní algoritmus pro řešení problému - zobecněná bisekční šířka

Jaroslav Veselý

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

10. května 2017

1 Definice problému

1.1 Vstupní data

- n = přirozené číslo představující počet uzlů grafu G, $n >= 10$
- k = přirozené číslo řádu jednotek představující průměrný stupeň uzlu grafu G, $n >= k >= 3$
- $G(V, E)$ = jednoduchý souvislý neorientovaný neohodnocený graf o n uzlech a průměrném stupni k
- a = přirozené číslo, $5 =< a =< n/2$

1.2 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 počet 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.

1.3 Výstup algoritmu

Výpis disjuktních podmnožin uzlů X a Y a počet hran tyto množiny spojující.

2 Popis sekvenčního algoritmu

Program očekává 2 vstupní parametry. První je cesta k souboru s grafem, druhý je konstanta a.

Sekvenční algoritmus je založen na generování všech možných rozdělení grafu do dvou disjunktních podmnožin. Následném výpočtu velikosti hranového řezu a uchovávání minimálního řešení.

Generování probíhá postupným rozhodováním, zda uzel bude v množině X nebo Y, následně se u obou možností dále rozhoduje o uzlech, které ještě nebyly rozřazeny. Větvení končí, pokud je celé možné rozdělení nebo je zřejmé, že již negenerujeme přijatelné řešení (překročení hodnoty a).

3 Popis paralelního algoritmu a jeho implementace v OpenMP

Oproti sekvenční variantě program umožňuje zadat třetí parametr - počet vláken (výchozí hodnota je 1).

Algoritmus nejprve jednovláknově vygeneruje uzly (podproblémy) rozhodovacího stromu (po "patrech") tak, aby počet těchto uzlů byl dostatečně velký. Pro počet vláken p se vygeneruje alespoň p^2 problémů. Následně je použita direktiva $parallel for$ s dynamickým plánováním, které vykazovalo nejlepší časové výsledky. Pravděpodobně z důvodu různorodosti ořezávání generovaného rozhodovacího stromu.

4 Popis paralelního algoritmu a jeho implementace v MPI

Kromě parametrů pro MPI program očekává stejné parametry jako v předchozí sekci.

Nejprve se inicializuje MPI, načte se graf a master MPI proces začne generovat podproblémy stejným postupem jako při datové paralelizaci v předchozí sekci. Následně se odešlou problémy jednotlivým slavům a master očekává neblokující příjem výsledků. Dále se dostane do cyklu, ve kterém počítá jednovláknově (DFS pomocí zásobníku) další z vygenerovaných podproblémů a po určitém počtu cyklů výpočtu (100) se provede kontrola pomocí $MPI\_Testsome$, obslouží se slavové, kteří dopočítali a pokračuje výpočet.

Pokud zbývá vypočítat méně podproblému než je MPI procesů, master již jen obsluhuje, jelikož počítá jednovláknově, tedy i pomaleji. Po rozeslání všech podproblémů odešle tag značící konec a čeká na poslední běžící výpočty a protože je použit neblokující příjem, čekající cyklus obsahuje $usleep(1000)$ - "uspání"na 1s, aby se nejednalo o aktivní čekání. V průběhu si master uchovává nejlepší dosažené řešení, které po skončení výpočtu vypíše.

Slavové v nekonečné smyčce čekají na problém, počítají s použitím datového openMP paralelismu a odešlou výsledek. Končí jen pokud přijmou tag značící konec.

5 Naměřené výsledky a vyhodnocení

Pro měření byla použita rozvržení výpočetních jader z tabulky 1.

# MPI proc.	# vláken	# výp. jader
1	1	1
2	1	2
2	2	4
4	2	8
4	4	16
4	6	24
4	8	32
6	10	60

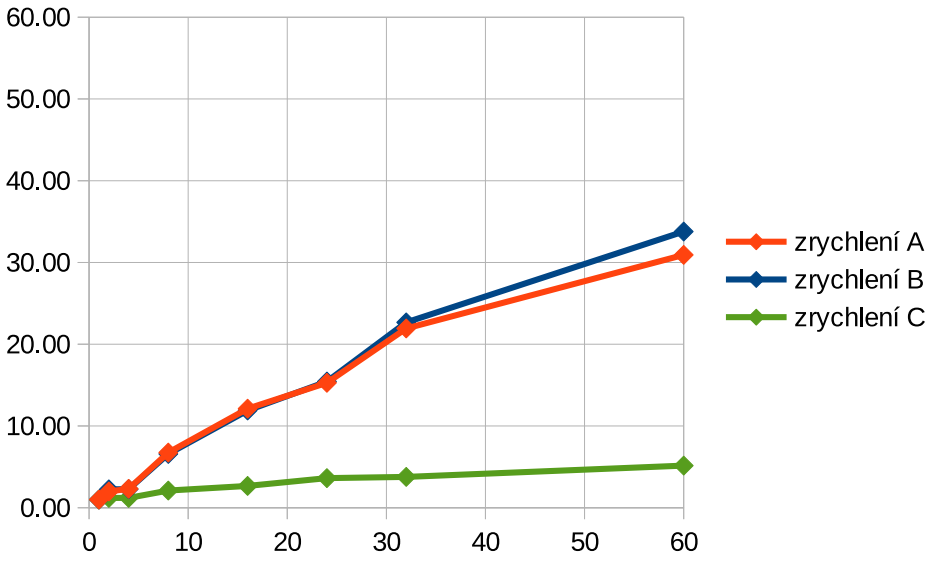
Tabulka 1: Rozvožení jader

Dále byly použity instance a parametry a z tabulky 2

vstupní graf (star)	a	pracovní označení
/home/courses/PDP/graph30_5.txt	15	A
/home/courses/PDP/graph30_7.txt	15	B
/home/courses/PDP/graph50_6.txt	7	C

Tabulka 2: Rozvožení jader

Naměřené výsledky jsou znázorněny grafem 1 a hodnoty jsou v tabulce 3, kde je vidět čas v sekundách a zrychlení oproti sekvenčnímu řešení.



Obrázek 1: Zrychlení

	A		B		C	
# výp. jader	čas (s)	zrychlení	čas (s)	zrychlení	čas (s)	zrychlení
1	475.816	1.00	501.482	1.00	407.395	1.00
2	238.753	1.99	222.779	2.25	337.985	1.21
4	205.798	2.31	221.138	2.27	339.053	1.20
8	70.5415	6.75	76.2609	6.58	193.639	2.10
16	39.2666	12.12	42.1786	11.89	152.764	2.67
24	31.1445	15.28	32.4943	15.43	112.409	3.62
32	21.7006	21.93	22.1292	22.66	107.998	3.77
60	15.3858	30.93	14.8434	33.78	79.1446	5.15

Tabulka 3: Naměřené výsledky

K nejlepšímu zrychlení dochází u 2 jader, to lze vysvětlit drobnou rozdílností algoritmu pro sekvenční řešení a pro 2 MPI procesy. Pokud jsou 2 MPI procesy, jeden je slave a druhý master. Master počítá pomocí stacku a slave používá postup pro datový paralelismus, kdežto sekvenční řešení je čistě rekurzivní.

Záhadou však zůstává znatelně malé zrychlení u instance C. Ořezávání rozhodovacího stromu je totožné u všech algoritmů při libovolných parametrech. Pro rozdělování

problémů platí totéž. Vztah velikosti grafu a parametru a má vliv na čas, ale neměl by mít vliv na takto znatelné snížení zrychlení. Jediným vysvětlením by mohlo být pomalé posílání většího objemu dat mezi MPI procesy, ale objem zprávy je maximálně $sizeof(MPI\_C\_BOOL).graphSize$, kde $graphSize$ je počet uzlů grafu.

6 Závěr

Semestrální práce se dle mého názoru v rámci možností velmi vydařila. To hlavně díky základnímu předpokladu, že paralelizace znatelně, i když ne ideálně, zrychluje reálný čas výpočtu. A také se podařilo přimět master MPI proces, aby ve volném čase mezi zadáváním práce otrokům pracoval.

Narazil jsem na dostatek problémů, abych se na nich naučil základy paralelizace, ale nebylo jich příliš mnoho na to, aby mě tento předmět znechutily. S výsledky i nabytými zkušenostmi jsem osobně spokojen.

7 Literatura