České vysoké učení technické v Praze
Fakulta informačních technologií

Semestrální práce

NI-PDP

Tomáš Kalabis

28. dubna 2022

s

Obsah

1 Definice problému a popis sekvenčního algoritmu

Jako semestrální práci jsem měl zadanou úlohu **MBP: bipartitní podgraf s maximální váhou**. Úkolem je nalézt podmnožinu hran **F** takovou, že podgraf **G**(**V**,**F**) je souvislý a bipartitní a váha **F** je maximální v rámci všech možných bipartitních souvislých podgrafů **G** nad **V**. Přičemž bipartitní graf je ten, jemuž lze množinu uzlů **V** rozdělit na disjunktní podmnožiny **U** a **W** tak, že každá hrana v **E** spojuje uzel z **U** s uzlem z **W**. Tato úloha byla omezena na takové grafy, jež mají váhy hran z intervalu $< 80, 120 >$.

Tato úloha byla řešena sekvenčně následovně. Nejprve jsem označil všechny vrcholy grafu *nedefinovaný* status. První vrchol jsem označil jako *část 1*. Následně jsem pomocí stromové rekurze přidával nebo nepřidával hrany a označoval vrcholy jako *část 1* nebo *část 2*. Pro co největší urychlení výpočtu jsem vytvářel takové podgrafy, které bipartitní můžou být. Tedy pokud jsem přidal hranu která již sousedila s částí 2, druhý vrchol jsem nutně obarvil částí 1 a nezkoušel jinou možnost. Druhým zrychlením bylo, že jsem hranu která měla oba vrcholy již validně obarvené, přidal a nezkoušel ji nepřidat. Třetím zrychlením bylo přidání metody ořezávaní *branch and bound*. Ta spočívala v principu zapamatování si dosavadní nejlepší konfigurace podgrafů. Pokud se při výpočtu narazí na stav, který již nemá šanci zlepšit dosavadní maximum, pak se tento podstrom stavů přeskočí.

Dalším případným zrychlením může být seřazení hran v sestupném pořadí podle váhy, aby maximální konfigurace byla nalezena co nejdříve a ořezávání bylo tak co nejefektivnější. To už ale má implementace nezahrnuje.

Implementace počítá všechny instance, které jsou v nad-složce `graf_mbp` přičemž každá instance musí být popsána souborem

`graf_<počet vrcholů grafu>_<průměrný stupeň uzlu grafu>.txt`.

Výstupem je pak maximální váha, počet těchto řešení a informační údaje jako čas výpočtu či počet rekurzí.

Pro demonstraci výsledků jsem vygeneroval instance pro výpočet. Jejich čas výpočtu je zachycen v následující tabulce.

instance	čas	počet vláken	počet procesů
graf_15_8.txt	116.05	1	1
graf_18_7.txt	227.07	1	1
graf_21_6.txt	145.42	1	1

2 Taskový paralelismus

Jedním ze zkoušených způsobů vícevláknového výpočtu problému *MBP* byl *task paralelismus* s pomocí knihovny *OpenMP*. Task paralelismus jsem řešil obalování značné části kódu v rekurzivní funkci jako `omp task`. Task byl přidělen jinému vláknu, každé 15 zanoření, aby nevznikala zbytečně velká režie.

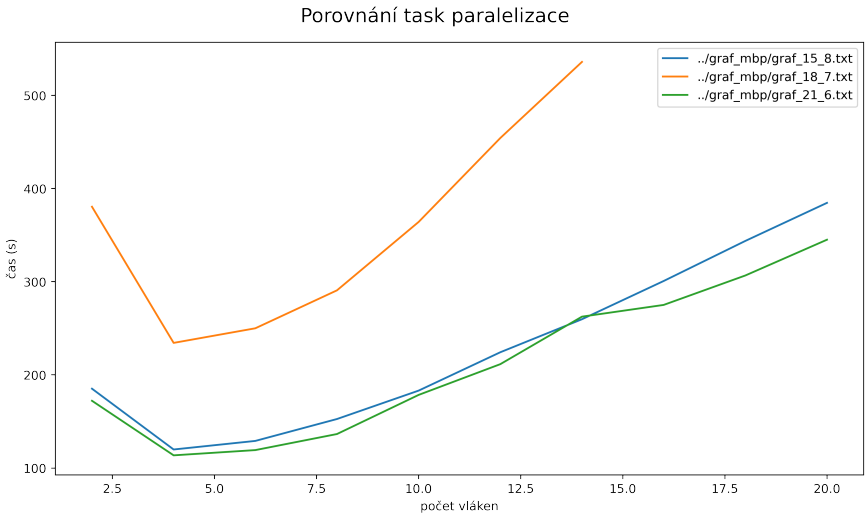
U paralelizace obecně bylo důležité zajistit, aby se počítač vyvaroval časově závislým chybám. Proto se sekce, kde se přepisuje dosavadní maximu zaobalila jako kritická sekce. To se provedlo pomocí příkazu `#pragma omp critical`.

Velkým zlepšením mé implementace by mohlo být ukládání proměnných potřebných pro rekurzivní podinstance dynamicky. To by umožnilo zbavení se nutnosti `#pragma omp taskwait`. Dalším zlepšením mé implementace by mohlo být experimentování s nastavovanou hodnotou a jejím optimálním nastavením. Posledních n zanoření by také mohlo dokončit tentýž vlákno.

V následující tabulce je zachycen čas výpočtu na instancích v závislosti na přidělených jádrech. Byly použiti stejné instance, jako při sekvenčním řešení.

instance	čas (s)	počet vláken	počet procesů
graf_15_8.txt	185.194	2	1
graf_15_8.txt	119.916	4	1
graf_15_8.txt	152.628	8	1
graf_15_8.txt	300.773	16	1
graf_15_8.txt	384.658	20	1
graf_18_7.txt	380.512	2	1
graf_18_7.txt	234.348	4	1
graf_18_7.txt	290.757	8	1
graf_21_6.txt	172.210	2	1
graf_21_6.txt	113.647	4	1
graf_21_6.txt	136.511	8	1
graf_21_6.txt	275.099	16	1
graf_21_6.txt	345.112	20	1

Pro lepší přehlednost jsem zobrazil výpočetní čas na grafu.



Obrázek 1: Škálování OpenMP task paralelizace

Z výsledků je zjevné, že režie v tomto typu paralelismu vyžaduje příliš.

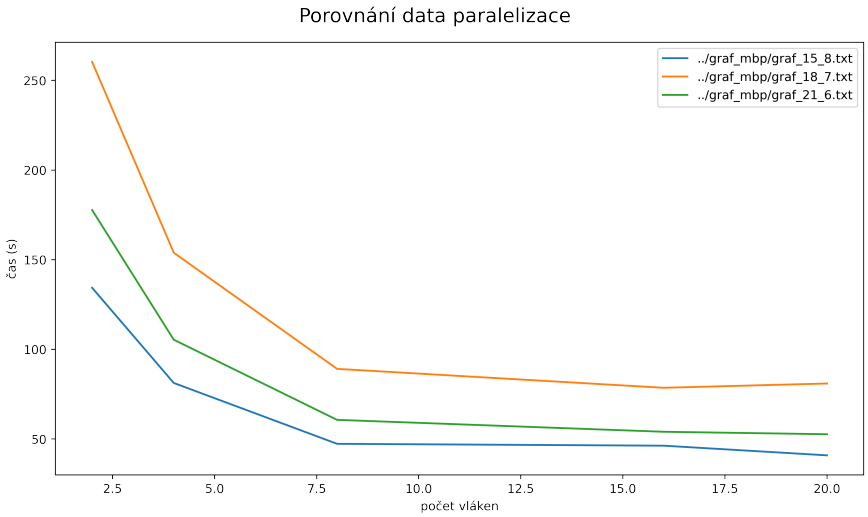
3 Datový paralelismus

Druhým zkoušeným způsobem vícevláknového vícevláknového výpočtu problému *MBP* byl *data paralelismus* opět pomocí knihovny *OpenMP*. Data paralelismus byl řešen následovně. Nejprve se pomocí algoritmu *BFS* získalo n disjunktních stavů. Tyto stavy byly následně předávány jednotlivým vláknům k do-

počítání. Pro předávání stavů byl využit konstrukt `omp parallel for`, který dynamicky přiděloval vláknům úlohy. Jako optimální n se ukázala hodnota 150.

Opět bylo třeba zajistit exkluzivní přístup do proměnné přepisující maximální dosaženou hodnotu vah. To bylo zajištěno opět pomocí `omp critial`.

Oproti task paralelizaci došlo ke zlepšení. Na 20 jádrech je úloha cca 3 krát rychlejší oproti sekvenčnímu. Z grafu lze opět vyčíst, že s vyšším počtem jader se od 8 vlákna příliš nezvětšuje rychlost výpočtu.



Obrázek 2: Škálování OpenMP datové paralelizace

instance	čas výpočtu	počet vláken	počet procesů
graf_15_8.txt	134.350	2	1
graf_15_8.txt	81.274	4	1
graf_15_8.txt	47.339	8	1
graf_15_8.txt	46.298	16	1
graf_15_8.txt	40.918	20	1
graf_18_7.txt	260.309	2	1
graf_18_7.txt	153.900	4	1
graf_18_7.txt	89.133	8	1
graf_18_7.txt	78.588	16	1
graf_18_7.txt	80.974	20	1
graf_21_6.txt	177.718	2	1
graf_21_6.txt	105.388	4	1
graf_21_6.txt	60.699	8	1
graf_21_6.txt	54.057	16	1
graf_21_6.txt	52.689	20	1

4 MPI

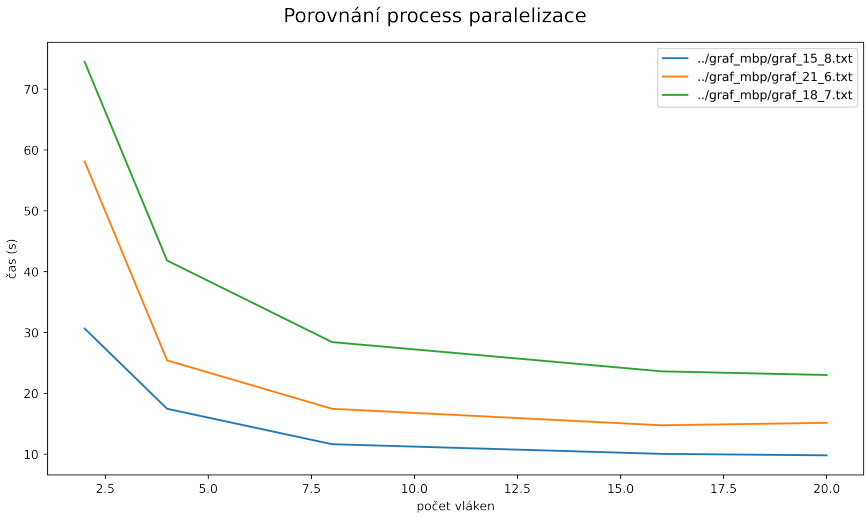
Implementace pro více procesů byla implementována přes knihovnu *OpenMPI*. Implementace byla typu mater/slave. Master zajišťoval běh programu, přijimal a rozesílal úlohy od/k slave procesům a spravoval globálně maximální dosaženou hodnotu vah. Slave procesy naopak přijímaly požadavky k výpočtu od Master procesu. Každý Slave proces prováděl výpočet pomocí *Data paralelismu* blíže popsaným dříve. Master proces nejprve rozeslal všem Slave procesům úlohy a poté ve `while` smyčce čekal na odpověď některého z nich. Jakmile odpověď přišla, Master si uložil případné nové dosavadní maximum a témuž procesu ihned vrátil další úlohu k vypočtení s již aktualizovaným dosavadním maximem. To Master proces opakoval dokud ještě byla práce. Po vypočtení všech úloh bylo rozesláno z Master procesu všem Slave procesům požadavek, aby se ukončili.

Při výpočtu je třeba nastavit dvě konfigurace a těmi jsou počet disjunktních stavů získaných z BFS na úrovni Master procesu a na úrovni Slave procesů. Experimentálně byly nastaveny tyto hodnoty na 50 resp. 200.

Pro vylepšení MPI úlohy může značně pomoci rozesílání nově nalezeného maxima rovnou ostatním Slave procesům i Master procesu. Druhým výrazným zlepšením může být zaměstnání zbývajících jader na Master procesu.

Výsledky měření jsou zachyceny v následující tabulce. V porovnání oproti sekvenčnímu řešení jsou řešení až desetinásobně rychlejší. Z grafu lze však vyčíst, že ke zrychlení přijde relativně rychle a později už je zrychlení zanedbatelné.

instance	čas (s)	počet jader	počet procesů
graf_15_8.txt	30.655	2	4
graf_21_6.txt	58.113	2	4
graf_18_7.txt	74.483	2	4
graf_15_8.txt	17.470	4	4
graf_21_6.txt	25.424	4	4
graf_18_7.txt	41.837	4	4
graf_15_8.txt	11.649	8	4
graf_21_6.txt	17.472	8	4
graf_18_7.txt	28.435	8	4
graf_15_8.txt	10.057	16	4
graf_21_6.txt	14.757	16	4
graf_18_7.txt	23.623	16	4
graf_15_8.txt	9.818	20	4
graf_21_6.txt	15.157	20	4
graf_18_7.txt	23.019	20	4



Obrázek 3: Škálování MPI paralelizace

5 Závěr

Celkově pro mě byla semestrální práce velkým přínosem. Vyzkoušel jsem si implementaci různými způsoby a knihovnami. Byl sem příjemně překvapen snadným používáním knihovny *OpenMP*. Úlohy s touto knihovnou byli velmi snadné. Na druhou stranu použití *OpenMPI* bez nadstavby *Boost* bylo již těžší, zejména debugování.