Semestrální projekt MI-PDP 2022/2023:

Paralelní algoritmus pro řešeni problému hledání bipartitního souvislého podgrafu hranově ohodnoceného grafu s maximální vahou

Pavel Plotnikau

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

Definice problému a popis sekvenčního algoritmu

Problém v této semestrální práci lze popsat následovně: mějme neorientovaný graf G = (V, E) s ohodnocenými hranami, kde V je množina vrcholů a E je množina hran s váhami. Hledáme bipartitní souvislý podgraf G' = (V', E') grafu G takový, že součet vah hran v tom podgrafu je maximální.

Naivní algoritmus hledání řešení je rekurzivní funkce, která ověřuje všechny možné varianty podgrafu. Funkce přijímá jako vstup pole hran, ze kterých se konstruuje podgraf, index aktuální hrany a maximální možnou váhu daného podgrafu. Na začátku mají všechny hrany v poli nedefinovaný stav, index aktuální hrany je 0 a váha podgrafu je rovna váze původního grafu. Algoritmus funkce je následující:

1. Na začátku funkce je provedena kontrola, zda maximální možná váha podgrafu není menší nebo rovna nejlepší dosažené váze. Pokud je to pravda, funkce se ukončí, protože již není potřeba pokračovat v hledání.

2. Pokud index aktuální hrany není roven počtu hran (což znamená, že nebyly prozkoumány všechny hrany), provede se následující:

    2.1. Hrana s aktuálním indexem změní svůj stav na EDGE_NOT_USED (není součástí podgrafu), pak se funkce zavolá znovu s aktuálním podgrafem a indexem další hrany. Přitom se sníží maximální váha tohoto podgrafu o váhu nepoužívané hrany.

    2.2. Dále se stav aktuální hrany změní na EDGE_USED (je použita v podgrafu), a pak se funkce zavolá znovu, ale tentokrát bez snížení maximální možné váhy podgrafu.

3. Pokud index aktuální hrany není se rovná počtu hran znamená to, že byly prozkoumány všechny hrany a je čas vyzkoušet vytvořený podgraf. Výsledný podgraf se pak spolu s jeho váhou předává následující funkci, která pomocí algoritmu BFS zkontroluje jeho souvislost a bipartitnost. Pokud daný graf splňuje všechny podmínky a má vyšší váhu než dříve nalezené řešení, stává se novým řešením a jeho váha je zapsána jako největší.

Pro testování tohoto algoritmu byly vybrány tři instance ohodnocených grafů: graf_10_6.txt, graf_12_5.txt a graf_15_5.txt. Tyto instance byly převzaty z oficiálních webových stránek předmětu na cources.fit.cvut.cz a budou použity ve všech následujících měřeních. Pro spuštění je třeba na konci příkazu zadat soubor, se kterým má program pracovat (*./pdp1 graf_10_6.txt*). Výsledky jsou následující:

	graf_10_6.txt	graf_12_5.txt	graf_15_5.txt
ms	60409	30908	222092

Popis paralelního algoritmu a jeho implementace v OpenMP

Taskový paralelismus

Toto řešení je skoro identické se sekvenčním řešením. Hlavní rozdíl spočívá v tom, že program vytváří nový task pro každé volání rekurzivní funkce. Všechny tasky lze rozdělit do dvou skupin:

- tasky, které vytvářejí nové tasky
- tasky kontrolující získaný podgraf

Byla také přidána kritická sekce při porovnávání současného řešení s nejlepším. Tyto porovnání probíhají na začátku rekurzivní funkce a také na konci kontroly podgrafů.

Při spuštění programu můžete zadat požadovaný počet vláken, se kterými bude algoritmus pracovat. To lze udělat přidáním -t do příkazu, přičemž název souboru musí být zapsán poslední (*./pdp2_1 -t 10 graf_10_6.txt*).

Datový paralelismus

Hlavním rozdílem oproti předchozí verzi paralelního algoritmu je přidání a integrace struktury, ve které je možné uložit aktuální stav podgrafu. Přehled všech možných podgrafů grafu lze představit jako strom, v němž listy budou konečné podgrafy a všechny ostatní uzly jsou mezistavy. Tato verze algoritmu umožňuje uložit mezistav do zvláštního bufferu pro pozdější paralelní výpočet.

Pro to byla také přidána proměnná, která určuje hloubku podstromů, s nimiž bude algoritmus pracovat paralelně. Tento parametr lze definovat při spuštění programu přidáním -d do příkazu. Je také možné definovat požadovaný počet vláken, jako tomu bylo v taskové verzi algoritmu. Název souboru musí být také zadán jako poslední (*./pdp2_1 -t 10 -d 20 graf_10_6.txt*).

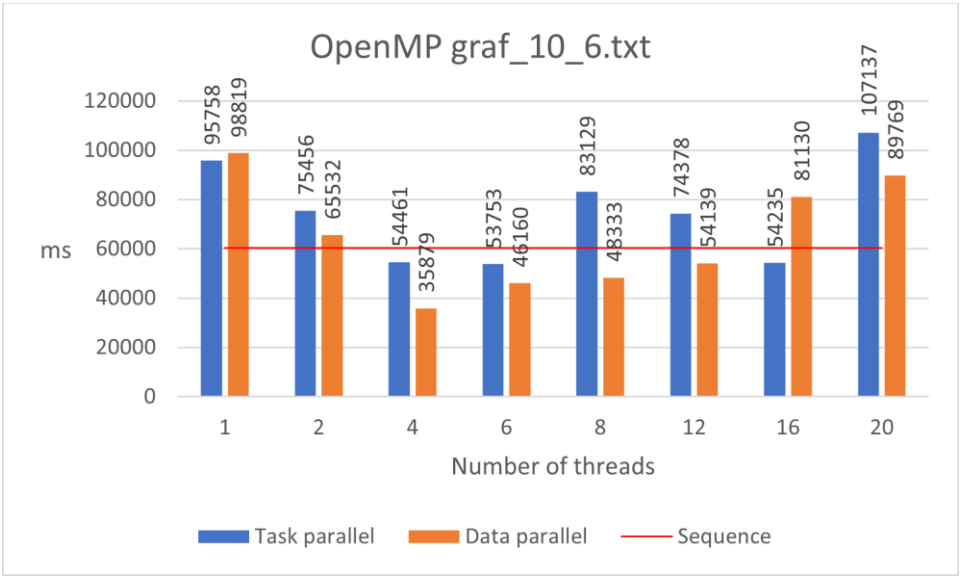
Tento algoritmus vypadá následovně:

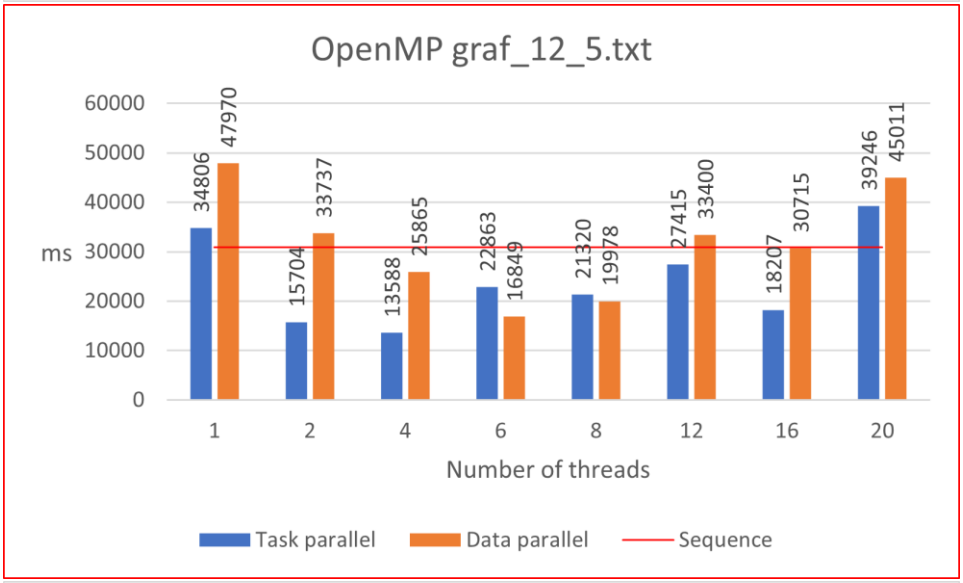
1. Před spuštěním se inicializuje buffer mezistavů. Jeho velikost je určena na základě hloubky zadané uživatelem.
2. Poté se zavolá rekurzivní funkce pro konstrukci podgrafů. Rozdíl mezi touto funkcí a předchozími verzemi spočívá v tom, že po dosažení určité hloubky funkce uloží aktuální mezistav do předem definovaného bufferu a vrátí se o krok zpět.
3. Po ukončení sekvenční části algoritmu se spustí paralelní for smyčka, v níž každé vlákno vezme jeden mezistav a znovu spustí rekurzivní funkci s tímto stavem. Po skončení funkce vlákna pokračují v přebírání dalších stavů, dokud neprojdou celým bufferem.

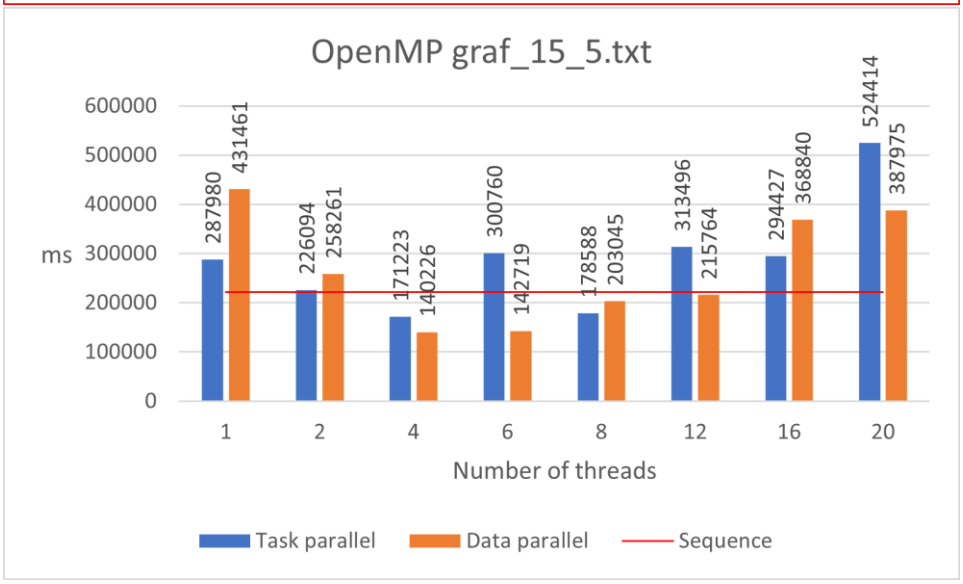
Měření

Obě verze OMP algoritmu byly testovány se stejnými instancemi jako sekvenční řešení. Každý test byl proveden několikrát s různým počtem vláken: 1, 2, 4, 6, 8, 12, 16 a 20. Pro datový paralelismus byla

hloubka zvolena tak, aby se konečný počet mezistavů rovnal 1024 pro každou instanci. Výsledky jsou následující:







Jak je vidět z výše uvedených grafů, oba algoritmy dosahují nejlepších výsledků při použití 4-6 vláken. Pokud je počet vláken příliš vysoký nebo příliš nízký, mohou algoritmy vykazovat horší nebo podobné výsledky jako sekvenční algoritmus.

Při celkovém pohledu na výsledky však algoritmus založený na datovém paralelismu dosahuje ve většině testů lepších nebo srovnatelných výsledků v porovnání s taskovým paralelismem.

Je však třeba poznamenat, že obě verze algoritmů mají špatnou škálovatelnost s ohledem na počet vláken a také nezkracují výrazně čas potřebný k nalezení řešení ve srovnání se sekvenčním algoritmem. Při použití 4-6 vláken pro malé a střední instance se doba provádění algoritmu snížila přibližně 2krát a 1,5krát pro velké instance. To může být způsobeno velkým počtem alokací paměti a neefektivním přístupem vláken ke globální paměti.

Popis paralelního algoritmu a jeho implementace v MPI

V této verzi programu byl algoritmus rozšířen o práci s více procesy podle principu Master-Slave. Za základ byla vzata verze OMP algoritmu, která využívá datový paralelismus.

Master algoritmus

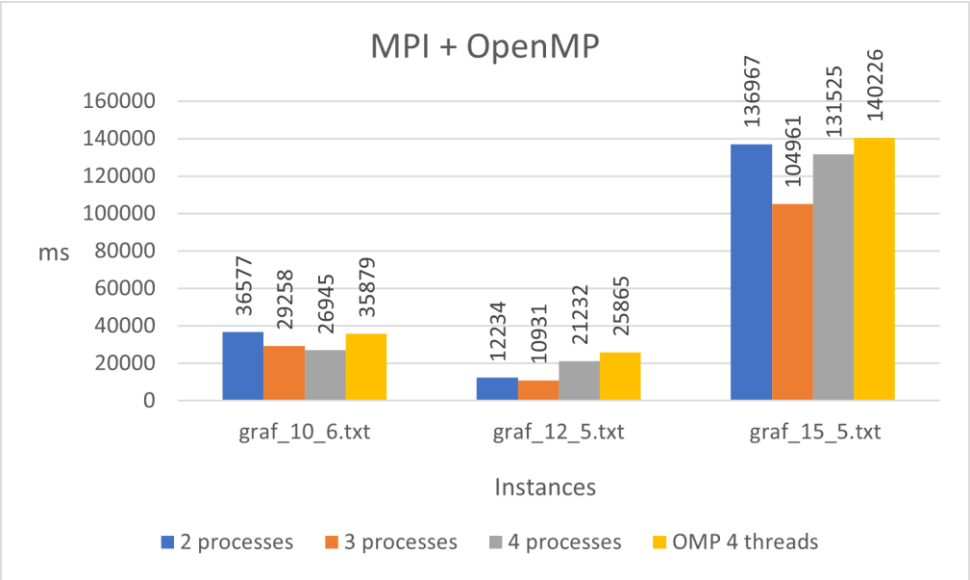
1. První polovina algoritmu (vyplnění bufferu mezistavů) probíhá stejně jako u OMP verze.
2. Po dokončení sběru mezistavů master alokuje buffer, přes který část z nich pošle ostatním procesům. Velikost bufferu závisí na počtu a velikosti mezistavů a počtu procesů včetně hlavního.
3. Po inicializaci bufferu se spustí cyklus, v němž se všem procesům včetně hlavního přidělí všechny dostupné mezistavy. Jakmile jsou všechny spolu s jejich počtem zapsány do bufferu, odešlou se do konkrétního procesu. Jakmile jsou všechny spolu s jejich počtem zapsány do bufferu, odešlou se do konkrétního procesu.
4. Poté, co master zpracuje všechny zbývající mezistavy, počká na výsledky zbývajících procesů. Jakmile budou k dispozici všechny výsledky, vybere nejlepší z nich a dokončí svou práci.

Slave algoritmus

1. Nejdříve proces alokuje buffer o stejné velikosti jako master a bude čekat na data od něj.
2. Jakmile obdrží data od masteru, proces je rozdělí na mezistavy a pokračujte v provádění algoritmu datového paralelismu.
3. Po dokončení algoritmu odešle řešení hlavnímu procesu a dokončí svou práci.

Měření

Měření byla provedena třikrát na všech třech instancích s použitím 2, 3 a 4 procesů a počtu vláken, při kterém bylo dosaženo nejlepších výsledků v předchozích testech. Všechny výsledky jsou zobrazeny v následujícím grafu:



Na grafu je vidět, že nejlepšího výsledku pro všechny instance bylo dosaženo při použití 3 procesů. Při použití většího nebo menšího počtu procesů se výsledek přibližně rovná nejlepšímu výsledku OMP algoritmu se 4 vlákny. Z hlediska zvýšení rychlosti oproti OMP řešení algoritmus nejlépe fungoval na menších instancích, kde se rychlost zvýšila 2,5krát. S růstem velikosti instance se tento rozdíl snižuje na 1,3-1,4násobek.

Závěr

Konečná rychlost provádění algoritmu na malých instancích se při použití MPI řešení zvyšuje přibližně 3krát ve srovnání se sekvenční verzí algoritmu. Při zvětšení velikosti instance se tento rozdíl oproti sekvenčnímu algoritmu sníží až na dvojnásobné zrychlení. I přes celkové snížení doby běhu však algoritmus nedokázal dosáhnout dostatečně velkého zvýšení rychlosti a také se ukázala jeho špatná škálovatelnost vůči počtu procesů a vláken a velikosti instancí.

Algoritmus není efektivní a existuje potenciál pro zlepšení. Tak například proces sestavování pole mezistavů by mohl být paralelizován pomocí taskového paralelismu. Stejným způsobem, ale pomocí paralelního for-cyklu , by bylo možné zrychlit kopírování mezistavů do bufferu pro přenos mezi procesy. Co se týče samotného procesu výpočtu, mohla by být přidána možnost vzájemné výměny mezivýsledků mezi procesy, což by mohlo zabránit zbytečným výpočtům a výrazně zkrátit dobu běhu každého z procesů zvlášť.