**Závěrečná technická zpráva**

Karel Vrabec

**4. 5. 2022**

**ZADÁNÍ SEMESTRÁLNÍ ÚLOHY**

Zkoumaným problémem byl **minimální hranový řez souvislého grafu (MHR)**.

Vstup:

* (počet uzlů grafu)
* (váhy jednotlivých hran grafu)
* (velikost výsledné množiny uzlů)

Úkol:

* nalézt rozklad množiny uzlů grafu do dvou disjunktních podmnožin a tak, aby a nebo opačně
* součet ohodnocení všech hran takových, že a (tj. váha hranového řezu mezi a ) je **minimální**

Výstup:

* výpis **všech hranových řezů** s minimální váhou (příslušnost všech uzlů k dané množině, počet hran řezu a váha tohoto řezu)

**POPIS SEKVENČNÍHO ALGORITMU A JEHO IMPLEMENTACE**

Sekvenční algoritmus je typu **BB-DFS**. Vždy existuje **alespoň jedno řešení** (vždy existuje hranový řez mezi množinami a ). **Cena**, kterou **minimalizujeme**, je součet vah hran řezu (tzv. **váha hranového řezu**) mezi množinami a .

Struktura:

* **Zpracování vstupu:**

Vstupem algoritmu je parametr a cesta k souboru, ve kterém se nachází parametr včetně .

Příklad spuštění: ./seq graf\_15\_14.txt 5

Pro paralelní algoritmy je spuštění obdobné.

* **Vlastní algoritmus:**

Pomocí rekurze postupně procházím všechny uzly grafu, kterým přiřazuji příznak či ( → uzel patří do , → uzel patří do ). Při každém zavolání dané rekurzivní funkce inkrementuji počítadlo volání.

Daná větev **skončí úspěšně**, pokud byly označeny **všechny vrcholy** a velikost jedné z množin a je . Pokud je váha nového řešení **menší než** váha stávajícího řešení, pak je stávající řešení nahrazeno tímto novým řešením. Pokud je váha nového řešení **ekvivalentní** se stávající, pak je nové řešení pouze **uloženo** jako další z možných řešení.

Pro zrychlení výpočtu používám následující **tři podmínky**. Pokud velikosti podmnožin a přesáhnou , větev je ukončena. Pokud je váha aktuálního řezu **větší než** dosud nalezené minimum, větev je ukončena. Pokud je součet **váhy aktuálního řezu** a **dolního odhadu váhy** zbývajícího řezu taktéž **větší než** dosud nalezené minimum, větev je ukončena. Díky tomu je sekvenční algoritmus **rychlejší než** výpočet hrubou silou.

**Váhu aktuálního řezu** získám tak, že postupně procházím uzly **přiřazené** množině . Pro každý takový uzel sečtu váhy hran, které vedou do uzlů přiřazených množině a uložím (včetně počtu těchto hran). Součet těchto uložených hodnot je **kýžená váha a počet hran řezu**.

**Dolní odhad váhy zbývajícího řezu** získám tak, že postupně procházím **nepřiřazené** uzly grafu a pro každý z nich spočítám **navýšení váhy řezu** (vůči již přiřazeným uzlům), pokud by daný uzel patřil do , nebo do . Z těchto dvou hodnot následně uložím **menší**. Součtem těchto minim je **kýžený odhad**.

Dalším trikem, který používám pro zrychlení výpočtu, je **označení prvního vrcholu jako** . Tímto se zbavuji zbytečných symetrických řešení, a tím pádem i nežádoucího výpočtu.

* **Zpracování výstupu:**

Výstupem je **počet řešení** a pro každé z nich také: **přiřazení jednotlivých vrcholů** k množinám a , **počet hran řezu** a **váha tohoto řezu**. Dalším výstupem je také **celkový počet rekurzivních volání**.

Počáteční měření:

|  |  |  |  |
| --- | --- | --- | --- |
| **Soubor** | **Parametr** | **Reálný čas** | **Počet volání** |
| graf\_15\_14.txt | 5 | 0.005 s | 9.6 K |
| graf\_20\_17.txt | 10 | 0.25 s | 423.9 K |
| graf\_30\_20.txt | 15 | 110.912 s | 89.3 M |

**POPIS PARALELNÍHO OPENMP ALGORITMU A JEHO IMPLEMENTACE**

Paralelní OpenMP algoritmus je vylepšenou verzí původního sekvenčního algoritmu, která je nově doplněna o **OpenMP**. Paralelizace se rozlišuje na **funkční a datovou**. Algoritmus je ve velké míře podobný sekvenčnímu, odlišuje ho však použití **více vláken** (nejen jednoho). Nově přibyla **kritická sekce**, ve které se modifikuje nejlepší řešení a která vylučuje přístupy jednotlivých vláken. Dále přibyla deklarace inkrementace počtu rekurzivních volání jako **atomické operace**.

Funkční (taskový) paralelismus:

* Když procházím jednotlivé uzly a přiřazuji jim příznaky či , výpočet se rekurzivně rozděluje **do dvou větví výpočtu** (do první, kde je právě zpracovávaný uzel označen jako , a do druhé, kde je tento uzel označen jako ). Vzniká tak **strom rekurzivních volání**. Pomocí OpenMP se taková rekurzivní volání převádí na tzv. **tasky**, které jsou následně vloženy do **task poolu**, odkud jsou přiřazena ke zpracování jednotlivým vláknům.

Zároveň nastavuji hranici, po které se již volání **nepřevádí** na tasky a výpočet probíhá **sekvenčně**. Je jím vrchol před předposledním vrcholem (tj. třetí hladina od dna stromu). Hranici je možné libovolně měnit a ovlivňovat rychlost výpočtu, ale po podrobném zkoumání se tato možnost **neukázala jako efektivní**.

V základní implementaci používám celkem **4 vlákna**. Na začátku výpočtu deklaruji, že první rekurzivní volání musí být provedeno **pouze jedním vláknem** (jinak by se výpočet spustil 4x, dle počtu vláken).

Počáteční měření (pro 4 vlákna):

|  |  |  |  |
| --- | --- | --- | --- |
| **Soubor** | **Parametr** | **Reálný čas** | **Počet volání** |
| graf\_15\_14.txt | 5 | 0.007 s | 9.7 K |
| graf\_20\_17.txt | 10 | 0.145 s | 413.9 K |
| graf\_30\_20.txt | 15 | 32.693 s | 87.5 M |

Datový paralelismus:

* Nejdříve je z **počátečního stavu** pomocí **algoritmu BFS** vygenerován určitý počet následníků, kteří jsou přibližně na stejné hladině stromu. Tento počet stavů lze **libovolně měnit a ovlivňovat tak rychlost výpočtu**.

Získané stavy jsou **z fronty** převedeny **do vektoru**, nad kterým pracuje **paralelní cyklus**. Ten se řídí určitou **strategií**. Po podrobném zkoumání jsem dospěl k závěru, že **dynamická strategie** (tj. vlákna si iterace rozebírají nahodile podle toho, zdali jsou právě vytížená či ne) je pro mou implementaci **nejefektivnější** (z hlediska reálného času). Každá iterace cyklu obsahuje **rekurzivní volání sekvenčního algoritmu**, který následně vyřeší podstrom náležící danému stavu.

Počáteční měření (pro 4 vlákna):

|  |  |  |  |
| --- | --- | --- | --- |
| **Soubor** | **Parametr** | **Reálný čas** | **Počet volání** |
| graf\_15\_14.txt | 5 | 0.007 s | 9.6 K |
| graf\_20\_17.txt | 10 | 0.113 s | 415.7 K |
| graf\_30\_20.txt | 15 | 19.959 s | 88.9 M |

**POPIS PARALELNÍHO MPI ALGORITMU A JEHO IMPLEMENTACE**

Algoritmus vychází zejména z **datového paralelismu**, ale kromě **více vláken** je potřeba zajistit jeho funkčnost i pro **více procesů**. Výpočet se tedy rozděluje na dvě hlavní větve – **master** (obstarává hlavní proces) a **slave** (obstarává ostatní procesy). Procesy mezi sebou vzájemně komunikují prostřednictvím **zasílání zpráv** (v mém případě taková zpráva představuje **vektor shortů**). Odesílatel zprávu **serializuje**, příjemce ji naopak **deserializuje**.

Master:

* Z **počátečního stavu** je nejdříve pomocí **BFS expanze** vygenerován určitý počet následníků (přibližně na stejné hladině stromu, expanzi je opět možné **limitovat**). Následně dochází k **počáteční distribuci** těchto stavů jednotlivým **slave** procesům tak, aby všechny tytoprocesy měly co na práci.

Jakmile nějaký **slave** proces svou práci **dokončí**, **master** proces mu obratem pošle **další práci** (pokud se ve frontě ještě nějaké rozpracované stavy nachází). Pokud je fronta již prázdná, veškerá práce byla dokončena a všechny **slave** procesy jsou postupně ukončeny. Tím končí i výpočet **master** procesu. **Master** proces se také stará o **příjem vypočítaných řešení** od **slave** procesů a ukládání **nejlepšího z nich**.

Slave:

* Proces čeká v **nekonečné smyčce** na přidělení práce od **master** procesu. Jakmile se tak stane, výpočet pokračuje již známým **datovým paralelismem** (tzn. vygenerování určitého parametrizovaného počtu stavů, převod stavů z fronty do vektoru, rozebrání iterací nad tímto vektorem jednotlivými vlákny dle dynamické strategie, dopočítání daných podstromů sekvenčně). Na konci tohoto výpočtu pošle **slave** proces **master** procesu **výsledek**, ke kterému dospěl. **Master** proces může také **slave** procesu přikázat, aby se **ukončil** a opustil smyčku.

Počáteční měření (pro 4 procesy a 4 vlákna):

|  |  |  |  |
| --- | --- | --- | --- |
| **Soubor** | **Parametr** | **Reálný čas** | **Počet volání** |
| graf\_15\_14.txt | 5 | 0.033 s | 10.5 K |
| graf\_20\_17.txt | 10 | 0.072 s | 557.8 K |
| graf\_30\_20.txt | 15 | 12.628 s | 135.4 M |

**NAMĚŘENÉ VÝSLEDKY A VYHODNOCENÍ**

Všechna měření jsem provedl na clusteru **star.fit.cvut.cz** s pomocí předpřipravených skriptů, vstupních dat a plánovače dostupného přes příkaz **qrun**.

Pro měření jsem vybral **právě 3** různé instance problému (jednu z dostupných testovacích dat, zbylé dvě samostatně vygenerované) se sekvenční časovou složitostí **mezi 4 a 10 minutami** naměřenou na **jednom jádru** uzlu klastru.

Pro OpenMP (task + data) instance jsem změřil paralelní čas na **2, 4, 8, 16 a 20 výpočetních jádrech** uzlu klastru. Obdobně jsem měřil také MPI instance, navíc pro **právě 4** procesy.

Veškeré výsledky všech měření uvedených v tomto dokumentu jsou dostupné na clusteru **star.fit.cvut.cz** v uživatelské složce **vrabekar**.

Sekvenční algoritmus:

|  |  |  |  |
| --- | --- | --- | --- |
| **Soubor** | **Parametr** | **Reálný čas ()** | **Reálný čas ()** |
| graf\_40\_8.txt | 15 | 469.771 | 7.83 |
| graf\_35\_10.txt | 18 | 419.177 | 6.98 |
| graf\_45\_5.txt | 16 | 336.617 | 5.61 |

Funkční paralelismus:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Reálný čas při vláknech ()** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 113.322 | 48.537 | 26.981 | 20.594 | 20.384 |
| graf\_35\_10.txt | 18 | 83.16 | 35.456 | 25.109 | 15.427 | 20.824 |
| graf\_45\_5.txt | 16 | 41.23 | 34.213 | 9.456 | 7.394 | 4.626 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Zrychlení při vláknech** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 4.145 | 9.679 | 17.411 | 22.811 | 23.046 |
| graf\_35\_10.txt | 18 | 5.041 | 11.822 | 16.694 | 27.172 | 20.13 |
| graf\_45\_5.txt | 16 | 8.164 | 9.839 | 35.598 | 45.526 | 72.766 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Efektivnost při vláknech** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 2.073 | 2.42 | 2.176 | 1.426 | 1.152 |
| graf\_35\_10.txt | 18 | 2.52 | 2.956 | 2.087 | 1.698 | 1.006 |
| graf\_45\_5.txt | 16 | 4.082 | 2.46 | 4.45 | 2.845 | 3.638 |

Datový paralelismus:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Reálný čas při vláknech ()** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 169.248 | 81.64 | 53.83 | 51.615 | 52.027 |
| graf\_35\_10.txt | 18 | 91.39 | 46.001 | 27.065 | 24.183 | 25.404 |
| graf\_45\_5.txt | 16 | 70.951 | 32.233 | 18.005 | 14.778 | 13.801 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Zrychlení při vláknech** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 2.776 | 5.754 | 8.727 | 9.101 | 9.029 |
| graf\_35\_10.txt | 18 | 4.587 | 9.112 | 15.488 | 17.334 | 16.5 |
| graf\_45\_5.txt | 16 | 4.744 | 10.443 | 18.696 | 22.778 | 24.391 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Efektivnost při vláknech** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 1.388 | 1.439 | 1.091 | 0.569 | 0.451 |
| graf\_35\_10.txt | 18 | 2.293 | 2.278 | 1.936 | 1.083 | 0.825 |
| graf\_45\_5.txt | 16 | 2.372 | 2.611 | 2.337 | 1.424 | 1.22 |

MPI (4 procesy):

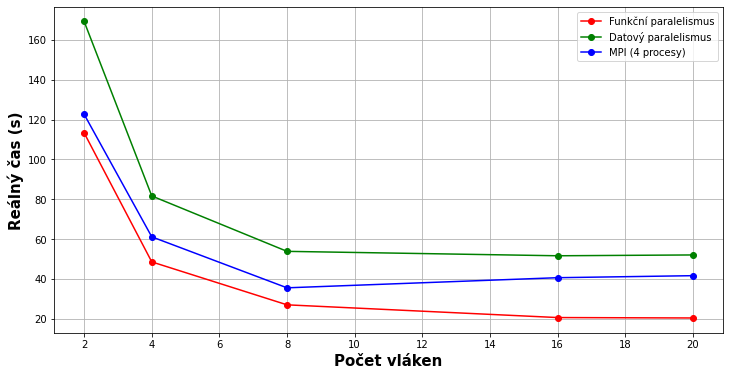
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Reálný čas při vláknech na slave proces ()** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 122.651 | 61.119 | 35.52 | 40.605 | 41.618 |
| graf\_35\_10.txt | 18 | 102.979 | 53.837 | 32.64 | 31.665 | 31.165 |
| graf\_45\_5.txt | 16 | 94.702 | 48.054 | 31.436 | 30.293 | 30.899 |

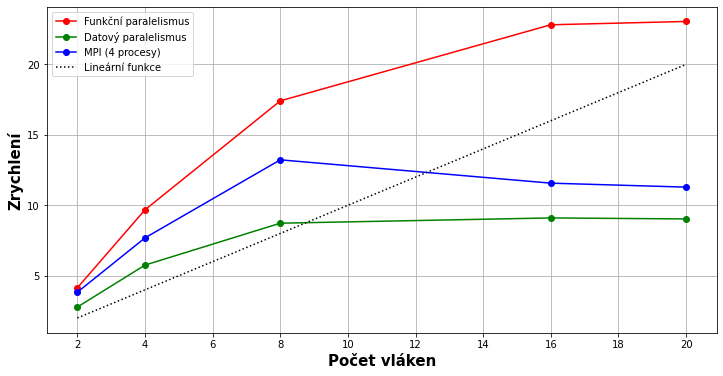
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Zrychlení při vláknech na slave proces** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 3.83 | 7.686 | 13.226 | 11.569 | 11.288 |
| graf\_35\_10.txt | 18 | 4.071 | 7.786 | 12.842 | 13.238 | 13.45 |
| graf\_45\_5.txt | 16 | 3.554 | 7.005 | 10.708 | 11.112 | 10.894 |

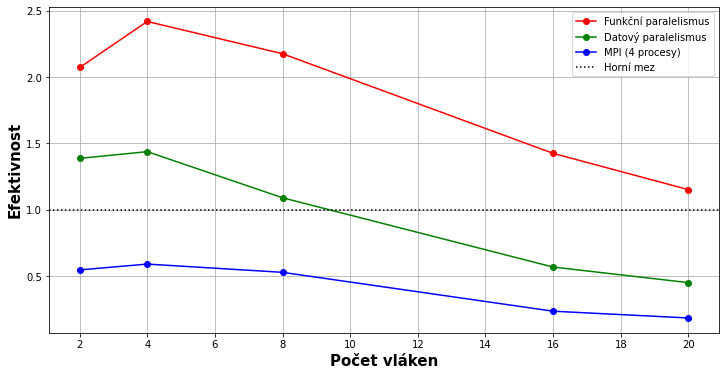
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vstup** | | **Efektivnost při vláknech na slave proces** | | | | |
| **Soubor** | **Parametr** | **2** | **4** | **8** | **16** | **20** |
| graf\_40\_8.txt | 15 | 0.547 | 0.591 | 0.529 | 0.236 | 0.185 |
| graf\_35\_10.txt | 18 | 0.582 | 0.599 | 0.514 | 0.27 | 0.22 |
| graf\_45\_5.txt | 16 | 0.508 | 0.539 | 0.428 | 0.227 | 0.179 |

Předešlé výsledky jsem pro lepší vizualizaci zobrazil v následujících grafech.

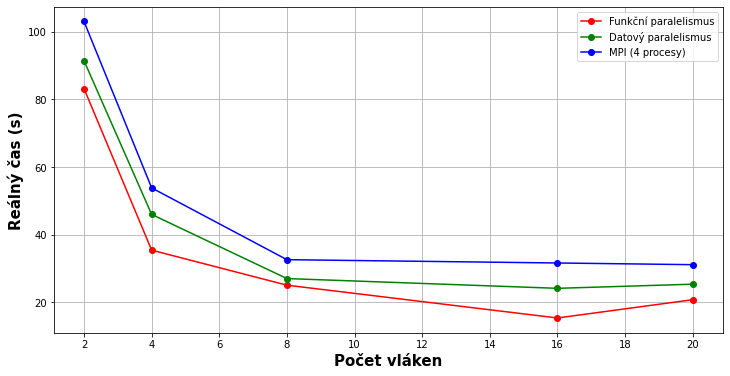
graf\_40\_8.txt ():

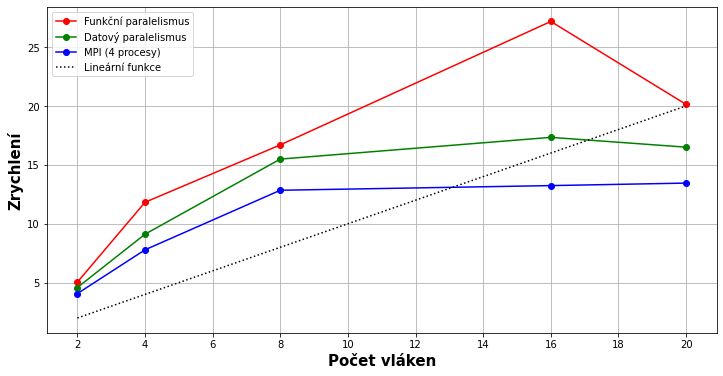


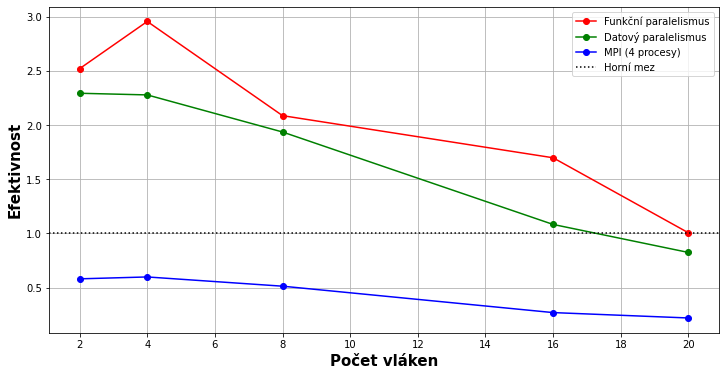




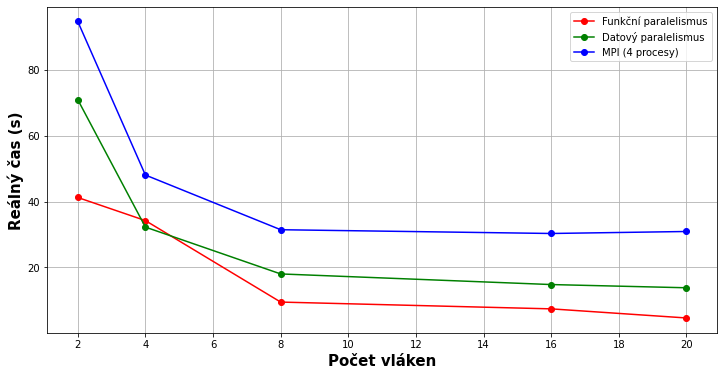
graf\_35\_10.txt ():

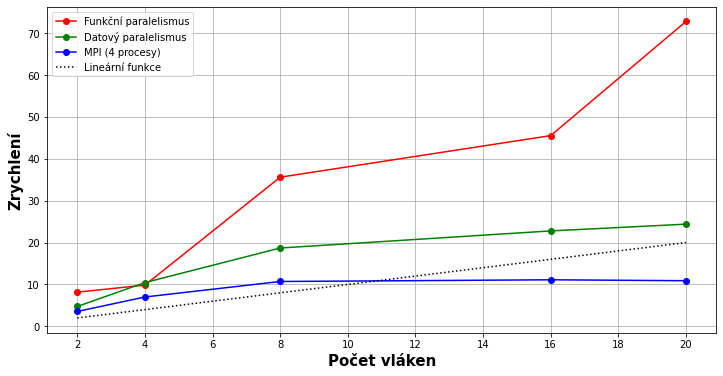


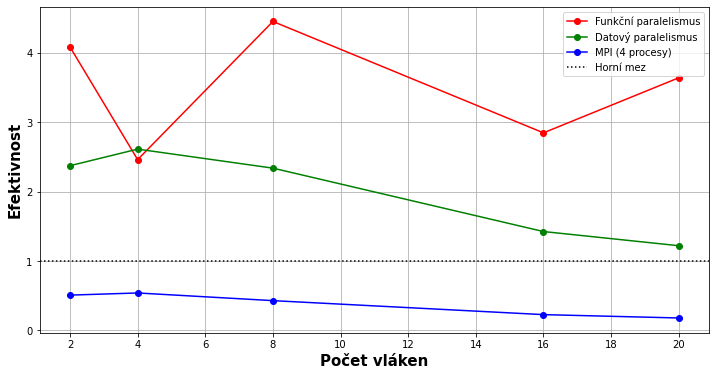




graf\_45\_5.txt ():







**Reálný čas** u všech tří paralelních algoritmů (task + data + MPI) odpovídal mému očekávání. Pro **rostoucí** počet vláken pomalu **klesal**. V některých případech, kdy bylo použito mnoho vláken (např. 16, 20) dokonce **mírně vzrostl**, což mohlo být způsobeno přibývající režií.

**Zrychlení** jsem spočítal jako **podíl** naměřeného **sekvenčního času** a naměřených **paralelních časů**. V ideálním případě by mělo být rovno **počtu používaných vláken** (tzv. **lineární zrychlení**). Avšak pro menší počty vláken bylo zrychlení dokonce **lepší než očekávané** (tzv. **superlineární zrychlení**). Pro větší počty vláken se však zrychlení pomalu **zhoršovalo**.

**Efektivnost** jsem spočítal jako **podíl** vypočítaného **zrychlení** a **počtu používaných vláken**. Výsledky by měly být , ale ve většině případů tuto mez **překračovaly**, čímž nesplnily mé očekávání. Příčinou může být např. lepší zpracování stavů paralelním algoritmem (tj. lepší prořezání stavového prostoru), poněvadž je zpracovával v jiném pořadí (než u sekvenčního algoritmu).

**Škálovatelnost** je schopnost paralelního algoritmu dodržovat **paralelní optimalitu**  
(tj. platí **cenová optimalita** ⬄ **lineární zrychlení** ⬄ **konstantní efektivnost**), pokud klesá/roste **počet používaných vláken** a **velikost problému**. V ohledu na naměřené výsledky (především zrychlení, efektivnost) a grafy lze usoudit, že paralelní algoritmy **neškálují příliš dobře**.

Při implementaci sekvenčního algoritmu jsem **nedosáhl referenčních hodnot** reálného času (uvedených na Courses). Algoritmus tedy lze i nadále **vylepšovat**, a tím měnit jeho vlastnosti (zejména **rychlost**). Je možné např. **vylepšit a zrychlit** funkce na **počítání váhy a dolního odhadu váhy** (tj. používat je více chytřeji a méně častěji, nebo váhu a dolní odhad počítat průběžně). Jedná se o **nejčastěji volané funkce algoritmu**, které způsobují podstatné **zvýšení časové náročnosti**. Případně lze také vylepšit meziprocesovou komunikaci v MPI apod. Mimo jiné lze **měnit limity BFS expanzí**, a tím ovlivňovat rychlost paralelních algoritmů (data + MPI).

**ZÁVĚR**

Výsledkem semestrální práce jsou čtyři implementace algoritmu pro řešení problému **minimálního hranového řezu souvislého grafu (MHR)**. Implementace na sebe navazují a každá z nich vylepšuje tu předchozí. První z nich představuje algoritmus typu **BB-DFS**. Druhá a třetí využívá funkcionalit knihovny **OpenMP** pro **vícevláknové** zpracování. Poslední navíc využívá funkcionalit knihovny **MPI** pro **víceprocesové** zpracování. Závěrečné testování těchto implementací proběhlo na školním clusteru **star.fit.cvut.cz**.