ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

OPTIMALIZÁCIA DEKOMPOZÍCIE PARALELNÝCH A DISTRIBUOVANÝCH VÝPOČTOV VYBRANÝCH GRAFOVÝCH ÚLOH VO VYSOKOVÝKONNOM POČÍTANÍ

Dizertačná práca

Evidenčné číslo práce: 28360020203003

Študijný program:	Aplikovaná informatika
Študijný odbor:	informatika
Pracovisko:	Katedra Informatiky
	Fakulta riadenia a informatiky, Žilinská univerzita v Žiline
Školiteľ:	doc. Ing. Jarmila Škrinárová, PhD.

Žilina, 2020 **Mgr. Adam Dudáš**

ABSTRAKT

DUDÁŠ, Adam. Optimalizácia dekompozície paralelných a distribuovaných výpočtov vybraných grafových úloh vo vysokovýkonnom počítaní [dizertačná práca]. Žilinská univerzita v Žiline. Fakulta riadenia a informatiky. Katedra informatiky. Školiteľ: doc. Ing. Jarmila Škrinárová PhD. Žilina, 2020. 100 strán.

V tejto dizertačnej práci sa zameriavame na prezentovanie analyzovaných poznatkov, algoritmov, metodík a nástrojov relevantných pri riešení problematiky optimalizácie dekompozície paralelných a distribuovaných výpočtov vybraných grafových úloh vo vysokovýkonnom počítaní. Ako prostriedok riešenia uvedenej problematiky sme zvolili problém hranového trojofarbovania grafov, konkrétne sa sústredíme na regulárne hranové trojofarbenie pri skupine kubických grafov nazývaných snarky. Práca obsahuje analýzu teoretických poznatkov týkajúcich sa paralelných a distribuovaných výpočtov, dekompozície výpočtov a vybraných poznatkov z oblasti teórie grafov. Analyzujeme aktuálny stav riešenia z pohľadu paralelných a distribuovaných systémov a z pohľadu algoritmov využívaných pri ofarbovaní grafov. Prezentujeme navrhnuté metodiky, nástroje a algoritmy, ktoré boli experimentálne overené. Všetky experimenty vyhodnocujeme pomocou vybraných kritérií.

Kľúčové slová: paralelné počítanie, distribuované počítanie, dekompozícia výpočtu, vysokovýkonné počítanie, ofarbovanie grafov, snark

ABSTRACT

DUDÁŠ, Adam. Optimization of decomposition of parallel and distributed computations of chosen graph tasks in high performance computing [dissertation thesis]. University of Žilina. Faculty of Management and Informatics. Department of Computer Science. Supervisor: doc. Ing. Jarmila Škrinárová PhD. Žilina, 2020. 100 pages.

This thesis is focused on presentation of analyzed findings, algorithms, methodologies and tools which are relevant in the solving process of optimization of decomposition of parallel and distributed computations of chosen graph tasks in high performance computing. As a mean of solution of this topic we chose problem of edge 3-coloring of graphs - we are focused on proper edge 3-coloring of subset of cubic graphs called snarks. This thesis consists of the analysis of theory related to parallel and distributed computing, decomposition of computations and selected information from the area of graph theory. We analyze the state of art from the point of view of parallel and distributed systems and from the point of view of the algorithms used for coloring of graphs. We also present designed methodologies, tools and algorithms, which were experimentally verified. All experiments were evaluated with the use of chosen criteria.

Keywords: parallel computing, distributed computing, decomposition of computation, high performance computing, graph coloring, snark

POĎAKOVANIE

Chcel by som poďakovať doc. Ing. Jarmile Škrinárovej PhD. za spoluprácu a cenné rady v mojom štúdiu a vedeckej činnosti v rámci doktorandského štúdia.

Tak isto by som chcel poďakovať doc. Mgr. Jánovi Karabášovi PhD. za pomoc v oblasti teórie grafov a oponentom práce za cenné pripomienky a rady.

OBSAH

ZOZNAM ILUSTRÁCIÍ A TABULIEK

ÚVOD

Teória grafov v sebe zahŕňa mnohé problémy, ktoré je možné riešiť pomocou operácií na grafoch. V našej práci sa zaoberáme **hranovým ofarbovaním grafov**, ktoré je relevantné v niekoľkých oblastiach – rozvrhovanie, prideľovanie rádiových frekvencií, optimalizácia kompilátorov alebo SAT solvery.

Keďže hranové ofarbovanie grafov je **NP-úplný problém**, používanie paralelných a distribuovaných metód je na jeho výpočet žiadané. V našom prípade počítame s potrebou ofarbenia veľkého množstva grafov. Túto úlohu **dekomponujeme** na menšie podúlohy, ktoré riešime paralelne alebo distribuovane.

Naším cieľom je vytvoriť metódy a algoritmy na efektívne paralelné, distribuované alebo kombinované ofarbenie veľkých množín grafov pričom dbáme na optimalizačné kritériá ako **čas ukončenia poslednej úlohy** a tak isto na **dĺžku trvania výpočtu jednotlivých podúloh**.

V prvej časti práce uvádzame pojmy z oblasti **paralelného a distribuovaného počítania**, ktoré sú relevantné v našej práci. Venujeme sa paralelným a distribuovaným výpočtom, modelom paralelizácie výpočtu, druhom paralelizmov, paralelným a distribuovaným architektúram a modelom dekompozície úlohy na menšie podúlohy.

Druhá kapitola práce sa vzťahuje na stručné predstavenie pojmov a konceptov z problematiky teórie grafov, ktoré riešime v tejto práci. V práci riešime **hranové ofarbovanie veľkých množín grafov**, pričom sa zameriavame na konkrétny typ grafov, pri ktorých je samotné hranové ofarbovanie problematické. V tejto práci sa zaoberáme len **kubickými grafmi**.

Kapitola tri ponúka prehľad **algoritmov, využívaných pri hranovom ofarbovaní grafu**. Obsahuje algoritmus hranového backtrackingu (tento algoritmus používame v experimentoch), Kowalikov algoritmus ofarbovania hrán a heuristickú metódu premiestňovania konfliktných vrcholov autorov Fiolu a Vilaltellovej.

V štvrtej kapitole prezentujeme dosiahnuté výsledky. Prezentujeme **vlastné algoritmy a metódy** na hranové ofarbovanie veľkých množín grafov s využitím paralelného a distribuovaného počítania. Všetky algoritmy a metodiky sme experimentálne overili, pričom výsledky experimentov ponúkame v prislúchajúcich

častiach kapitoly. Algoritmy sme prezentovali v rámci domácich a zahraničných vedeckých konferencií a sú publikované v niekoľkých z autorských výstupov.

Štvrtá kapitola obsahuje aj výsledky prezentované ako súčasť **článkov v zborníkoch z vedeckých IEEE konferencií** evidovaných v databázach Web of Science a Scopus:

- Dudáš, A., Škrinárová, J., Voštinár, P., Siláči, J.: Improved process of running tasks in the high performance computing.
- Dudáš, A., Škrinárová, J., Vesel, E.: Optimization design for parallel coloring of a set of graphs in the High-Performance Computing.
- Vesel, E., Škrinárová, J., Dudáš, A.: Comparison of in-house HPC calculation with public cloud computing for parallel algorithm containing recursive functions.
- Škrinárová, J., Dudáš, A.: Optimization of Functional Decomposition of Parallel and Distributed Computations in Graph Coloring with the use of High Performance Computing. [Current contents] *(v recenznom konaní)*
- Dudáš, A., Škrinárová, J.: Edge coloring of set of graphs with the use of data decomposition and clustering. *(v recenznom konaní)*

Tieto publikácie boli čiastočne založené na predchádzajúcom výskume prezentovanom v:

- Škrinárová, J., Dudáš, A., Vesel, E.: Model of education and training strategy for the management of HPC systems.
- Škrinárová, J., Dudáš, A.: A Methodology for the professional training of the management and evaluation of HPC systems.

Kapitola tiež obsahuje výsledky prezentované ako súčasť neevidovaného **abstraktu** a **prezentácie** na zahraničnej konferencii ParNum 2019, ktorej výstupom bolo pozvanie do špecializovaného čísla časopisu Computing and Informatics (časopis je evidovaný v databáze Web of Science, Current contents connect), zameraného na výpočtové systémy, paralelné a distribuované výpočty, teoretické poznatky a softvérové inžinierstvo.

Na záver práce uvádzame **vyhodnotenie dosiahnutých výsledkov** a ponúkame pohľad na ďalšie možné oblasti, ktoré je potrebné v budúcnosti skúmať.

CIELE PRÁCE

Hlavným cieľom práce je návrh, implementácia a overenie algoritmov, metodík a nástrojov, ktoré prispejú k optimalizácii dekompozície paralelných aplikácií pri práci s vybranými ťažkými úlohami z oblasti teórie grafov – v našom prípade ide konkrétne o hranové trojofarbovanie grafov. Konkrétnymi cieľmi práce je potvrdenie alebo zamietnutie nasledujúcich hypotéz:

- H1 - Algoritmus hranového backtrackingu nie je citlivý na počiatočný vrchol ofarbovania grafu. To znamená, že ak pri výpočte použijeme rôznu postupnosť hrán, bude doba výpočtu algoritmu pre zadaný graf rovnaká.
- H2 - Neexistuje také poradie ofarbovania hrán grafu, že po jeho aplikovaní na množinu grafov, bude čas ofarbenia celej množiny grafov redukovaný. Táto hypotéza súvisí s optimalizáciou časovej náročnosti algoritmu hranového backtrackingu.
- H3 - Nie je možné rozdeliť množinu grafov na podmnožiny (zhluky) tak, že pre každú podmnožinu grafov je možné nájsť také poradie ofarbovania hrán, ktoré znižuje čas potrebný na ofarbenie každého grafu v danej podmnožine.
- H4 - Neexistuje také poradie ofarbovania hrán grafu, že čas hranového ofarbovania je pre každý graf z množiny rovnaký (resp. je v stanovenej tolerancii - intervale). Táto hypotéza súvisí so správnou dekompozíciou úlohy podľa 1.3.

Okrem cieľov práce formalizovaných v hypotézach je potrebné preskúmať možnosti paralelizácie výpočtov týkajúcich sa hranového trojofarbovania grafov. Tento cieľ je výsledkom osobnej komunikácie s hlavným riešiteľom v rámci projektu VEGA 1/0487/17 - Algoritmy na grafoch a algebraických štruktúrach a tiež bol zahrnutý v slovenskom manifeste týkajúcom sa smerovania výskumu v oblasti teórie grafov z roku 2015.

Doplnkovým cieľom práce je preskúmať možnosti využitia metód zhlukovania pri práci s grafovými údajmi. Cieľom je zhlukovanie, ktoré prispeje k efektívnej dekompozícii problému.

1 PARALELNÉ A DISTRIBUOVANÉ POČÍTANIE

Paralelné a distribuované počítanie je prítomné v niekoľkých odvetviach informatiky – algoritmy, počítačové architektúry, siete, operačné systémy alebo softvérové inžinierstvo. Paralelné a distribuované počítanie stavia na základných systémových konceptoch a pridáva k nim niekoľko konceptov ako súbežné vykonávanie procesov, vzájomné vylúčenie, zasielanie správ a modely so zdieľanou pamäťou.

V práci sa venujeme výpočtu ofarbenia veľkých množín grafov pomocou paralelných, distribuovaných alebo kombinovaných algoritmov. Zameriavame sa na dekompozíciu daného problému a hľadáme taký model, ktorý uľahčí a zrýchli výpočet.

1.1 PARALELNÉ POČÍTANIE

Technológie založené na paralelnom spracovaní sú prítomné vo väčšine nových procesorov a v mnohých výpočtových zariadeniach od štandardných osobných počítačov až po pracovné stanice a superpočítače.

Hlavnou motivačnou silou za týmto vývinom je fakt, že paralelizmus umožňuje výrazné **výkonnostné a časové zlepšenie výpočtov** s využitím štandardných technológií. Z toho vyplýva výrazné zníženie nákladov na výpočet problému v porovnaní s vývojom špecializovanej vysokovýkonnej infraštruktúry [39].

Idea počítačov s jedným procesorom začína pôsobiť archaicky a zastaralo. V súčasnosti je potrebné upraviť prístup k počítaniu:

- Chceme zvýšiť výkon počítača s využitím jedného procesora tak, aby pre svoj chod potreboval primerané množstvo energie. Na dosiahnutie požadovaného výkonu je praktickejšie **využitie viacerých jednoduchých procesorov** [36]. Výsledkom tohto pozorovania je tvrdenie, že ak aplikácia nie je dostatočne rýchla na stroji s jedným procesorom, potom, v prípade, že nevyužíva paralelné počítanie, nebude rýchla ani na multiprocesorovom stroji.
- Je potrebné vytvoriť nástroje, ktoré dokážu rozpoznať paralelizmus. V algoritme je priestor na zrýchlenie jeho vykonávania v takom prípade, že niektoré **podúlohy** môžu byť **počítané súčasne**. Podmienkou takéhoto súčasného vykonania podúlohy je uchovanie správnosti informácie.

- **Optimalizácia výkonu systému** závisí od správnej paralelizácie programu na všetkých úrovniach: algoritmická, vývojová, úroveň operačného systému, kompilačná a hardvérová.
- Pri paralelnom počítaní je potrebné vziať do úvahy počet procesorov používaných pri výpočte, **komunikačné náklady** medzi jednotlivými procesormi a medzi procesormi a pamäťou. Z tohto pohľadu rozlišujeme dva typy problémov:
    - **Problémy závislé od výpočtu** (z angl. Compute - bound problems) – pri týchto problémoch je potenciálne zrýchlenie výpočtu závislé od rýchlosti výpočtu algoritmu na jednotlivých procesoroch.
    - **Problémy závislé od komunikácie** (z angl. Communication - bound problems) – v tejto množine problémov je potenciálne zrýchlenie výpočtu závislé na rýchlosti dodávania dát procesorom v resp. extrakcie dát z procesorov [35].

Paralelné výpočty môžeme definovať ako výpočty, ktoré simultánne používajú viaceré výpočtové prostriedky. Pre svoju aktivitu využívajú viaceré procesory (angl. Processing element, PE). Ich úlohy sú delené na diskrétne časti, zvané podúlohy, ktoré je možné riešiť súbežne a každá z týchto diskrétnych častí je následne rozdelená na postupnosť inštrukcií, ktoré je možné vykonávať súbežne na rôznych procesoroch [2].

1.1.1 PARALELNÉ ALGORITMY A PROGRAMY

Paralelný algoritmus je taký postup riešenia problému, pri ktorom je možné všetky jeho podúlohy riešiť paralelne v rovnakom čase za predpokladu nezávislosti dát medzi jednotlivými podúlohami [35].

V **paralelnom programovaní** vzrastá počet základných inštrukcií tak isto ako potrebný počet hardvérových inštrukcií – keďže môžeme spravovať súčasne niekoľko stoviek procesorov, medzi ktorými môžu prebiehať milióny medziprocesorových interakcií [21]. To spôsobí, že komplexný program je modulárne - zložený z jednoduchých komponentov. Samotné komponenty majú štruktúru vysokoúrovňových abstrakcií ako dátové štruktúry, cykly alebo procedúry.

Paralelné programy tvoríme pomocou niekoľkých modelov:

- **Vláknový model** – je základná jednotka pre plánovanie činnosti procesora. Pozostáva z čítačky inštrukcií, sady registrov a zásobníka. Tento model vieme

implementovať niekoľkými spôsobmi, napríklad POSIX threads alebo OpenMP [8].

- **Model zasielania správ** – úlohy si vymieňajú údaje zasielaním a prijímaním správ. Skupina úloh používa každá svoju lokálnu pamäť pri výpočte. Implementujeme pomocou Message Passing Interface (MPI) [8, 21].
- **Dátovo paralelné systémy** – paralelné spracovanie sa v tomto modeli zameriava na skupiny úloh pracujúce s dátovou množinou. Každá skupina úloh pracuje na inej časti množiny. Medzi implementácie patria Fortran 90, 95 alebo Vysokovýkonný Fortran (z angl. Highperformance Fortran, HPF) [10, 21].
- **SPMD model** – model jedného programu a niekoľkých dátových množín (z angl. Single program multiple data). Každá úloha vykonáva ten istý program na inej množine dát [21].
- **MPMD model –** model niekoľkých programov a niekoľkých dátových množín (z angl. Multiple program muliple data). Úlohy vykonávajú ten istý program s rôznymi dátami [21].
- **Hybridné modely** – sú tvorené spájaním ostatných paralelných programových modelov. Štandardnými kombináciami sú OpenMP v spojení s MPI, POSIX threads spojené s MPI alebo dátovo paralelný model, implementovaný v HPF v spojení s MPI [21].

Paralelizmus je možné implementovať na **rôznych úrovniach systému** pričom je potrebné využívať hardvérové a softvérové techniky:

- **Paralelizmus na úrovni dát** simultánne vykonáva operácie na viacerých bitoch jednej inštancie dát alebo na viacerých dátach. Príkladom takejto paralelizácie by mohol byť paralelný súčet, paralelné násobenie alebo delenie binárnych čísel, VPAs (z angl. Vector Processing Arrays) alebo systolické polia s viacerými vzorkami dát.
- **Paralelizmus na úrovni inštrukcií** súčasne vykonáva viac ako jednu inštrukciu na procesore. Príkladom paralelizmu na úrovni inštrukcií je zreťazenie inštrukcií pomocou tzv. pipeliningu.
- Pri **paralelizme na úrovni vlákien** sa vykonáva niekoľko softvérových vlákien súčasne na jednom alebo viacerých procesoroch. Vlákno je časť programu, ktorá zdieľa zdroje procesora s inými vláknami.

- Proces je program, ktorý beží na počítači a má alokované zdroje počítača ako pamäťové miesto a registre. **Paralelizmus na úrovni procesov** je založený na multitaskingu a časovom zdieľaní, pričom je na jednom alebo viacerých procesoroch vykonávaných niekoľko programov súčasne [35].

1.1.2 PARALELNÉ POČÍTAČOVÉ SYSTÉMY

Pre účely paralelných a distribuovaných výpočtov bolo vyvinuté veľké množstvo rôznych typov **paralelných počítačových systémov**. Najznámejším rozdelením týchto systémov je **Flynnova taxonómia**, ktorá berie do úvahy dva toky – inštrukčný a dátový. Podľa toho, či jednotlivé toky obsahujú jeden (single) alebo viac (multiple) prvkov poznáme vo Flynnovej klasifikácii štyri typy paralelných architektúr [10, 35]:

- **Jeden inštrukčný a jeden dátový tok** (angl. Single Instruction Single Data Stream, SISD) – príkladom takéhoto systému by mohol byť jeden procesor.
- **Jeden inštrukčný a niekoľko dátových tokov** (angl. Single Instruction Multiple Data Stream, SIMD) – všetky procesory vykonávajú rovnaké inštrukcie na rovnakých dátach. Každý procesor má vlastné dáta v lokálnej pamäti a procesory si navzájom dáta vymieňajú pomocou jednoduchých komunikačných schém.
- **Niekoľko inštrukčných a jeden dátový tok** (angl. Multiple Instruction Single Data Stream, MISD) – niekoľko procesorov vykonáva rôzne inštrukcie na rovnakých dátach.
- **Niekoľko inštrukčných a niekoľko dátových tokov** (angl. Multiple Instruction Multiple Data Stream, MIMD) – každý procesor vykonáva vlastné inštrukcie na svojich lokálnych dátach. Príkladom takýchto paralelných procesorov sú viacprocesorové systémy a viacvláknové multiprocesory.

Flynnova klasifikácia je pomerne hrubá a samotná **synchronizácia** medzi procesormi Flynnom nebola uvažovaná ako jedno z kritérií systému. Pri pohľade na najčastejšie využívané paralelné architektúry, by sme mohli vytvoriť nasledovnú, jemnejšiu klasifikáciu [35]:

- **Multiprocesory so zdieľanou pamäťou**, tiež nazývané UMA (z angl. Uniform Memory Accesss), umožňujú jednoduchý vývoj paralelného softvéru a podporujú zdieľanie kódu a dát [37]. Zdieľaná pamäť je používaná ako prostriedok na komunikáciu medzi procesormi. Všetky procesory

v architektúre so zdieľanou pamäťou vedia pristupovať k rovnakému adresnému priestoru pomocou prepojovacej siete. Tento typ architektúry umožňuje ľubovoľnému procesoru pristupovať k ľubovoľnému pamäťovému modulu. V systémoch s viacerými pamäťovými modulmi je možný prístup viacerých procesorov do viacerých pamäťových modulov súčasne. Jedná sa o UMA systém - každý procesor vie pristupovať k ľubovoľnému pamäťovému modulu v rovnakom čase. Pri multiprocesoroch so zdieľanou pamäťou je potrebné riešiť synchronizačné problémy aby sa predišlo konfliktom v zapisovaní alebo čítaní dát jedným alebo viacerými procesormi.

- **Multiprocesory s distribuovanou pamäťou**, tiež NUMA (z angl. Nonuniform Memory Accesss), sa vyznačujú tým, že každý pamäťový modul je asociovaný s procesorom. Každý procesor môže priamo pristupovať k svojej pamäti ale na prístup k neasociovaným pamäťovým modulom musí byť v systéme zavedený mechanizmus zasielania správ (z angl. Message passing mechanism). Na rozdiel od multiprocesorov so zdieľanou pamäťou, je pri NUMA systémoch prístup k pamäťovým modulom nejednotný. Podľa toho, či je systém s distribuovanou pamäťou zložený z identických alebo heterogénnych procesorov hovoríme o symetrickom resp. nesymetrickom multiprocesore - SMP (z angl. Symmetric multiprocessor) resp. ASMP (z angl. Asymmetric multiprocessor).
- **SIMD procesory** môžu patriť do triedy multiprocesorov so zdieľanou pamäťou ale aj multiprocesorov s distribuovanou pamäťou. SIMD systémy, ktoré sú založené na zdieľanej pamäti sú vhodné pre aplikácie, ktoré potrebujú časté interakcie s dátami, kde jeden procesor plní funkciu producenta nových dát a ostatné procesory funkciu konzumenta dát. Každý procesor vykonáva rovnakú úlohu v synchronizácii s ostatnými procesormi. Vykonávaná úloha môže byť jednoduchá inštrukcia, vlákno alebo proces.
- **Systolické procesory** obsahujú procesory, ktoré väčšinou vykonávajú rovnakú úlohu. Štandardné prepojenie jednotlivých procesorov je zložené z prepojení medzi susednými procesormi a niekoľkými globálnymi prepojeniami. Každý procesor má malé pamäťové miesto na ukladanie dát a medzivýsledkov výpočtu. Systolické architektúry sú vhodné na implementáciu algoritmov, ktoré sú pravidelné s jednoduchými dátovými závislosťami. Príkladom

takýchto algoritmov sú algoritmy lineárnej algebry, hľadanie reťazcov a vzorov, digitálne filtre a podobne. Problémom so systolickými procesormi je ich návrh. Systolický procesor je štandardne navrhnutý na implementáciu špecifického algoritmu - pre implementáciu iného algoritmu, musí byť procesor navrhnutý znovu. Druhým problémom je problém s rozdelením I/O operácií vyplývajúci zo samotnej architektúry procesora.

- **Klastrové systémy** sú zložené z dvoch alebo viacerých počítačov, ktoré sú využívané na výpočet určeného problému alebo jeho časti. Vo výpočtovom klastrovom systéme je prepojenie medzi jednotlivými počítačmi zabezpečená pomocou lokálnej siete (LAN). Počítače v klastrovom systéme komunikujú navzájom a so zdieľanou pamäťou. Zdieľaná pamäť musí byť schopná komunikovať s viacerými procesormi zároveň. Klientský počítač distribuuje úlohu medzi procesory klastra a po výpočte úlohy zozbiera výsledky.
- **Gridové systémy** poskytujú prístup k výpočtovým prostriedkom distribuovaným pomocou siete WAN. Gridový systém môže byť zložený z veľkého počtu procesorov distribuovaných na veľkej geografickej ploche. V porovnaní s klastrovým počítaním, gridový systém je veľký klaster počítačov, ktoré sú navzájom prepojené pomocou WAN siete ako Internet.
- **Viacjadrové procesory** sú multiprocesorové systémy, ktoré majú všetky procesory uložené na rovnakom čipe. Pod pojmom viacjadrový procesor môžeme tiež rozumieť systém, kde sú procesory uložené na rôznych čipoch ale tie používajú jedno balenie (napríklad multičipový modul). Takéto uzavreté balenie umožňuje vysokú rýchlosť medzi procesorovej komunikácie bez vysokej energetickej náročnosti.
- **Tokové multiprocesorové systémy** sú SIMD alebo MIMD stroje, ktoré sú zložené z tokových procesorov. Tokový procesor je definovaný ako procesor, ktorý pracuje s dátovými tokmi a jeho architektúra obsahuje jadrá na prácu s takýmito tokmi [38]. Koncept tokového spracovania dát je úzko spätý s grafickými procesormi (GPU).

1.2 DISTRIBUOVANÉ POČÍTANIE

Komponenty distribuovaného systému sú počítače, navzájom prepojené sieťou, ktoré komunikujú a koordinujú svoju prácu zasielaním správ [40]. Sieťou prepojené

počítače môžu byť fyzicky rozdelené akoukoľvek vzdialenosťou – môžu byť na rôznych kontinentoch, v rovnakej budove ale aj v rovnakej miestnosti. Definícia distribuovaných systémov v sebe zahŕňa niekoľko významných poznatkov [40]:

- **Súbežnosť –** Pri sieťovo prepojených počítačoch je štandardom súbežné vykonávanie programov - každý používateľ môže pracovať na svojom počítači súbežne s inými používateľmi pričom zdieľajú zdroje ako internetové stránky alebo súbory. Zvýšením počtu zdrojov v sieti (napr. počítačov) je možné zvýšiť kapacitu systému na prácu so zdieľanými prostriedkami. Kľúčovým konceptom vyplývajúcim zo súbežného vykonávania programov je koordinácia týchto programov.
- **Neexistujú globálne hodiny –** V prípade, že niekoľko počítačov v sieti potrebuje navzájom spolupracovať, na vzájomnú koordináciu úkonov využívajú vymieňanie správ. Táto koordinácia je závislá na spoločnom čase, v ktorom sa jednotlivé akcie programu vykonajú. Problémom sú však obmedzenia v presnosti, s ktorou dokážu počítače v sieti synchronizovať svoje hodiny – v takomto systéme nie je možné nastaviť jeden presný, korektný čas. Tento problém je priamym dôsledkom toho, že v systéme je na komunikáciu využívané len zasielanie správ po sieti.
- **Nezávislé zlyhania** – Dôsledkom zlyhania siete je, že počítače zapojené v tejto sieti sa stanú nedostupné, aj keď samotná funkčnosť počítačov nie je ovplyvnená. Programy v týchto počítačoch nie sú schopné detegovať, či nastalo zlyhanie siete, alebo je sieť len nezvyčajne spomalená. Tak isto zlyhanie počítača, alebo neočakávané ukončenie programu na jednom z počítačov nie je hneď oznámené ostatným komponentom v sieti. Každý komponent systému môže zlyhať nezávisle a ostatné komponenty to nemusí vôbec ovplyvniť.

Z definície distribuovaných systémov a ich konštrukčných vlastností vyplýva niekoľko podmienok/pravidiel, ktoré by mali distribuované systémy dodržiavať [40]:

- Sieťové prepojenie počítačov umožňuje používateľom prístup k službám a aplikáciám heterogénnej množiny počítačov a sietí. **Heterogenita** je možná na niekoľkých komponentoch distribuovaného systému: siete, počítačový hardvér, operačný systém, programovacie jazyky a implementácie jednotlivých používateľov systému. Programy rôznych používateľov

navzájom nie sú schopné komunikovať v prípade, že nepoužívajú rovnaké štandardy. Jedným z takýchto štandardov je využitie tzv. middlevéru, ktorý okrem riešenia heterogenity poskytuje aj jednotný výpočtový model pre programátorov serverov a distribuovaných aplikácií. Middlevér je softvér, ktorý poskytuje programové abstrakcie a zahladzuje heterogenitu jednotlivých systémových častí (sietí, hardvéru, operačných systémov, programovacích jazykov).

- **Otvorenosť** systému spôsobuje to, že je možné systém rozširovať a implementovať rôznymi spôsobmi. Otvorenosť distribuovaného systému je definovaná najmä možnosťou pridávania nových služieb schopných zdieľania zdrojov. Systémy, ktoré sú navrhnuté tak, aby podporovali zdieľanie zdrojov označujeme ako **otvorené distribuované systémy**. Môžu byť rozšírené na hardvérovej úrovni pridaním počítačov do siete a na softvérovej úrovni pridaním nových služieb alebo novej implementácii starých služieb.
- **Bezpečnosť** v prípade informačných zdrojov sa skladá z troch komponentov:
    - diskrétnosť – ochrana pred neautorizovanými používateľmi,
    - integrita – ochrana pred neautorizovaným upravením alebo zničením dát,
    - dostupnosť – ochrana proti prerušeniu z dôvodu prístupu ku zdrojom.

  Väčšina bezpečnostných problémov je v distribuovaných systémoch uspokojivo vyriešených, pretrvávajú však dva problémy, ktoré je v budúcnosti potrebné riešiť:
    - **Odmietnutie servisných útokov** – Je možné, že používateľ bude chcieť prerušiť vykonávanie jednej zo služieb. Toto môže byť dosiahnuté zahltením služby takým množstvom dopytov, že ostatní používatelia, ktorí chcú službu využiť, nebudú mať možnosť so systémom pracovať.
    - **Bezpečnosť mobilného kódu** – S mobilným kódom musí byť zachádzané s dávkou opatrnosti. Pri obdržaní programu pomocou elektronickej pošty a jeho následným spustením v rámci distribuovaného systému, nie je možné predpokladať aké následky samotné spustenie bude mať.

- Systém je **škálovateľný** ak ostáva efektívny aj po pridaní značného množstva zdrojov a používateľov. Návrh škálovateľného distribuovaného systému so sebou prináša niekoľko výziev:
    - **Náklady na udržiavanie fyzických zdrojov** – S rastúcim dopytom po určitom zdroji, by malo byť možné systém rozširovať tak, aby boli náklady na toto rozšírenie zhodné so samotným dopytom.
    - **Udržiavanie výkonnosti** – Ak zoberieme do úvahy riadenie množiny dát, ktorých veľkosť je proporčná k počtu používateľov systému, chceme pristupovať k týmto dátam bez straty výkonnosti. Algoritmy, ktoré používajú hierarchické dátové štruktúry sú v tomto ohľade lepšie škálovateľné ako tie, ktoré používajú lineárne štruktúry. Nie sú však bezstratové. V prípade, že máme hierarchicky štruktúrovanú množinu dát o veľkosti *n* prvkov, čas na prístup k tejto množine dát je *O(log n)*. Aby sme systém mohli nazvať škálovateľným, maximálna strata výkonu by nemala byť vyššia, než strata pri využívaní hierarchickej štruktúry.
    - **Zachovanie softvérových zdrojov** – Je náročné odhadnúť dopyt, ktorý by mohol v rámci systému v budúcnosti vzniknúť. Prílišné kompenzovanie pre prípadné potreby budúcich používateľov je v istých prípadoch horšie ako prispôsobenie systému zmene, napríklad väčšie internetové adresy zaberajú viac pamäťového miesta.
    - **Vyhnutie sa úzkym hrdlám v programoch** – Algoritmy by mali byť decentralizované, aby sa vyhlo úzkym hrdlám v programoch. Pri niektorých často navštevovaných zdieľaných zdrojoch (napríklad stránke pristupovanej mnohými používateľmi) môže nastať pokles vo výkonnosti. Na zlepšenie výkonnosti takýchto zdieľaných zdrojov je vhodné použitie vyrovnávacích pamätí a replikácie týchto zdrojov.

  V ideálnom prípade by sa systém a softvér nemali meniť pri zmene škály systému. Takáto stabilita je však ťažko dosiahnuteľná.
- Ak by proces, ktorý riadi zdieľané zdroje mohol v jednom čase prijať len jednu žiadosť, systém by mal zníženú priepustnosť. Práve preto služby a aplikácie povoľujú **súbežné** spracovávanie niekoľkých žiadostí.

- **Transparentnosť** je prezentovanie komponentov distribuovaného systému takou formou, aby bol systém vnímaný ako celok (nie ako kolekcia oddelených častí). Transparentnosť je v distribuovaných systémoch implementovaná podľa ANSA [41] s pridaním tzv. mobilnej transparentnosti:
    - **Prístupová transparentnosť** je založená na používaní rovnakej sady príkazov pre prácu s lokálnymi a vzdialenými zdrojmi.
    - **Lokačná transparentnosť** zabezpečuje, že k zdrojom je možné pristupovať bez potreby vedomosti o tom, či sú vzdialené alebo lokálne.
    - **Súbežná transparentnosť** poskytuje možnosť vykonávania niekoľkých procesov, ktoré využívajú zdieľané zdroje súbežne bez toho aby medzi nimi došlo k vyrušeniam.
    - **Replikačná transparentnosť** je založená na možnosti využívania niekoľkých inštancií zdrojov s cieľom zvýšenia spoľahlivosti a výkonu systému.
    - **Transparentnosť chybovosti** zabezpečuje zapuzdrenie chýb v systéme takým spôsobom, ktorý umožňuje používateľom a aplikáciám ukončiť ich úlohu aj napriek chybe hardvérového alebo softvérového komponentu.
    - **Mobilná transparentnosť** dovoľuje zdrojom a klientom pohybovať sa v systéme bez toho aby boli ovplyvnené operácie alebo používateľské programy.
    - **Výkonnostná transparentnosť** zabezpečuje možnosť prekonfigurovať systém s cieľom zlepšenia výkonu podľa zmien v záťaži na systém.
    - **Transparentnosť škálovateľnosti** je založená na možnosti rozšírenia systému alebo aplikácií bez zmien v štruktúre systému alebo algoritmov.
- Hlavné vlastnosti systému, ktoré vplývajú na **kvalitu služby** sú spoľahlivosť, bezpečnosť a výkon. Novším, podstatným aspektom kvality služby je prispôsobivosť zmenám konfigurácie systému a dostupnosti zdrojov. Spoľahlivosť a bezpečnosť sú kritické pri návrhu väčšiny počítačových systémov. Výkon ako súčasť kvality služby bol definovaný ako prispôsobivosť zmenám konfigurácie systému a priepustnosť systému.

1.3 DEKOMPOZÍCIA ÚLOHY

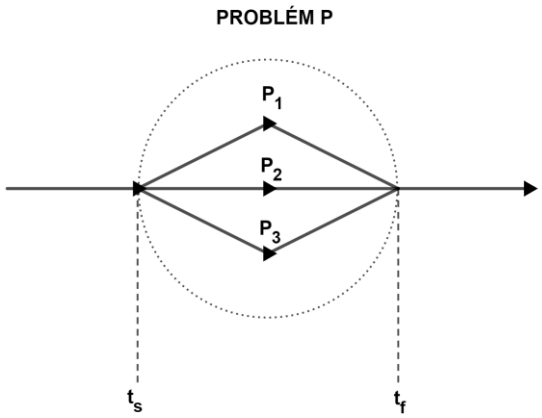
Dekompozíciu problému môžeme opísať ako rozkladanie výpočtov na menšie časti. Pre takéto rozdelenie úlohy je vhodné používať niektoré z **modelov dekompozície**.

Optimalizáciou týchto modelov myslíme ich využívanie a prípadné upravovanie tak, aby sme minimalizovali čas výpočtu.

Na základe dekompozície problému vznikne množina nezávislých alebo vzájomne komunikujúcich procesov (úloh), ktorú možno priradiť vhodnej paralelnej architektúre [2]. Pri prideľovaní úloh na jednotlivé procesory dbáme na vyrovnávanie záťaže s cieľom minimalizovať náklady.

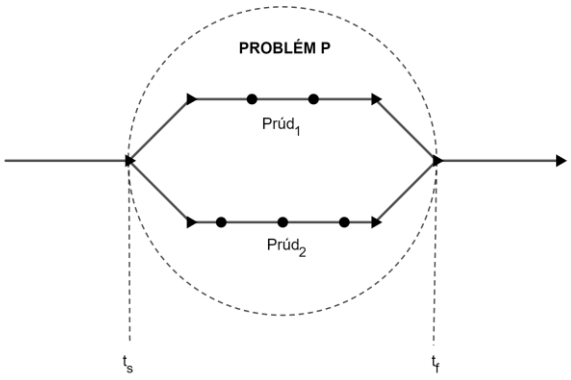
Základom pre dekompozíciu problémov je poznanie **idealizovaných typoch paralelizmov**:

- **Jednoduchý paralelizmus** – v ktorom je problém *P* rozdelený na množinu *n* podproblémov P_1, P_2, ... P_n , ktoré riešime nezávisle, pričom používame nezávislé množiny údajov. Takéto riešenie podproblémov začína v počiatočnom čase t_s a končí vo finálnom čase t_f [2].



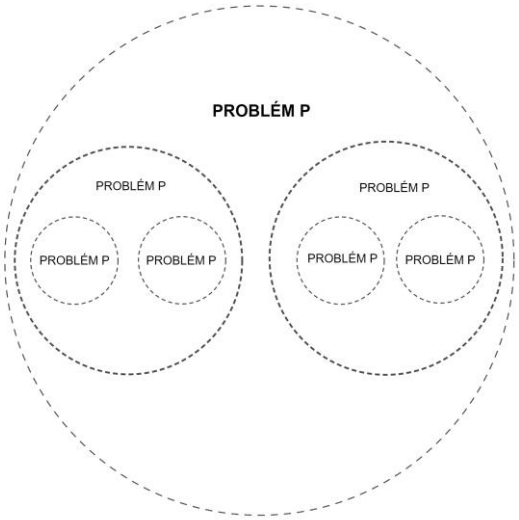
Obrázok 1.1 – Jednoduchý paralelizmus, problém P je zložený z troch podproblémov P_1, P_2, P_3

- **Prúdový paralelizmus** – podobne ako pri jednoduchom paralelizme je potrebné problém *P* rozdeliť na množinu *n* podproblémov P_1, P_2, ... P_n, ktoré však budú riešené v postupnosti – prúde – jeden za druhým a pritom spracujú rozsiahlu množinu údajov [2].



Obrázok 1.2 – Prúdový paralelizmus, podproblémy problému P spracovávané v dvoch prúdoch

- **Expanzívny paralelizmus** – na rozdiel od jednoduchého a prúdového paralelizmu, expanzívny paralelizmus je definovaný rekurzívnym riešením *n* tých istých problémov [2].



Obrázok 1.3 – Expanzívny paralelizmus, rekurzívne riešenie problému P

- **Masívny paralelizmus** – je paralelizmom využívaným pri rozsiahlej množine funkčne identických problémov *P*, ktoré spracovávajú paralelne rozsiahlu množinu údajov v identickom čase [2].

Pre rozdelenie úlohy na časti je nutné používať niektoré z **modelov dekompozície**. Podľa [2] rozoznávame 4 druhy dekompozície problému:

- **Jednoduchá dekompozícia** – Keď označíme časový úsek na vykonanie úlohy spojenej s riešením problému P_i ako $T(P_i)$, potom je paralelný čas výpočtu

$t_f - t_s$ pri $p = n$ procesoroch určený ako maximum časových intervalov $T(P_i)$, čiže [2]:

$$T(n,P) = t_f - t_s = max \{ T(P_i) \mid i = 1...n \}, if\ p = n \quad (1.1)$$

Preto pri jednoduchej dekompozícii požadujeme približne **rovnaké časové intervaly** $T(P_i)$ riešenia jednotlivých podproblémov, vyjadrené nasledovne [2]:

$$T(P_1) \approx T(P_2) \approx ... \approx T(P_n) \quad (1.2)$$

- **Funkcionálna dekompozícia** – Ak predpokladáme, že $T(P_i)$ označuje časový interval na riešenie podproblému P_i nachádzajúceho sa v prúde a spracovávaná údajová množina na vstupe obsahuje m prvkov, efektívne využitie prúdového spracovania pri použití p procesorov podlieha splneniu dvoch podmienok:
  1. V ideálnom prípade je potrebné problém P dekomponovať na $n = p$ podproblémov, pre ktoré platí [2]:

     $$T(P_1) \approx T(P_2) \approx ... \approx T(P_n) \quad (1.3)$$

     V opačnom prípade procesory budú zbytočne čakať na ukončenie práce iných procesorov [2].
  2. Pre zabezpečenie vysokej výkonnosti výpočtu je potrebné, aby počet prvkov m spracovávanej vstupnej množiny bol dostatočne vysoký, aby dostatočne nasýtil prúd paralelne pracujúcich procesorov [2].
- **Hierarchická dekompozícia** – je dekompozíciou expanzívne paralelného problému. Ak považujeme $T(P)$ za časový interval na vykonanie úlohy pre riešenie problému P a tento problém je riešený rekurzívne dvoma tými istými úlohami (do m úrovní) z čoho vyplýva, že jeho celková veľkosť je $n = 2m-1$, celkový paralelný čas je $O(log_n)$. Pri architektúrach SIMD je možné odvodiť redukovaný počet procesorov (menší ako očakávaných $O(n)$) pri ktorom algoritmus zachováva stále svoju optimálnosť. V prípade architektúr MIMD nie je výhodné vykonávať úlohu s expanzívnym paralelizmom bez akéhokoľvek riadenia, pretože výkonnosť je daná nielen počtom procesorov, ale tiež komunikačnými nákladmi [2].
- **Údajová dekompozícia** – je funkčne jednoduchá, keďže je tá istá úloha vykonávaná na nezávislých údajových množinách. Zaťaženie procesorov je

závislé na dekompozícii samotných údajov. Ak na základe dekompozície údajov možno dosiahnuť funkciu, ktorá je do takej miery jednoduchá, aby bolo možné použiť architektúru SIMD, potom je možné dosiahnuť veľmi výkonný výpočet [2].

Z tejto klasifikácie je zrejmé, že zjednocujúcim podstatným konceptom pri dekomponovaní úloh na podúlohy je **dĺžka trvania jednotlivých podúloh**. V ideálnom prípade by malo vykonanie všetkých podúloh trvať **rovnaký čas**, čo vedie k stavu, keď podúlohy nemusia v systéme čakať na dokončenie dlhšej časti výpočtu.

Autori Grama, Gupta et al. v [44] uvádzajú štyri fundamentálne skupiny techník dekompozície úlohy na podúlohy:

- **Rekurzívna dekompozícia** – je metóda dekompozície založená na problémoch riešiteľných pomocou stratégie rozdeľuj a panuj. V tomto modeli dekompozície je úloha rozdelená na skupinu nezávislých podúloh. Každá z týchto podúloh je dekomponovaná rekurzívnym spôsobom na menšie podúlohy. Takéto rekurzívne delenie podúloh na menšie podúlohy je vykonávané pokiaľ nedosiahneme požadovanú minimálnu veľkosť podproblémov.
- **Dátová dekompozícia –** je vhodná pri práci s algoritmami, ktoré pracujú s veľkými dátovými štruktúrami. V tomto modeli je úloha dekomponovaná v dvoch krokoch. V prvom kroku sú dáta, nad ktorými prebiehajú výpočty rozdelené na menšie časti. V druhom kroku je toto rozdelenie dát využité na rozdelenie výpočtov do podúloh. Dátová dekompozícia môže byť zavedená pri:
    - **Vstupných dátach –** je vhodné využiť v prípade, že výstup z behu programu je funkciou vstupu programu.
    - **Výstupných dátach** – takúto dekompozíciu je možné použiť ak každá časť výstupu môže byť vypočítaná nezávisle od ostatných častí.
    - **Dátach v programe** – v prípade, že vstupné dáta sú transformované na dáta, ktoré sa nachádzajú len v programe a tie sú následne transformované na výstupné dáta, vieme dátovú dekompozíciu aplikovať v časti programu po transformácii vstupných dát na programové dáta.

Údajová dekompozícia uvedená v [2] je ekvivalentom dátovej dekompozície podľa [44].

- **Dekompozícia bádaním** (z angl. exploratory decomposition) – je model dekompozície využívaný pri úlohách, ktorých pridelené výpočty sú zamerané na prehľadávanie priestoru s cieľom nájdenia riešení problému. Pri dekompozícii bádaním, je potrebné rozdeliť prehľadávaný priestor na menšie časti a každú z týchto častí priestoru prehľadávame súbežne pokiaľ nie je nájdené riešenie problému.
- **Dekompozícia domnienkou** (z angl. Speculative decomposition) – je použiteľná v prípade, keď sa program môže vydať jednou z niekoľkých možných výpočtových ciest. Príkladom je algoritmus, v ktorom jedna podúloha vykonáva výpočet, ktorého výsledok je používaný pri rozhodovaní aký druh výpočtu sa vykoná v nasledujúcej časti programu. Všetky možnosti, pre ktoré sa môže algoritmus vo svojom behu rozhodnúť, sú paralelne preskúmané.

Tieto modely sa navzájom nevylučujú – často sú kombinované do **hybridných dekompozičných metód** [44]. Ak je výpočet štruktúrovaný v niekoľkých krokoch, je možné rôzne modely dekompozície uplatniť v rôznych krokoch výpočtu.

1.4 ZRÝCHLENIE A EFEKTIVITA PARALELNÉHO VÝPOČTU

Ak máme sekvenčný čas výpočtu vykonávaného na jednom procesore $T(n,1)$ a čas výpočtu toho istého problému na p procesoroch $T(n,p)$, je prirodzené vypočítať **zrýchlenie** výpočtu $S(n,p)$ ako pomer:

$$S(n,p) = T(n,1) / T(n,p) \tag{1.4}$$

Zrýchlenie dosiahnuteľné na paralelnom počítači však môže byť značne ohraničené existenciou malej časti sekvenčného kódu, ktorú **nie je možné paralelizovať** [2].

Amdahlovo pravidlo je matematický model, podľa ktorého sa určuje maximálne zrýchlenie sekvenčnej aplikácie po jej paralelizácii. Tento matematický model neberie do úvahy škálovateľnosť úlohy, veľkosť úlohy považuje za konštantnú. Toto pravidlo je aj napriek tomu využiteľné. Uvádzame ho vo vzťahu 1.5 [29]:

$$S(P) = 1 / (\alpha + 1 - \alpha/P) < 1/\alpha \tag{1.5}$$

pričom S vo vzťahu označuje zrýchlenie, α označuje podiel z celkového času výpočtu stráveného výpočtom neparalelizovateľnej časti programu a P je počet procesorov systému, na ktorom výpočet prebieha.

Druhým podstatným pravidlom je **Gustafsonovo pravidlo**, uvedené vo vzťahu 1.6 [30]. Zrýchlenie paralelného výpočtu v škálovateľných systémoch je možné opísať Gustafsonovým pravidlom.

$$S(P) = \alpha + P(1 - \alpha) \tag{1.6}$$

pričom rovnako ako pri Amdahlovom pravidle S označuje zrýchlenie, α označuje podiel z celkového času výpočtu stráveného výpočtom neparalelizovateľnej časti programu a P označuje počet procesorov systému.

Efektivita E paralelného programu je definovaná ako [21]:

$$E = S / P \tag{1.7}$$

kde S označuje zrýchlenie a P je počet procesorov použitých pri výpočte.

2 HRANOVÉ TROJOFRBOVANIE SNARKOV

V tejto práci sa zaoberáme riešením vybraných ťažkých problémov z teórie grafov, pričom za ťažké problémy považujeme tie, ktoré sú neriešiteľné v polynomiálnom čase.

Na rozdiel od úloh riešiteľných s lineárnou časovou zložitosťou (napríklad vyhľadanie najmenšieho prvku v nezotriedenom poli) alebo úloh s polynomiálnou časovou zložitosťou (napríklad výberové triedenie prvkov poľa), problém **regulárneho hranového ofarbovania kubických grafov**, rozoberaný v tejto kapitole, má úplnú nepolynomiálnu časovú zložitosť (NP-úplný problém) [12].

NP-úplný problém je taký, ktorý patrí zároveň do nepolynomiálnej a nepolynomiálnej-ťažkej triedy zložitosti. Všetky problémy z tejto triedy sú riešiteľné v polynomiálnom čase na nedeterministickom Turingovom stroji [19].

Táto práca je zameraná na hranové ofarbovanie veľkých množín grafov, pričom ide o konkrétny typ grafov, pri ktorých je samotné hranové ofarbovanie problematické - **snarky**.

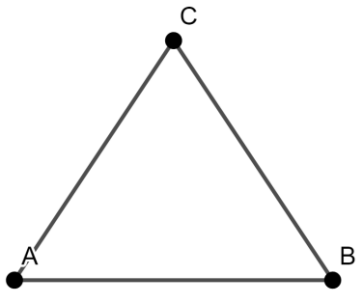
2.1 KUBICKÉ GRAFY

Graf *G* je zložený z [7]:

- **vrcholov** - prvky množiny *V(G)* – z anglického *vertices*. Na obrázku 2.1 sú vrcholy označené veľkými písmenami *A*, *B* a *C*.
- **hrán** – prvky množiny *E(G)* – z anglického *edges* – hrana je spojením medzi dvoma vrcholmi, preto je možné označiť ju práve pomocou dvoch vrcholov, ktoré spája.

Graf *G* je dvojica množín *V* a *E*, pričom prvky množiny *E* sú dvojprvkovými podmnožinami množiny *V* [9]:

$$G = (V, E) \text{ pričom } E \subseteq [V]^2 \tag{2.1}$$



Obrázok 2.1 – Jednoduchý graf

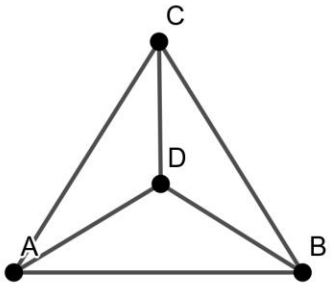
V práci budeme graf reprezentovať okrem diagramov, ako obrázok 2.1, aj **maticami susedností grafu** *G*. Matica susedností *A* grafu *G* je štvorcová matica *n*n* pričom *n* predstavuje počet vrcholov grafu. Riadky a stĺpce tejto matice sú indexované vrcholmi grafu a pre naše použitie budeme v matici uvádzať len hodnoty 1 a 0. Hodnota 1 je na mieste v matici susedností zapísaná vtedy, keď medzi vrcholmi, pomocou ktorých je toto miesto indexované, existuje hrana. V inom prípade je miesto zaplnené hodnotou 0.

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Obrázok 2.2 – Matica susedností grafu na obrázku 2.1

Základným pojmom, pomocou ktorého sme v našej práci vymedzili skupinu grafov, s ktorými budeme ďalej pracovať je **stupeň vrcholu grafu**. Ak je vrchol *V* stupňa 3, označujeme ho ako **deg(V) = 3**. Tento pojem označuje počet hrán, ktoré do vybraného vrcholu vchádzajú resp. z neho vychádzajú. **Najvyšší** (maximálny) stupeň vrcholu v grafe *G* označujeme ako ***Δ(G)***. V práci sa nebudeme sústreďovať na orientované grafy, v ktorých hrany grafov majú definovaný smer.

V tejto práci sa zaoberáme výhradne **kubickými grafmi**. Graf je kubický práve vtedy, keď sú všetky jeho **vrcholy stupňa tri** (obrázok 2.3). Veľké množstvo problémov z oblasti teórie grafov je riešiteľných práve vtedy, keď sú riešiteľné na množine kubických grafov [18].

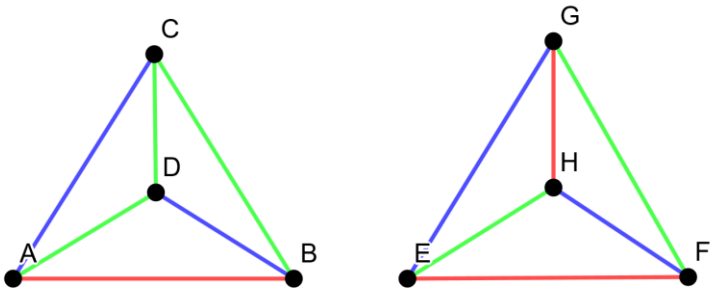


Obrázok 2.3 – Príklad kubického grafu

2.2 REGULÁRNE HRNOVÉ OFARBOVANIE KUBICKÝCH GRAFOV

Hranovým ofarbením grafu myslíme priradenie farieb jednotlivým hranám grafu. Takéto ofarbenie považujeme za regulárne práve vtedy, keď sa v grafe nenachádza žiadny **konflikt**, tj. so žiadnym z vrcholov grafu nesusedia dve alebo viac hrán ofarbených rovnakou farbou.

Najmenší počet farieb použiteľných pri regulárnom hranovom ofarbovaní grafu G označuje **hranový chromatický index grafu $\chi'(G)$** [9].



Obrázok 2.4 – Neregulárne (vľavo) a regulárne (vpravo) ofarbenie kubického grafu

Platí **Vizingova veta** [9], ktorá hovorí že minimálny potrebný počet farieb na ofarbenie grafu je v intervale $\langle \Delta(G), \Delta(G)+1 \rangle$. Formálny zápis Vizingovej vety o minimálnom potrebnom počte farieb:

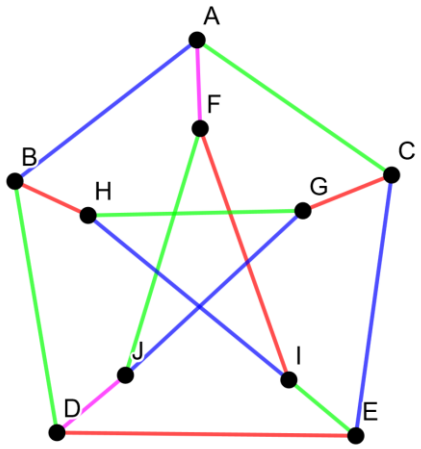
$$\Delta(G) \leq \chi'(G) \leq \Delta(G)+1 \qquad (2.2)$$

kde $\Delta(G)$ je maximálny stupeň vrcholu v grafe G a $\chi'(G)$ je hranový chromatický index grafu G. Pri kubických grafoch, kde je každý vrchol stupňa tri, uvažujeme o troch alebo štyroch farbách potrebných pre ofarbenie daného grafu.

2.3 HRANOVO 3-NEOFARBITEĽNÉ GRAFY

Ak zvolíme z množiny kubických grafov náhodným výberom graf G, s vysokou istotou bude chromatický index tohto grafu rovný trom. Takýto kubický graf voláme **hranovo 3-ofarbiteľný** [11]. Poznáme však malú množinu kubických grafov, pre ktoré na regulárne hranové ofarbenie potrebujeme použiť štyri farby. Grafy z tejto množiny kubických grafov sa nazývajú **hranovo 3-neofarbiteľné** alebo **snarky**[1] [3].

Najmenší známy graf, patriaci do skupiny snarkov je Petersenov graf zobrazený na obrázku 2.5.



Obrázok 2.5 – Regulárne ofarbenie Petersenovho grafu pomocou štyroch farieb

Táto skupina grafov sa pri samotnom hranovom ofarbovaní správa problematicky. Snarky majú chromatický index $\chi'(G) = 4$. Aby sme však zistili či je graf G snarkom, musíme ho hranovo ofarbiť s využitím troch farieb. Z toho vyplýva, že algoritmus, pomocou ktorého chceme graf G ofarbiť, musí prejsť všetkými možnosťami hranového trojofarbenia grafu G, aby vylúčil možnosť existencie regulárneho hranového trojofarbenia tohto grafu. Až po prejdení všetkých možných hranových ofarbení troma farbami vieme o grafe povedať či je alebo nie je snarkom.

[1] Pomenovanie bolo prevzaté z básne Lewisa Carolla s názvom *The Hunting of the Snark (An Agony in 8 Fits)*. Samotnému výrazu snark však autor nepriradil žiadny význam. V polovici 19. storočia bol výraz "snarking" používaný na všeobecný opis zvuku.

V tabuľke 2.1 uvádzame počet nájdených snarkov vygenerovaných pomocou nástrojov *Snarkhunter* a *Minibaum* [17]. V čase písania tejto práce nie je možné vygenerovať všetky grafy niektorých veľkostí, preto bola pre tieto skupiny grafov vygenerovaná len vzorka, ktorá je v tabuľke 2.1 označená znakom ‘+‘.

Tabuľka 2.1 – Počet vygenerovaných snarkov

Počet vrcholov grafu	Počet grafov
10	1
18	2
20	6
22	31
24	155
26	1 297
28	12 517
30	139 854
32	1 764 950
34	25 286 953
36	404 899 916
38	19 775 768+
40	25+

Pro potreby zjednodušenia čitateľnosti textu budeme v ďalších častiach práce používať výraz **(hranové) ofarbenie grafu** vo význame **hranové trojofarbenie grafu**.

3 ANALÝZA STAVU RIEŠENIA ÚLOH V OBLASTI TEÓRIE GRAFOV

V tejto kapitole sa venujeme analýze súčasného stavu riešenia niektorých ťažkých problémov z oblasti teórie grafov. Sústredíme sa najmä na výsledky dosiahnuté v oblasti grafovej **ofarbiteľnosti**.

Grafy môžeme regulárne hranovo ofarbovať rôznymi spôsobmi - pomocou rôznych algoritmov. Štandardným algoritmom, využívaným na hranové ofarbovanie grafov, je **hranový backtracking**. Pre porovnanie uvádzame aj novšie prístupy k problematike - Kowalikov **algoritmus ofarbovania hrán** EdgeColor [13] a **heuristika premiestnenia konfliktných vrcholov** (angl. Conflict vertex displacement, CVD) autorov Fiolu a Vilaltellovej [14].

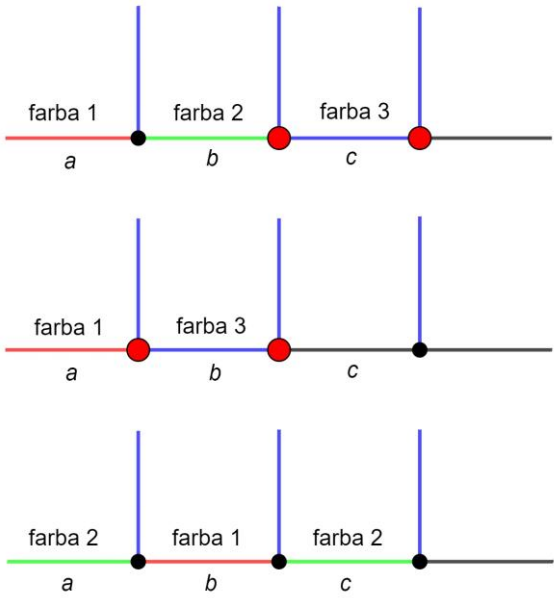
3.1 ALGORITMUS HRANOVÉHO BACKTRACKINGU

Tento algoritmus pracuje na princípe postupného ofarbovania hrán grafu vopred určenou **postupnosťou troch farieb**. Algoritmus sa pri nájdení rozporu v ofarbovaní grafu **vráti na predošlú hranu**, ktorú prefarbí a pokračuje v ďalšom ofarbovaní grafu. Ak ani jedno z možných ofarbení problematickej hrany nie je regulárne, algoritmus sa vracia k hrane **predchádzajúcej obe prefarbované hrany**. Algoritmus pokračuje v tomto postupe pokiaľ nie je celý graf regulárne ofarbený alebo pokiaľ algoritmus neprejde všetkými možnosťami ofarbenia grafu. Časová zložitosť hranového backtrackingu je $O(2^{n-1})$, pričom n je počet vrcholov zadaného grafu.

Samotný algoritmus reprezentujú tieto kroky:

- Algoritmus postupne berie 3 farby a ofarbuje za sebou nasledujúce hrany pokiaľ:
    - graf je regulárne ofarbený,
      alebo
    - nastal konflikt v ofarbovaní grafu.
- V prípade, keď nastane konflikt pri ofarbovaní sa algoritmus vracia na už ofarbené hrany a prefarbuje ich ďalšou farbou v poradí pokiaľ:
    - konflikt je vyriešený – v tomto prípade môže algoritmus pokračovať v ďalšom hranovom ofarbovaní podľa prvého kroku algoritmu,

- všetky možnosti hranového trojofarbenia boli neregulárne. V tomto prípade je ofarbovaný graf snarkom.



Obrázok 3.1 – Príklad hranového backtrackingu

Na obrázku 3.1 je zobrazená časť kubického grafu ofarbovaná pomocou algoritmu hranového backtrackingu. Algoritmus postupne ofarbuje hrany *a*, *b* a *c* určenou postupnosťou farieb. V našom prípade je farba 1 červená, farba 2 zelená a farba 3 modrá. Keďže niektoré z hrán na obrázku už boli ofarbené, na hrane *c* dôjde k iregulárnosti ofarbenia (s vrcholmi pri hrane susedia dve modré hrany). Algoritmus sa preto vráti na hranu *b*, ofarbenú farbou 2, ktorú prefarbí ďalšou z farieb v poradí (farbou 3). Tu však nastáva nový konflikt a preto sa algoritmus posúva na hranu *a* a zopakuje rovnaký postup pričom začína od farby 2.

V našej práci budeme ďalej pracovať len s algoritmom hranového backtrackingu – tento algoritmus je jednoduchý na implementáciu a na grafoch rôznych veľkostí funguje s efektivitou, ktorá v niektorých prípadoch prevyšuje efektivitu ostatných algoritmov opísaných v tejto kapitole.

3.2 ALGORITMUS OFARBOVANIA HRÁN

Kowalik [13] vylepšil algoritmus autorov Beigela a Eppsteina [15], v ktorom prezentujú ofarbovanie hrán grafu pomocou troch farieb. Časová zložitosť **algoritmu ofarbovania hrán** je $O(1.344^n)$, kde *n* označuje počet vrcholov ofarbovaného grafu.

Samotný algoritmus sa skladá z troch procedúr:

- procedúra **EdgeColor**,
- procedúra **FittingMatching**,
- procedúra **SetSwitches**.

V algoritme sú použité výrazy uvedené v tabuľke 3.1.

Tabuľka 3.1 – Výrazy použité v algoritme ofarbovania hrán

subkubický graf	graf, ktorého maximálny stupeň vrcholu je $\Delta(G) \leq 3$
matching	množina navzájom nesusediacich hrán
cyklus	cesta hranami a vrcholmi grafu, ktorej počiatočný vrchol je aj jej koncovým vrcholom. V algoritme využívame **trojcyklus**, ktorý obsahuje práve tri hrany a **štvorcyklus**, ktorý obsahuje práve štyri hrany.

3.2.1 PROCEDÚRA EDGECOLOR

Procedúra generuje zo zadaného subkubického grafu *G* množinu subkubických grafov *A*, pričom žiadny z grafov z tejto množiny neobsahuje trojcykly ani štvorcykly. Ak je aspoň jeden z grafov tejto množiny ofarbiteľný troma farbami, pôvodný graf je tiež ofarbiteľný troma farbami. Pre každý graf z množiny *A* volá procedúra *EdgeColor* procedúru *FittingMatching*. Výstupom z procedúry *EdgeColor* je hodnota TRUE v prípade ak je graf ofarbiteľný troma farbami alebo FALSE v iných prípadoch.

```
INPUT: subkubický graf G
OUTPUT: TRUE v prípade, že je graf G regulárne ofarbiteľný troma farbami,
v inom prípade FALSE

IF existuje v z V(G) taký, ktorého stupeň ∈{0,1} THEN
      RETURN EdgeColor(G-v)
```

```
ELSE IF existuje uv z E(G) také, ktorého stupeň u = stupeň v = 2 THEN
      RETURN EdgeColor(G-uv)
ELSE IF graf G obsahuje trojitú hranu uv THEN
      RETURN EdgeColor(G-{u,v})
ELSE Predpokladajme, že u1(resp.v1) je susedným vrcholom u rozdielnym od v
      IF u1=v1 THEN
            RETURN FALSE
      ELSE
            RETURN EdgeColor(G-{u,v}+u1v1)
ELSE IF existuje 3-oj cyklus C THEN
Predpokladajme, že vieme vytvoriť graf G´ ako kontrakciu V(C) do jedného
vrcholu
      RETURN EdgeColor(G´)
ELSE IF existuje cesta xuzvy taká, že stupeň u = stupeň v = 2 THEN
      z´ ← susedný vrchol vrcholu z rozdielny od u a v
      RETURN EdgeColor(G-{z,v}+uy) alebo EdgeColor(G-{u,z,v}+xz´)
ELSE IF existuje 4-or cyklus C=xyzu pričom stupne vrcholov x, y a z = 3
a stupeň vrcholu u = 2 THEN
Predpokladajme, že x´ (v resp. y´, z´) sú susedné vrcholy vrcholu x (v
resp. y, z) mimo cyklu C
      RETURN EdgeColor(G-V(C)+x´y´) alebo EdgeColor(G-V(C)+z´y´)
ELSE IF existuje 4-or cyklus C=xyzu pričom stupne vrcholov x, y, z a u = 3
THEN
Predpokladajme, že x´ (v resp. y´, z´, u´) sú susedné vrcholy vrcholu x
(v resp. y, z, u) mimo cyklu C
RETURN EdgeColor(G-V(C)+{x´y´,u´z´}) alebo EdgeColor(G-V(C)+{x´u´,y´z´)
ELSE
      RETURN FittingMatching(G0,G,0)
```

Obrázok 3.2 – Pseudokód procedúry EdgeColor [13]

3.2.2 PROCEDÚRA FITTINGMATCHING

Vstupnými parametrami procedúry sú subkubické grafy G_0, G také, že $G \subseteq G_0$. Tretím vstupným parametrom je *matching M* (množina navzájom nesusediacich hrán) na grafe G_0 taká, že $V(M) \cap V(G) = \emptyset$ (nemajú spoločné vrcholy). Graf je hranovo ofarbiteľný troma farbami, vtedy keď obsahuje vhodný *matching* [13]. Každý vhodný *matching* je dokonalý *matching* (množina navzájom nesusediacich hrán obsahujúca všetky vrcholy). Každá trojica vrcholov v $V(G_0) - V(G)$ obsahuje množinu navzájom nesusediacich hrán (má *matching*). Každý vrchol B grafu G, ktorého stupeň v grafe G_0

deg(B) = 3, označujeme ako *forced*. Ako *switch* označujeme štvorhrannú cestu v grafe *G*, ktorej iba vnútorné vrcholy nadobúdajú hodnotu *forced*. Hrana *e* v grafe *G* bude označená ako *allowed* ak sú oba jej koncové vrcholy *forced* a zároveň nepatrí *switch*. Hodnota hrany *weight* je súčet stupňov vrcholov hrany. Keď je každá časť grafu *G* *switch* je volaná procedúra *SetSwitches*.

Obrázok 3.3 – *Switch* (vľavo) a *allowed* hrany E, F a F, G (vpravo)

```
IF každá prepojená časť grafu G je switch THEN
      RETURN SetSwitches(G₀, G, M)
ELSE IF existuje forced vrchol v z V(G) ktorého stupeň = 0 THEN
      RETURN FALSE
ELSE IF existuje vrchol, ktorý nie je forced, v z V(G) ktorého stupeň = 0
THEN
      RETURN FittingMatching(G₀, G–{v}, M)
ELSE IF existuje vrchol, ktorý nie je forced, v z V(G) ktorého stupeň = 1
THEN
      u ← susedný vrchol vrcholu v v grafe G
      RETURN FittingMatching(G₀, G–{u,v}, M ∪ {uv})
ELSE
      uv ← ktorákoľvek allowed hrana z grafu G s najvyššou váhou
      RETURN FittingMatching(G₀, G–{u,v}, M ∪ {uv}) alebo
FittingMatching(G₀, G–uv, M)
```

Obrázok 3.4 – Pseudokód procedúry FittingMatching [13]

3.2.3 PROCEDÚRA SETSWITCHES

Procedúra overuje či G_0 obsahuje vhodný *matching* M_0 taký, že $M_0 = M \cup N$ pre $N \subseteq E(G)$.

```
M´ ← M
FOR EACH switch s = xuvy v grafe G DO
      M´ ← M ∪ {xu, vy}
WHILE G₀ – M´ obsahuje nepárny cyklus C DO
IF existuje switch s alebo kombinácia switchov s1 a s2 ktoré zlepšia M´
      THEN M´ ← M´ ⊕E(s) (resp. M´ ← M´⊕(E(s1) ∪ E(s2))
ELSE
      RETURN FALSE
RETURN TRUE
```

Obrázok 3.5 – Pseudokód procedúry SetSwitches [13]

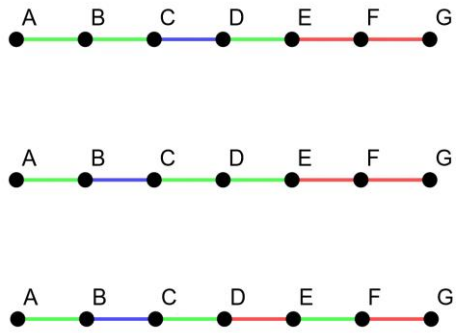
3.3 HEURISTIKA PREMIESTNENIA KONFLIKTNÝCH VRCHOLOV

Základným princípom tejto heuristiky je **premiestnenie konfliktov na grafe** [14]. Konfliktom v tomto prípade myslíme ofarbenie viacerých hrán pri jednom vrchole rovnakou farbou, čo spôsobuje iregulárnosť ofarbenia grafu.

Algoritmus začína **náhodným ofarbením** všetkých hrán kubického grafu. V hlavnom programe následne prebieha niekoľko krokov. Algoritmus náhodne vyberie vrchol pri ktorom nastáva konflikt. Jedna z hrán je považovaná za začiatok **dvojhranového reťazca**. Tento reťazec obsahuje hranu, ktorej ofarbenie spôsobuje konflikt a hranu ofarbenú inou farbou. Pri výmene ofarbenia týchto dvoch hrán nastáva **premiestnenie konfliktu v rámci grafu** [14].

Túto operáciu opakujeme pokiaľ:

- nedosiahneme miesto v grafe, kde sa nachádza ďalší konflikt, čím sa oba konflikty **vzájomne zrušia**. Na obrázku 3.6 uvádzame príklad vzájomného zrušenia konfliktov. Konflikt pri vrchole B sa zruší pomocou výmeny farieb hrán dvojhranového reťazca ohraničeného vrcholmi B, D. Táto výmena vytvorí konflikt pri vrchole D, ktorý je následne odstránený výmenou farieb hrán v dvojhranovom reťazci ohraničenom vrcholmi D, F. Táto výmena súčasne zruší konflikt pri vrchole F.



Obrázok 3.6 – Vzájomné zrušenie konfliktov pri vrcholoch B a F

- alebo počet týchto výmen nedosiahne **stanovený limit** – zväčša počet vrcholov grafu.

Takýto postup sa opakuje pre každý konflikt pokiaľ nie je graf regulárne ofarbený troma farbami alebo je dosiahnutý **fixný repetičný limit** R zavedený pre vyhnutie sa nekonečnej slučke. Tento limit označuje počet zvolených konfliktne ofarbených hrán a vynuluje sa vždy, keď sa v grafe zníži počet konfliktov. Keď je limit R dosiahnutý, postup je spustený odznova s novým náhodným ofarbením grafu.

Maximálny počet nových spustení programu je daný **iteračným limitom** L. Ak po dosiahnutí tohto limitu nenájde algoritmus regulárne ofarbenie daného kubického grafu, považujeme tento graf za snark.

Zložitosť algoritmu je autormi meraná na $O(n^2)$ [14].

```
INPUT: graf G, hodnoty pre maxL a maxR
OUTPUT: TRUE v prípade, že je graf G regulárne ofarbiteľný troma farbami,
v inom prípade FALSE

DO
      CVD(G, maxR)
      Zvýš L o 1
WHILE CVD(G) = TRUE alebo L = maxL

IF L = maxL THEN
      RETURN FALSE
ELSE
      RETURN TRUE
```

Obrázok 3.7 – Pseudokód hlavnej procedúry heuristiky CVD

```
CVD(G, maxR):

náhodne ofarbi graf G pomocou troch farieb
skontroluj ofarbenie a nájdi konflikty

WHILE R ≠ maxR alebo existujú konflikty DO
      náhodne vyber jeden konflikt
      DO
            predpokladajme že hrany u a v sú ofarbené rovnakou farbou
            vyber hranu x susediacu s hranou u (v resp. v) takú,
            že x ≠ v (v resp. u)
            vymeň x a u (v resp. v)
            zvýš n o 1
      WHILE n = počet vrcholov grafu G alebo zrušenie konfliktu
      IF n = počet vrcholov grafu G THEN
            Zvýš R o 1
      ELSE
            R = 0

IF neexituje konflikt THEN
      RETURN TRUE
ELSE
      RETURN FALSE
```

Obrázok 3.8 – Pseudokód vedľajšej procedúry heuristiky CVD

4 MODELY PARALELNÝCH VÝPOČTOV NA OFARBOVANIE GRAFOV

Algoritmy, zaoberajúce sa riešením problémov z oblasti teórie grafov, sú z pohľadu paralelizácie na viacprocesorových systémoch náročné [20, 24]. Pri riešení týchto problémov sa stretávame s dátovými problémami, softvérovými a hardvérovými problémami. Pri paralelizácii týchto **úloh z pohľadu dát** sú podstatnými problémami [28]:

- **Výpočty orientované na dáta** – výpočty v mnohých grafových algoritmoch sú sústredné na štruktúru grafu (vrcholy, hrany), na ktorom prebiehajú. Tieto operácie sú zložitejšie na priamu interpretáciu v kóde programu. Príkladom takéhoto problému je reprezentácia grafu pomocou matice susedností (dvojrozmerného poľa), s ktorou sa v rámci programu ťažko narába.
- **Nepravidelné problémy –** dáta v grafových úlohách sú často nepravidelné. Práve táto nepravidelnosť sťažuje dekompozíciu problému na efektívne paralelné spracovanie. Matica susedností grafu môže mať mnoho miest, v ktorých sa nachádzajú zhluky hodnôt a iné, úplne prázdne miesta.
- **Vysoký pomer času prístupu k dátam a času výpočtov –** algoritmy na grafoch sú často založené na práci so štruktúrou grafu, preto je pomer medzi prístupom k dátam a výpočtami vyšší. Táto vlastnosť grafových algoritmov preto často vedie k veľkému podielu z času behu algoritmu venovanému práci s pamäťou.

Pri tvorbe paralelných aplikácií, v ktorých využívame grafové algoritmy sa stretávame s niekoľkými **hardvérovými a softvérovými problémami**. Medzi tieto problémy patria [28]:

- **Granularita úlohy –** je pomer veľkosti výpočtu a komunikácie, jemnozrnný paralelizmus obsahuje malé množstvo výpočtov oproti komunikácii, naopak hrubozrnný paralelizmus obsahuje veľké množstvo výpočtov oproti komunikácii [22]. Pri vývoji paralelného programu je potrebné vedieť kedy je do programu vhodné vložiť paralelizmus. Pre operácie v grafových algoritmoch, ktorých čas behu je blízko k lineárnemu alebo lineárny, môžeme paralelizmus využiť len na veľmi nízkej úrovni.

- **Problémy s prístupom do pamäte –** paralelné grafové algoritmy sa vyznačujú problematickým prístupom do zdieľanej pamäte. Viac paralelných procesov sa môže pokúsiť o simultánny prístup na jedno pamäťové miesto.
- **Vývoj softvéru –** mnohé z problémov týkajúcich sa paralelných grafových algoritmov sú náročné na prácu s pamäťou, preto je potrebné pri programovaní venovať tomuto aspektu zvýšenú pozornosť. Pri vývoji softvéru sa odporúča využívanie knižníc špecializovaných na prácu s grafovými algoritmami.

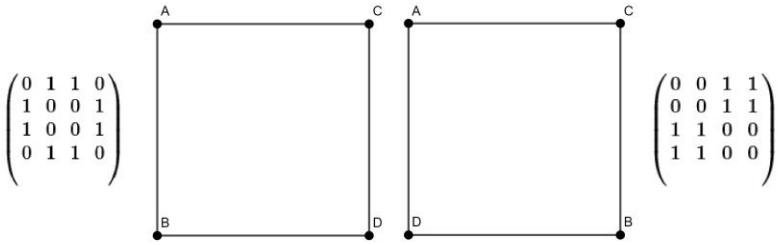
Pri skúmaní vlastností grafov je algoritmus regulárneho hranového ofarbovania grafov len **čiastočnou operáciou**. Preto je potrebné graf ofarbiť v čo najnižšom čase.

V algoritmoch, prezentovaných v tejto kapitole, pracujeme s konceptom **ofarbovania permutovaných kubických grafov**. V aplikáciách využívame programový model vlákien, pričom riešime tvorbu permutácií grafu pomocou konjugácie. Úlohu sme riešili v rámci projektu VEGA - **Algoritmy na grafoch a algebraických štruktúrach**. Na samotné ofarbovanie jednotlivých grafov používame algoritmus hranového backtrackingu prezentovaný v kapitole 3.1.

V práci sme navrhli 4 hypotézy. Začneme hypotézou H1, ktorá je východisková:

Pracovná hypotéza H1. *Algoritmus hranového backtrackingu nie je citlivý na počiatočný vrchol ofarbovania grafu. To znamená, že ak pri výpočte použijeme rôznu postupnosť hrán bude doba výpočtu algoritmu pre zadaný graf rovnaká.*

Nech je graf *G* reprezentovaný maticou susedností, ako bolo ukázané v kapitole 2.1. Ak sa potvrdí, že výpočty, ktoré používajú rôzne poradie ofarbovania hrán trvajú rovnako dlho, hypotéza bude potvrdená, v opačnom prípade bude vyvrátená. Aby sme mohli skúmať túto hypotézu, potrebujeme si pre graf *G* vytvoriť množstvo permutácií matice susedností tohto grafu. Takto vytvoríme množstvo modifikácií jedného grafu, ktoré budú mať rôznu postupnosť hrán.



Obrázok 4.1 – Príklad izomorfných grafov

Využijeme preto **izomorfizmus grafov**. Nech graf G' a zadaný graf G sú izomorfnými grafmi. Dôležitou vlastnosťou izomorfných grafov je, že ľubovoľná dvojica takýchto grafov má rozdielne matice susedností ale zhodné diagramy (zobrazené na obrázku 4.1 – uvádzame príklad nekubického grafu, pre jednoduchú vizualizáciu). V našom prípade to znamená, že ide o zhodné grafy s rozdielnou postupnosťou vrcholov a hrán. Ak je graf G reprezentovaný pomocou matice susedností A, rôznu postupnosť hrán grafu G (izomorfný graf G') vieme vytvoriť pomocou **permutácie matice susedností grafu** G. Permutáciu matice susednosti A grafu G vypočítame pomocou **konjugácie** (4.1).

$$A' = P^{-1} * A * P \tag{4.1}$$

kde A' je **permutovaná matica susedností grafu** $\boldsymbol{G}$, A je pôvodná **matica susedností grafu** $\boldsymbol{G}$, P označuje **permutačnú maticu grafu** (matica naplnená nulami s práve jednou jednotkou v každom riadku a každom stĺpci), P^{-1} označuje **transponovanú permutačnú maticu** P.

Takto získaná permutovaná matica susedností A' reprezentuje zmenené poradie hrán pôvodného grafu G. Podľa tohto poradia ofarbujeme hrany grafov v našich experimentoch.

4.1 PARALELNÉ OFARBOVANIE PERMUTÁCIÍ GRAFU

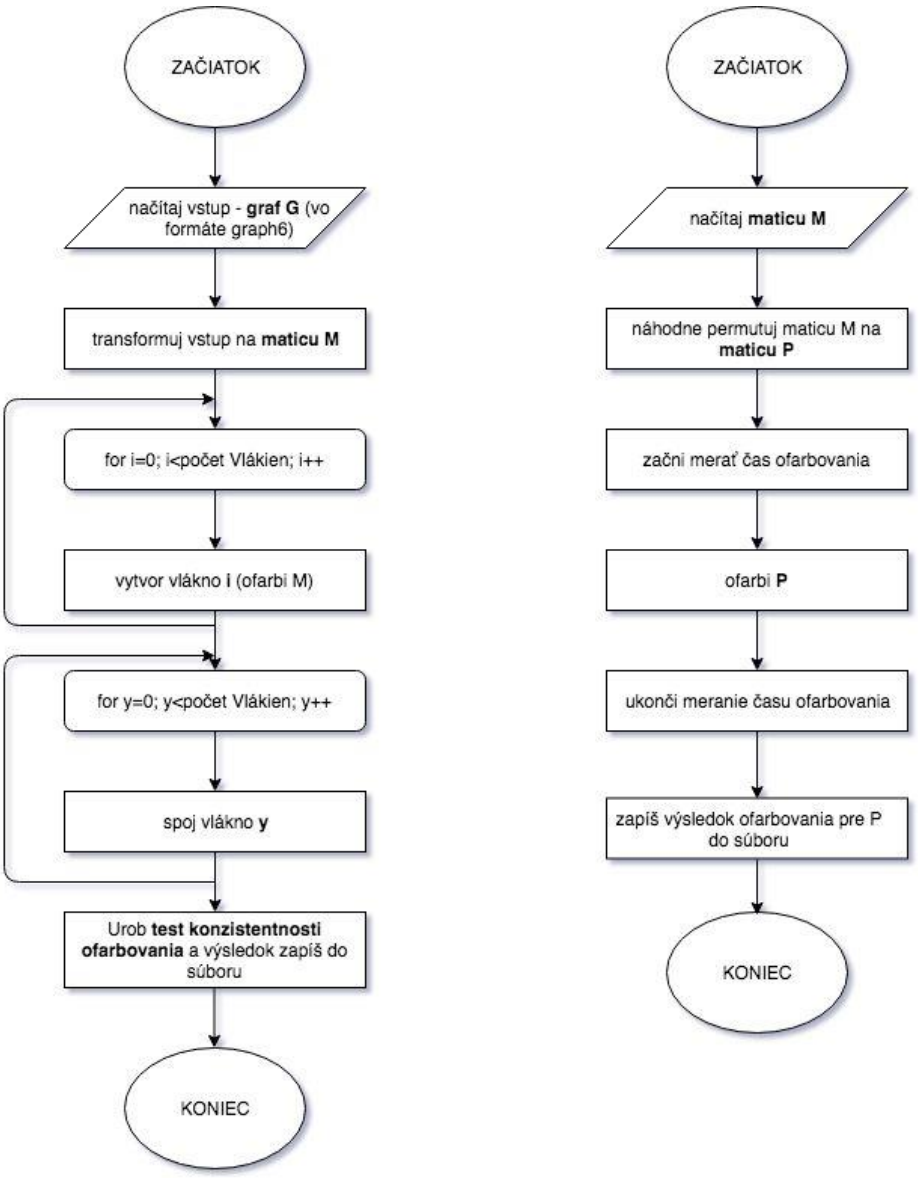
V tejto kapitole uvádzame metodiku a experimenty na potvrdenie alebo zamietnutie východiskovej hypotézy H1. Metodika a jej experimentálne overenie boli prezentované v rámci [III].

4.1.1 METODIKA ZAMERANÁ NA PARALELNÉ OFARBOVANIE PERMUTÁCIÍ GRAFU

Navrhujeme metodiku skúmania paralelného ofarbovania permutácií grafu:

- V prvom kroku potrebujeme nájsť vhodný **formát zápisu grafu** (resp. matice susedností grafu). Cieľom je nájsť taký formát zápisu grafu, ktorý zaberá najmenšie miesto v pamäti.
- Vstupný formát grafu (graf môže byť uložený napr. vo formáte graph6) **transformujeme** do tvaru, ktorý je potrebný pre funkcie programu (na matice susedností, v resp. dvojrozmerné polia naplnené celočíselnými hodnotami).

- V treťom kroku potrebujeme pre náš program zvoliť **vhodný model paralelného programovania** (vláknový model, ..., uvedené v podkapitole 1)**.**
- V každej paralelnej časti programu (napr. vlákne) sa pre vstupný graf *G* vypočíta **náhodná permutácia matice** tohto grafu. Následne túto **permutáciu ofarbíme** pomocou algoritmu hranového backtrackingu. Pre každú ofarbovanú permutáciu matice susedností grafu *G* **odmeriame čas ofarbovania**. Výsledky zapíšeme do súboru zdieľaného medzi vláknami.
- Posledný krok metodiky tvorí **test konzistentnosti ofarbovania** a jednoduchá **štatistika z behu programu**.



Obrázok 4.2 – Diagram navrhovanej metodiky, hlavná procedúra (vľavo) a procedúra Ofarbi (vpravo), ktorá je spracovávaná paralelne

Aby sme pre naše potreby dostali dostatočne širokú vzorku nameraných výsledkov, budeme počítať s ofarbením 100 až 1000 permutácií vstupného grafu. Ide o časovo náročnú úlohu vhodnú pre **paralelné spracovanie**. Všetky vytvorené permutácie matice susedností grafu *G* môžu byť ofarbované súčasne, pričom sa časy ofarbovania všetkých permutácií zapíšu do zdieľaného súboru.

Hypotézu H1 overujeme pomocou vytvorenia aplikácie na paralelné ofarbovanie permutácií grafu. **Vstupom pre aplikáciu** je jeden kubický graf zadaný vo formáte **graph6 –** úsporný spôsob zápisu grafu založený na rozložení matice susedností grafu na niekoľko jednociferných a dvojciferných čísel [27].

Výstupom pre každú permutáciu zadaného grafu je:

- **čas ofarbovania** konkrétnej permutácie grafu meraný v milisekundách,
- hodnota **TRUE** ak je permutácia grafu regulárne ofarbiteľná troma farbami, **FALSE** v inom prípade,
- **kód permutácie** grafu odvodený z permutačnej matice susedností grafu – kód udáva na ktorom mieste v riadku permutačnej matice sa nachádza hodnota 1. Permutačná matica je naplnená nulami a obsahuje práce jednu 1 v každom riadku a stĺpci.

```
90.00 ms FALSE PERMUTÁCIA: 37, 27, 59, 30, 21, 54, 24,
9, 50, 60, 71, 42, 43, 41, 7, 6, 12, 2, 4, 23, 63,
35, 64, 67, 61, 46, 31, 10, 14, 3, 40, 34, 13, 53, 62,
47, 38, 56, 11, 52, 18, 49, 22, 51, 0, 33, 29, 8, 32,
15, 58, 65, 68, 20, 1, 26, 16, 48, 69, 17, 19, 55, 39,
70, 45, 5, 66, 44, 25, 57, 28, 36,
```

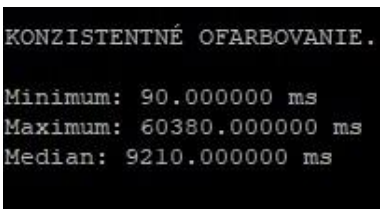
Obrázok 4.3 – **Príklad výstupu z aplikácie**

Samotný beh aplikácie sa skladá z troch hlavných krokov:

- Aplikácia pracuje s maticami susedností grafov, preto bolo ako prvé potrebné **preložiť vstup** z formátu graph6 na požadovaný formát. O toto prekladanie sa v aplikácii stará nami vytvorená funkcia, ktorej vstupom je počet vrcholov grafu a reťazec vo formáte graph6. Výstupom z tejto funkcie je matica susedností grafu.
- Matica susedností pôvodného grafu je ďalej rozposlaná na **paralelné spracovanie** pre určený počet **vlákien** aplikácie. Využívame model **POSIX threads** pričom ako parameter každému vláknu zadávame maticu susedností vstupného grafu. V každom z vlákien sa vykonáva niekoľko rôznych funkcií:

1. **Vytvorenie permutácie matice** – ako prvá sa z pôvodnej matice grafu vytvorí permutovaná matica. Túto funkciu sme zabezpečili implementáciou **konjugácie** podľa vzťahu 4.1, pričom permutačnú maticu *P* generujeme náhodne.
2. **Ofarbenie permutácie matice –** po vytvorení permutácie matice susednosti je následne graf, ktorý táto matica reprezentuje, ofarbený. Na ofarbenie grafu bol použitý algoritmus **hranového backtrackingu**. Výstupom z tohto algoritmu je hodnota TRUE, ak je graf regulárne ofarbiteľný troma farbami alebo hodnota FALSE v inom prípade.
3. **Zápis výsledku do zdieľaného súboru –** tento súbor je zdieľaný medzi všetkými vláknami aplikácie, preto bolo potrebné zabezpečiť synchronizáciu zápisu procesov do súboru. Každé vlákno do súboru zapisuje čas, za ktorý prebehlo ofarbenie permutácie grafu, hodnotu ofarbovania TRUE alebo FALSE a kód permutácie matice susedností grafu.

- **Testy konzistentnosti ofarbovania –** po dokončení práce všetkých vlákien sa následne vykoná **test konzistentnosti ofarbovania** – test, ktorého výsledkom je hodnota KONZISTENTNÉ OFARBOVANIE ak je výsledok ofarbovania všetkých permutácií zadaného grafu homogénny – ak sú všetky permutácie grafu ofarbiteľné alebo všetky permutácie grafu neofarbiteľné, v inom prípade nadobúda hodnotu NEKONZISTENTNÉ OFARBOVANIE. Po tomto teste sa vykoná jednoduchá štatistika spojená s dĺžkou ofarbovania permutácií grafu – konkrétne je vyhľadaný **minimálny** a **maximálny čas** ofarbovania permutácií a vypočíta sa **medián** všetkých časov ofarbovania permutácií zadaného grafu.



Obrázok 4.4 – Príklad testu konzistentnosti ofarbovania grafu a jednoduchej štatistiky

4.1.2 EXPERIMENTY ZAMERANÉ NA PARALELNÉ OFARBOVANIE PERMUTÁCIÍ GRAFU

Hypotézu sme v prvom kroku overovali na **Petersenovom grafe** (obrázok 2.5). Tento kubický graf sa skladá z 10 vrcholov a 15 hrán pričom o ňom vieme, že je nie je regulárne ofarbiteľný pomocou troch farieb – je snark. Petersenov graf sa však pri meraniach ukázal byť nedostatočným. Pri spustení aplikácie so 100 permutáciami matice susedností Petersenovho grafu bol beh ofarbovacieho algoritmu pre všetky z nich ukončený v čase pod milisekundu s homogénnym výsledkom FALSE.

Preto sme ako druhý graf na overenie hypotézy zvolili graf **Midzi 72**. Tento graf je snark, tak isto ako Petersenov graf. Skladá sa zo 72 vrcholov, pričom bol považovaný za graf s nadpriemerne dlhým časom ofarbovania, čo z neho urobilo vhodný cieľ pre naše testovanie [17].

Aplikáciu sme pri testovaní pomocou tohto grafu spúšťali v **Centre pre vysokovýkonné počítane** Univerzity Mateja Bela v Banskej Bystrici (z angl. High Performance Computing Center, HPCC UMB). Dostupná architektúra v Centre pre vysokovýkonné počítanie je zložená z:

- 22 serverov s 12 jadrami, 48GB RAM, bez GPU
- 2 servery s 12 jadrami, 48GB RAM - každý s GPU nVidia Tesla M2070 s 448 jadrami a 6GB RAM
- 3 servery so 16 jadrami, 64GB RAM, bez GPU
- 6 serverov so 16 jadrami, 128GB RAM, bez GPU
- 3 servery so 16 jadrami, 64GB RAM - každý s 2x nVidia Tesla K20 s 2496 CUDA jadrami a 5GB RAM
- **Spolu**: 560 fyzických jadier

Hypotézu sme overovali na troch veľkostiach problému. V prvej inštancii sme spustili program na ofarbenie 100 permutácií grafu Midzi 72. V rámci Centra pre vysokovýkonné počítanie v Banskej Bystrici nám na tento výpočet postačovala lokálna, používateľská architektúra.

V druhej inštancii sme spustili program s 500 permutáciami rovnakého problému a v tretej inštancii s 1000 permutáciami. Pre problémy s vyšším počtom permutácií matice susedností vstupného grafu sme potrebovali použiť rozšírenú architektúru. Konkrétne sa jednalo o použitie 15 výpočtových uzlov zo systému.

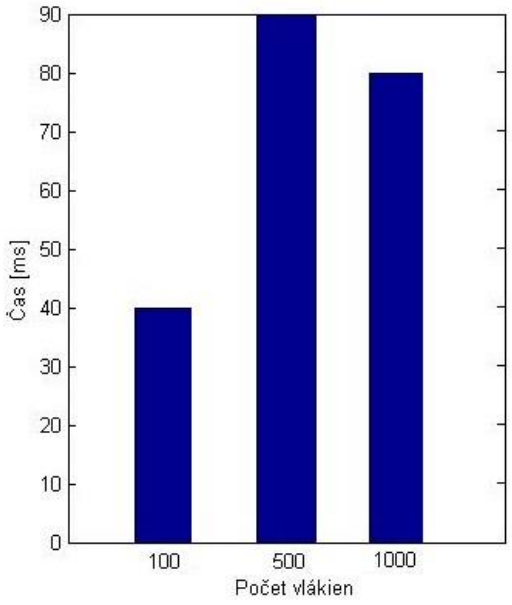
Pri každom behu programu sme merali najnižší čas ofarbovania grafu, najvyšší čas ofarbovania vstupného grafu a hodnotu mediánu pre ofarbovanie množiny permutovaných grafov.

Najkratšie časy ofarbovania grafov sa ukázali byť takmer zhodné – vo všetkých veľkostiach úlohy boli permutované grafy, ktorých čas ofarbovania bol najkratší < 1 milisekunda.

Významné rozdiely v časoch ofarbovania sa ukázali až pri **porovnávaní maxím** (resp. mediánov) z časov výpočtov algoritmu. V tabuľke 4.1 a na obrázku 4.5 prezentujeme porovnania najdlhších časov výpočtov pre jednotlivé veľkosti problému. V tabuľke 4.2 a na grafe na obrázku 4.6 porovnávame vypočítané mediány z časov pre všetky tri veľkosti problému.

Tabuľka 4.1 – Porovnanie maxím časov výpočtu algoritmu

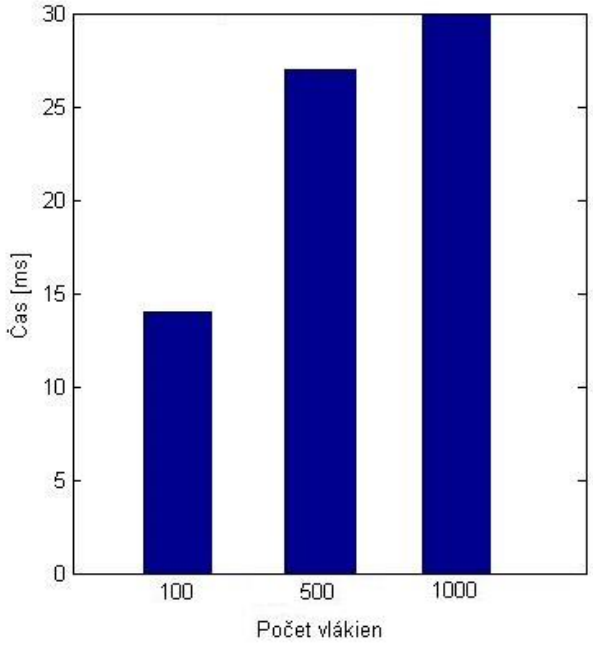
Počet permutácií matice susedností grafu	Maximum z časov výpočtu algoritmu v milisekundách
100	40
500	90
1000	80



Obrázok 4.5 – Porovnanie maxím časov výpočtu algoritmu

Tabuľka 4.2 – Porovnanie mediánov časov výpočtu algoritmu

Počet permutácií matice susedností grafu	Medián z časov výpočtu algoritmu v milisekundách
100	14
500	27
1000	30



Obrázok 4.6 – **Porovnanie mediánov časov výpočtu algoritmu**

Pri meraní sa ukázalo, že **doba výpočtu** algoritmu hranového backtrackingu je pre rôzne permutácie **výrazne odlišná**, čím sme vyvrátili pracovnú hypotézu H1. Namerané hodnoty sa pohybovali v intervale [0,3; 98] ms. Merania pre 100, 500 a 1000 permutácií sme zopakovali päťnásobne pre každú z veľkostí problému.

4.2 PARALELNÉ OFARBOVANIE PERMUTÁCIÍ MNOŽINY GRAFOV

Pri analýze výsledkov z meraní prezentovaných v kapitole 4.1.1 sme zistili, že exitujú také permutácie grafu Midzi 72, ktorých hranové ofarbovanie trvá menej ako milisekundu. Ak pre každý graf existuje také poradie ofarbovania hrán (resp. množina poradí) bolo by možné **minimalizovať čas ofarbovania** celých množín grafov práve zvolením takejto permutácie. Táto úvaha viedla k vytvoreniu pracovnej hypotézy H2.

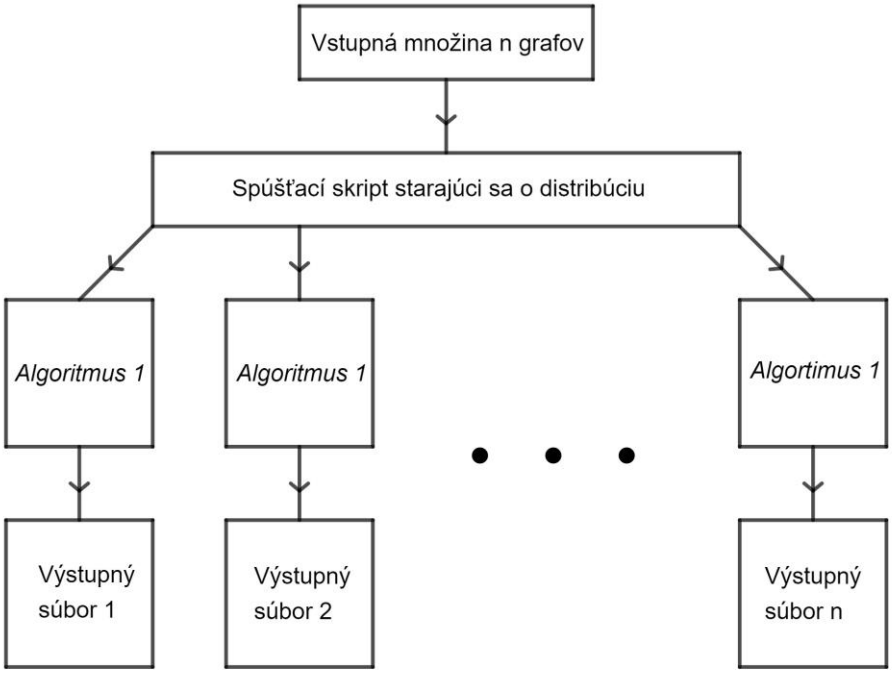
Pracovná hypotéza H2. *Neexistuje také poradie ofarbovania hrán grafu (permutácia), že po jeho aplikovaní na množinu grafov, bude čas ofarbenia celej množiny grafov redukovaný.*

Hypotéza H2 so sebou nesie predpoklad, že množina grafov, ktoré je potrebné hranovo ofarbiť bude **zhodnej veľkosti** – všetky grafy množiny majú rovnaký počet vrcholov a hrán.

Pre overenie pracovnej hypotézy H2 sme navrhli program, ktorý využíva kombináciu **distribuovaného a paralelného počítania**. Vstupom pre program je množina *n* grafov vo formáte graph6 – čo umožňuje použite **jedného vstupného súboru,** obsahujúceho množinu grafov. Z každého zo vstupných grafov paralelne ofarbujeme *p* permutácií. V jednom behu programu vypočítame ofarbiteľnosť pre *n*p* permutovaných grafov.

Program v toku distribuuje úlohy na jednotlivé stroje dostupného výpočtového systému, pričom každá z týchto úloh využíva paralelné vlákna na výpočet ofarbenia permutovaných grafov. Na obrázku 4.7 je náčrt schémy rozdeľovania úloh v programe.

Na paralelné ofarbovanie grafov v rôznom poradí hrán využívame algoritmus prezentovaný v podkapitole 4.1 – v schéme na obrázku 4.7 je samotný algoritmus označený ako **Algoritmus 1**.



Obrázok 4.7 – Schéma rozdelenia úloh v distribuovanom programe pre ofarbovanie permutácií množiny grafov

Výstupom z behu programu je *n* súborov - pre každý vstupný graf jeden súbor obsahujúci:

- **čas ofarbovania** permutácie grafu v milisekundách,
- hodnota **TRUE** ak je permutácia grafu regulárne ofarbiteľná troma farbami, **FALSE** v inom prípade,
- **kód permutácie** grafu odvodený z permutačnej matice susedností grafu a
- **počet rekurzívnych návratov** algoritmu hranového backtrackingu.

Keďže algoritmus hranového backtrackingu pri spätnom prefarbovaní hrán využíva rekurzívne funkcie, do výstupu z našej aplikácie sme pridali meranie počtu rekurzívnych návrtov ofarbovacieho algoritmu. Cieľom tohto merania je zistiť, či existuje spojitosť medzi časom hranového ofarbovania grafu a počtom rekurzívnych návratov ofarbovacieho algoritmu.

4.2.1 METODIKY ZAMERANÉ NA PARALELNÉ OFARBOVANIE PERMUTÁCIÍ MNOŽINY GRAFOV

Na algoritme, ktorý je predmetom tejto podkapitoly, sme navrhli dva typy metodík a vykonali dva typy experimentov [VI, VII]. Ciele jednotlivých navrhnutých metodík sú:

- Skôr, než sa budeme sústrediť na overenie pracovnej hypotézy H2 chceme preskúmať správanie sa úlohy ofarbovania veľkej množiny grafov pri využití výpočtových systémov s **virtualizáciou** na báze virtuálnych strojov a porovnať ho so správaním úlohy na fyzických strojoch. Zovšeobecnene môžeme povedať, že v prvom type experimentov skúmame vplyv virtualizácie na algoritmy obsahujúce **rekurzívne funkcie**.
- V druhom type experimentov hľadáme **špecifickú permutáciu** (resp. permutácie) grafu, ktorá môže byť použitá na **optimalizáciu času ofarbovania** vybranej množiny grafov. To znamená, že hľadáme poradie ofarbovania hrán grafu, ktoré po aplikovaní na celú množinu grafov zredukuje čas ofarbovania tejto množiny.

Navrhli sme osem experimentov – päť experimentov k prvej metodike a tri experimenty k druhej. Skúmali sme celkový čas výpočtu úlohy, minimálny a maximálny počet rekurzívnych volaní a minimálny a maximálny čas ofarbovania jednotlivých permutácií grafu. Tieto vlastnosti sme merali **na dvoch veľkostiach problému**. Ako vstup pre všetky experimenty sme použili množinu 34-vrcholových snarkov s cyklickou súvislosťou viac ako 5 z online databázy House of Graphs [17]. Táto množina grafov obsahuje 19 935 prvkov, pričom v jednotlivých experimentoch používame postupne narastajúce časti tohto datasetu. Prvú veľkosť problému nazývame **malými problémami** a sú v nich obsiahnuté tri experimenty:

- **Experiment 1** pozostáva z ofarbenia množiny 100 grafov v pomere 100 vstupných grafov, pričom z každého vstupného grafu ofarbujeme len jedno náhodné poradie hrán (permutáciu).
- V **experimente 2**, tak isto, ofarbujeme 100 grafov, ale v obrátenom pomere. Na vstupe máme jeden graf, z ktorého vyrobíme 100 náhodných permutácií na ďalšie hranové ofarbovanie.
- **Experiment 3** obsahuje ofarbenie 1000 grafov. Na vstupe programu máme 1000 grafov, z každého v behu aplikácie vyrábame 1 náhodnú permutáciu, ktorú následne ofarbíme.

Z meraní na skupine malých problémov v experimentoch 1, 2 a 3 vyberáme výpočtový systém, ktorý je pre prácu najvhodnejší. V experimentoch porovnávame tri výpočtové systémy, ktorých parametre sú uvedené v tabuľke 4.3.

Tabuľka 4.3 – Výpočtové systémy využité v experimentoch 1, 2 a 3

	Systém s fyzickými strojmi #1 [FS1]	Systém s fyzickými strojmi #2 [FS2]	Systém s virtuálnymi strojmi [VS]
Počet uzlov	5	10	4
Počet procesorov na uzle	16	12	12
RAM	4 GB	4 GB	4 GB

Výpočtové systémy s fyzickými strojmi boli súčasťou vysokovýkonného výpočtového klastra na Univerzite Mateja Bela v Banskej Bystrici prezentovaného v podkapitole 4.1.1. Výpočtový systém s virtualizáciou pozostáva z virtuálnych strojov na procesore Intel Xeon CPU E5-2676 v3 @ 2.4 GHz, ako prístup k tomuto procesoru sme využili služby Amazon Web Services.

Druhou veľkosťou riešeného problému sú **veľké problémy,** na riešenie ktorých využívame architektúry, ktoré sa na experimentoch 1 - 3 ukázali byť najlepšie. Táto časť experimentov pozostáva z:

- V **experimente 4** ofarbujeme 500 náhodných permutácií pre každý zo vstupných 2000 grafov. Dostávame ofarbenie pre množinu 1 000 000 grafov.
- **Experiment 5** dostáva na vstupe množinu 19 935 grafov, pričom z každého z nich hranovo trojofarbí 500 náhodných permutácií. Výsledná množina obsahuje 9 967 500 ofarebení grafov.

Experimenty 6, 7 a 8 sú zamerané na potvrdenie alebo zamietnutie pracovnej hypotézy 2. Navrhnuté experimenty, ktoré pracujú s permutovanými grafmi s cieľom optimalizácie času ofarbovania vybranej množiny grafov sú:

- Ako vstup pre **experiment 6** používame dateset 9 967 500 grafov (19 935 pôvodných snarkov, pričom bol každý permutovaný 500 krát), ktoré boli výstupom z experimentu 5. Keďže množina výstupov z experimentu 5 bola komplexnejšia, ako sme v tomto experimente potrebovali, vybrali sme len údaj o čase ofarbenia pre všetkých 9 967 500 grafov. Z vybraných časov ofarbovania sme pripravili množinu dát s najnižším a množinu dát s najvyšším časom ofarbovania pre každý z pôvodných 19 935 grafov. Datasety sme vizualizovali a analyzovali. Výstupom experimentu 6 je nasledovné rozhodnutie: Generovanie 500 náhodných permutácií pri ofarbovaní množiny

grafov vymeníme za pevne zvolených 500 permutácií s najnižším časom ofarbovania z predošlého merania.

- V **experimente 7** aplikujeme rozhodnutie experimentu 6 a zvolenú množinu permutácií s najnižším časom hranového ofarbovania aplikujeme na všetkých 19 935 grafov zo zvolenej množiny.
- Analogicky k experimentu 7, v **experimente 8** nahradíme generovanie 500 náhodných permutácií ofarbovaním grafov, ktoré permutujeme s použitím permutácie s najlepším časom ofarbovania. Túto permutáciu aplikujeme na vybranú množinu snarkov a meriame čas ofarbovania tejto množiny, ako aj jednotlivých grafov v nej.

4.2.2 EXPERIMENTY ZAMERANÉ NA PARALELNÉ OFARBOVANIE PERMUTÁCIÍ MNOŽINY GRAFOV

Obsahom tejto časti práce sú experimenty založené na metodikách navrhnutých v podkapitole 4.2.1.

V prvej časti podkapitoly prezentujeme experimenty a ich vyhodnotenie s cieľom skúmania vplyvu virtualizácie na algoritmus hranového trojofarbovania množiny grafov. Ako súčasť týchto experimentov skúmame spomalenie paralelného algoritmu, ktorý využíva **rekurzívne funkcie** v systéme s využitím virtualizácie.

Druhá časť podkapitoly obsahuje experimenty a ich vyhodnotenie pre potreby hľadania špecifických permutácií (poradí hrán), pomocou ktorých sme schopní **optimalizovať čas** hranového ofarbovania vybranej množiny grafov.

V experimentoch prezentovaných nižšie sme merali:

- **Celkový čas výpočtu –** čas od spustenia úlohy v systéme do ukončenia úlohy odmeraný systémom PBS [42], meraný v sekundách,
- **CPU čas výpočtu –** suma časov všetkých paralelných častí programu odmeraná pomocou systému PBS, meraný v sekundách,
- **Minimálny a maximálny počet rekurzívnych volaní** pre permutácie každého ofarbovaného grafu zo vstupnej množiny,
- **Minimálny a maximálny čas ofarbovania** pre permutácie každého z ofarbovaných vstupných grafov, meraný v milisekundách.

EXPERIMENT 1

Tabuľka 4.4 – 100 grafov, 1 permutácia

	FS1	FS2	VS
Celkový čas výpočtu [sek]	< 1	< 1	< 1
CPU čas výpočtu [sek]	3	5	1
Minimálny počet rekurzívnych volaní	32	33	1001
Maximálny počet rekurzívnych volaní	3 139 482	3 291 503	28 977
Minimálny čas ofarbovania [ms]	< 1	< 1	< 1
Maximálny čas ofarbovania [ms]	440	530	16

V tabuľke 4.4 sú uvedené výsledky meraní z experimentov, ktoré pozostávajú z ofarbenia 100 grafov. V experimente hranovo ofarbujeme jeden náhodne permutovaný graf pre každý zo vstupných 100 grafov.

EXPERIMENT 2

Experiment 2 pracuje s množinou 100 grafov. Algoritmus dostáva na vstupe jeden graf a vytvára 100 náhodných permutácií pre ofarbovací algoritmus. Merania z tohto experimentu sú uvedené v tabuľke 4.5. Vygenerovali sme 100 rôznych poradí ofarbovania toho istého grafu.

Tabuľka 4.5 – 1 graf, 100 permutácií

	FS1	FS2	VS
Celkový čas výpočtu [sek]	< 1	< 1	< 1
CPU čas výpočtu [sek]	2	3	1
Minimálny počet rekurzívnych volaní	23	32	672
Maximálny počet rekurzívnych volaní	3 328 041	3 211 076	40 619
Minimálny čas ofarbovania [ms]	< 1	< 1	< 1
Maximálny čas ofarbovania [ms]	540	610	5

EXPERIMENT 3

Vstupom pre experiment 3 je 1000 grafov z množiny 19 935 snarkov s 34 vrcholmi. Algoritmus ofarbuje jeden permutovaný graf pre každý zo vstupných grafov. Tabuľka 4.6 obsahuje výsledky namerané na všetkých výpočtových systémoch.

Tabuľka 4.6 – 1000 grafov, 1 permutácia

	FS1	FS2	VS
Celkový čas výpočtu [sek]	< 1	< 1	< 1
CPU čas výpočtu [sek]	15	16	3
Minimálny počet rekurzívnych volaní	5	33	710
Maximálny počet rekurzívnych volaní	2 511 002	3 501 601	37 210
Minimálny čas ofarbovania [ms]	< 1	< 1	< 1
Maximálny čas ofarbovania [ms]	330	480	9

Logickým krokom je overenie ofarbenia 1000 grafov v pomere 1 vstupný graf, z ktorého algoritmus vytvorí 1000 náhodných permutácií. Keďže v systémoch, s ktorými sme pri experimentoch pracovali, nebolo možné pracovať s 1000 paralelnými vláknami, ktoré súčasne ofarbujú grafy, takéto meranie nebolo možné vykonať.

Z experimentov 1, 2 a 3 na malých veľkostiach problému sme vybrali architektúry FS1 a VS. Architektúry boli vybrané na riešenie veľkých problémov (experimenty 4 a 5).

EXPERIMENT 4

Prvým z takzvaných veľkých problémov je hranové ofarbenie množiny 1 000 000 grafov. Pomer vstupných grafov k permutáciám týchto grafov je 2000 ku 500. Ofarbujeme 500 permutácií z každého z 2000 vstupných grafov.

Z výsledkov prezentovaných v tabuľke 4.7 je zrejmé, že čas ofarbovania tejto množiny grafov bol výrazne nižší na výpočtovom systéme, ktorý využíval virtualizáciu, ako na systéme, ktorý využíva čisto fyzické zdroje.

Tabuľka 4.7 – 2000 grafov, 500 permutácií

	FS1	VS
Celkový čas výpočtu [minúty]	34:59	2:07
CPU čas výpočtu [minúty]	44:58	15:10
Minimálny počet rekurzívnych volaní	42	10
Maximálny počet rekurzívnych volaní	9 693 080	1 000 780
Minimálny čas ofarbovania [ms]	< 1	< 1
Maximálny čas ofarbovania [ms]	1280	18

EXPERIMENT 5

V tabuľke 4.8 uvádzame výsledky z merania vykonaného v rámci experimentu 5, v ktorom vyrábame 500 permutácií pre každý graf z vybranej množiny. Týmto spôsobom sme vytvorili množinu 9 967 500 grafov, ktoré hranovo ofarbíme.

Na rozdiel od výsledkov v experimente 4, pozorujeme, že celkový čas výpočtu problému na výpočtových systémoch FS1 a VS je takmer totožný.

Tabuľka 4.8 – 19 935 grafov, 500 permutácií

	FS1	VS
Celkový čas výpočtu [hodiny]	2:41:13	2:52:35
CPU čas výpočtu [hodiny]	4:53:38	3:20:54
Minimálny počet rekurzívnych volaní	31	548
Maximálny počet rekurzívnych volaní	1 283 104	866 762
Minimálny čas ofarbovania [ms]	< 1	< 1
Maximálny čas ofarbovania [ms]	1260	29

Použitie virtualizácie pri výpočtoch spojených s hranovým ofarbovaním grafov pomocou paralelných algoritmov obsahujúcich rekurzívne funkcie **nevytvára spomalenie** alebo oneskorenie výpočtov. V prípade experimentov 3 a 4 bolo z časového hľadiska oveľa výhodnejšie využiť virtuálny výpočtový systém.

Využitie paralelného výpočtového systému v kombinácii s paralelným programovaním pre potreby problému ofarbovania množín grafov je vhodné a dôležité. Na výpočtovom systéme, ktorý obsahoval len fyzické zdroje sme výpočet tohto problému **6,563** násobne skrátili (tabuľka 4.9).

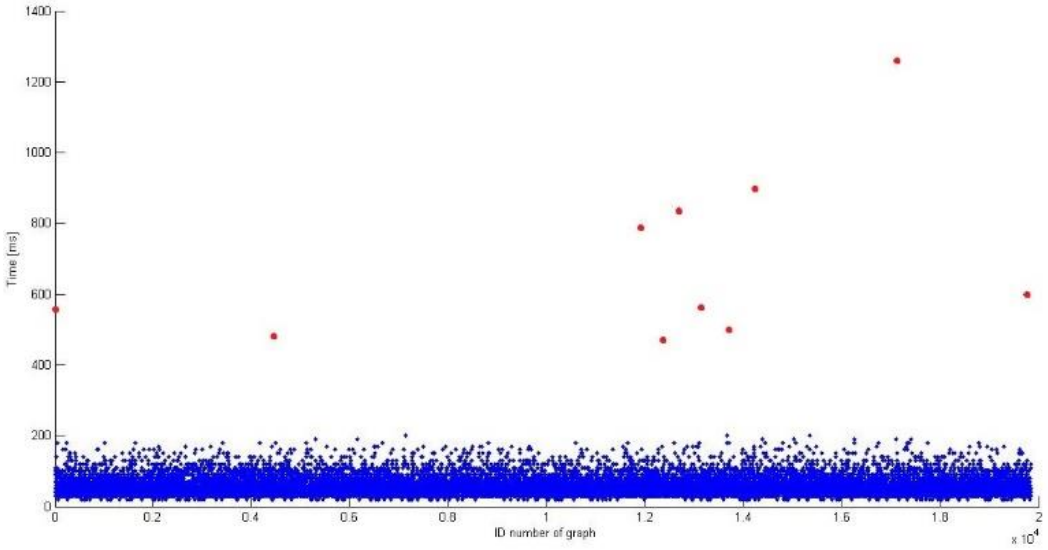
Tabuľka 4.9 – Vyhodnotenie meraní z experimentov 1 - 5

Výpočtový systém	5 uzlov, 16 procesorov na uzle, 4 GB RAM
Veľkosť problému	19 935 snarkov *x* 500 permutácií
Sekvenčný čas ofarbovania [hodiny]	17:36:32
Paralelný čas ofarbovania [hodiny]	2:41:13
Zrýchlenie výpočtu	6,563

EXPERIMENT 6

V experimente 6 uvádzame **najvyšší čas ofarbovania** zo všetkých 500 permutácií každého zo vstupných 19 935 grafov.

Z obrázku vidíme, že väčšina výpočtov ofarbovania trvala **v rozmedzí 20 až 200 milisekúnd**. Červenou farbou sme zvýraznili grafy, ktorých čas hranového ofarbovania bol extrémne dlhý.



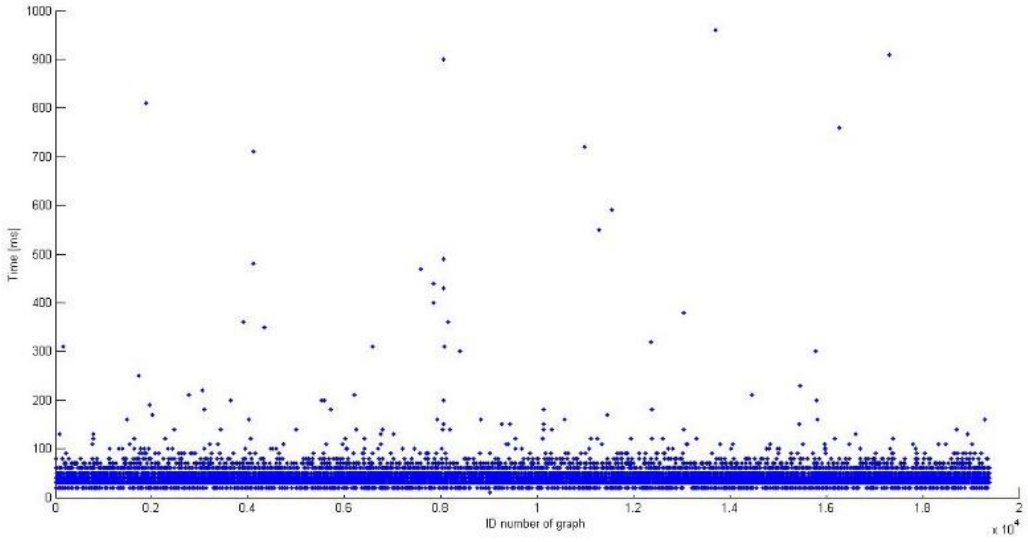
Obrázok 4.8 – Permutácie s maximálnym časom ofarbovania pre pôvodnú množinu 19 935 grafov

Tak isto ako sme vytvorili množinu dát s hodnotami pre najvyššie časy ofarbovania sme vytvorili dataset pre 19 935 **najkratších časov ofarbovania** – pre každý vstupný graf sme vybrali poradie hrán (permutáciu), ktoré bolo ofarbené v najkratšom čase. Všetky prvky tejto dátovej množiny sú permutácie s časom ofarbovania **kratším ako 1 milisekunda**.

EXPERIMENT 7

V experimente 7 sme vymenili generovanie náhodných 500 permutácií za **cielené permutovanie grafov** permutáciami, ktorých čas ofarbovania bol v predchádzajúcom experimente najnižší. Z predošlých meraní v experimente 6 sme vybrali množinu 500 permutácií, ktoré aplikujeme na pôvodnú množinu 19 935 grafov. Celú množinu grafov sme potom hranovo ofarbili.

Maximá časov ofarbovania pre každý graf z množiny sú vizualizované na obrázku 4.9.



Obrázok 4.9 – Maximálny čas ofarbovania pre vybraných 500 permutácií pôvodných 19 935 snarkov

Obrázok 4.9 poukazuje na dva zníženia v maximálnych časoch ofarbovania:

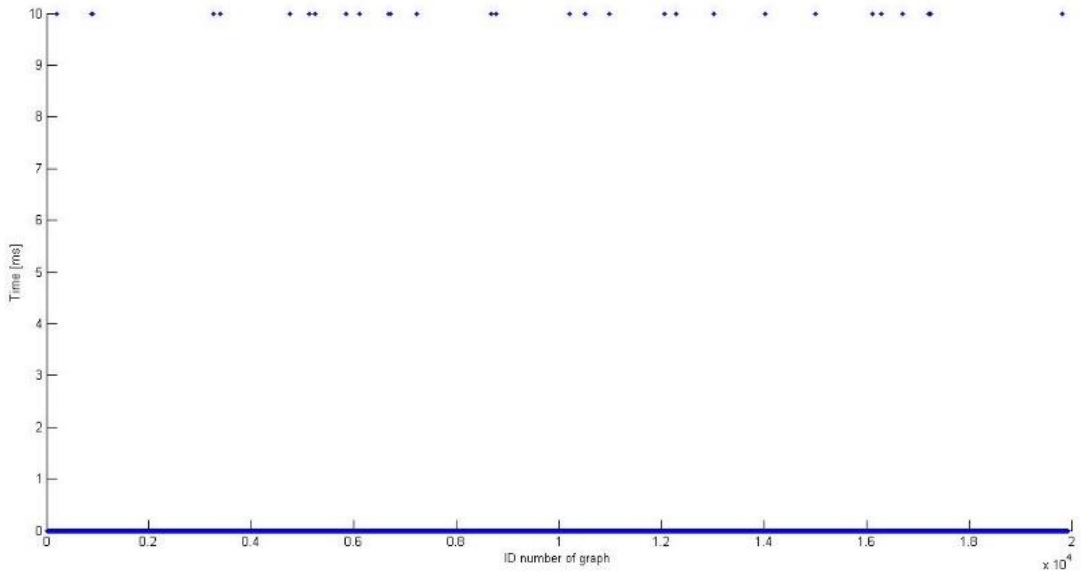
- Hlavná skupina grafov je hranovo ofarbená v čase medzi 20 a 100 milisekundami. V predošlom experimente bol tento časový interval medzi 20 a 200 milisekundami.
- Rozmedzie extrémne vysokých časov ofarbovania bolo znížené z 1260 milisekúnd na 960 milisekúnd.

EXPERIMENT 8

Z množiny 500 permutácií s najnižším časov ofarbovania sme zvolili permutáciu, ktorá má najnižší čas (pozri Prílohu A). Túto permutáciu sme aplikovali na pôvodnú množinu 19 935 grafov a ofarbili sme ich.

Obrázok 4.10, ktorý obsahuje výsledky z tohto experimentu, poukazuje na **pokles času hranového ofarbovania** na:

- hodnoty **pod 0,2 milisekundy** pre hlavnú skupinu grafov,
- a hodnoty v okolí **10 milisekúnd** v extrémnych prípadoch.



Obrázok 4.10 – Maximálny čas ofarbovania pre vybranú najlepšiu permutáciu pôvodných 19 935 snarkov

OPTIMALIZÁCIA ČASU HRANOVÉHO OFARBOVANIA MNOŽINY SNARKOV

V experimentoch 6 a 7 môžeme vidieť rôzne zníženia časov hranového ofarbovania. Algoritmus hranového backtrackingu je na vybranej množine snarkov **citlivý na počiatočnú hranu ofarbovania** (pracovná hypotéza H1).

V experimente 8 sme našli poradie hrán aplikovateľné na grafy z vybranej množiny, pomocou ktorého sme **znížili najvyššie časy hranového ofarbovania grafov** z pôvodného intervalu 20 až 1260 milisekúnd na 0,2 až 10 milisekúnd (tabuľka 4.10). Tento výsledok poukazuje na zamietnutie pracovanej hypotézy H2.

Možnosť ďalšej časovej redukcie hranového ofarbovania grafov je skúmaná v podkapitole 4.3, kde tiež skúmame iné vlastnosti časovej minimalizácie, súvisiace s dekompozíciou problému.

Tabuľka 4.10 – Porovnanie najvyšších časov ofarbovania v experimentoch 6 - 8

	Experiment 6	Experiment 7	Experiment 8
Najvyššie časy ofarbovania pre hlavnú skupinu grafov [ms]	20 – 200	20 – 100	0,2 – 10
Maximálny čas ofarbovania [ms]	1260	960	10

4.3 PARALELNÉ OFARBOVANIE GRAFOV S VYUŽITÍM NAJLEPŠÍCH PERMUTÁCIÍ

Pracovná hypotéza H2 bola v predchádzajúcej časti práce zamietnutá, čo znamená, že vieme nájsť takú permutáciu grafu, ktorej aplikovaním na celú množinu grafov znížime čas ofarbovania celej množiny grafov. Ako bolo ukázané v experimente 8 pôvodná množina grafov bola po aplikovaní zvolenej permutácie rozdelená na dve skupiny grafov, ktoré môžeme špecifikovať ako:

- grafy ofarbiteľné pod 1 milisekundu,
- grafy ofarbiteľné za približne 10 milisekúnd.

Toto pozorovanie viedlo k vytvoreniu pracovnej hypotézy H3.

Pracovná hypotéza H3. *Nie je možné rozdeliť množinu grafov na podmnožiny (zhluky) tak, že pre každú podmnožinu grafov je možné nájsť také poradie ofarbovania hrán, ktoré znižuje čas potrebný na ofarbenie každého grafu v danej podmnožine.*

Pričom sa snažíme hľadať také ofarbenia grafov, ktoré **minimalizujú čas** samotného hranového ofarbovania. Pre naše potreby sme zvolili hranicu 1 milisekundy - požadujeme, aby algoritmus ofarbil každý graf zo zvolenej množiny v čase pod jednu milisekundu.

Táto podmienka je dôležitá nie len z hľadiska redukcie časovej náročnosti hranového ofarbovania množiny grafov, ale aj z pohľadu **dekompozície**. Ako bolo uvedené v podkapitole 1.3, zásadným konceptom pri dekompozícii úlohy na podúlohy je podobnosť (v ideálnom prípade rovnosť) času výpočtu všetkých podúloh. V pracovnej hypotéze H4 sa venujeme práve tejto podstatnej požiadavke na efektívnu dekompozíciu úlohy.

Pracovná hypotéza H4. *Neexistuje také poradie ofarbovania hrán grafu, že čas hranového ofarbovania je pre každý graf z množiny rovnaký (resp. je v stanovenej tolerancii - intervale).*

4.3.1 METODIKA PRE PARALELNÉ OFARBOVANIE GRAFOV S VYUŽITÍM NAJLEPŠÍCH PERMUTÁCIÍ

Pre potreby potvrdenia alebo zamietnutia pracovnej hypotézy H3 sme aplikovali niekoľko poznatkov. Keďže hypotéza hovorí o rozdeľovaní grafov do skupín, ako prvú sme na vybranú množinu grafov použili **konvenčnú zhlukovaciu metódu k-priemerov** (z anglického k-means) podľa [43]. Tento algoritmus zhlukuje *n* prvkov do *k* zhlukov – v našom prípade ide o 19 935 prvkov (grafov) do 14 zhlukov, pričom počet zhlukov bol vybraný na základe takzvanej **metódy siluet** (z angl. Average silhouette method).

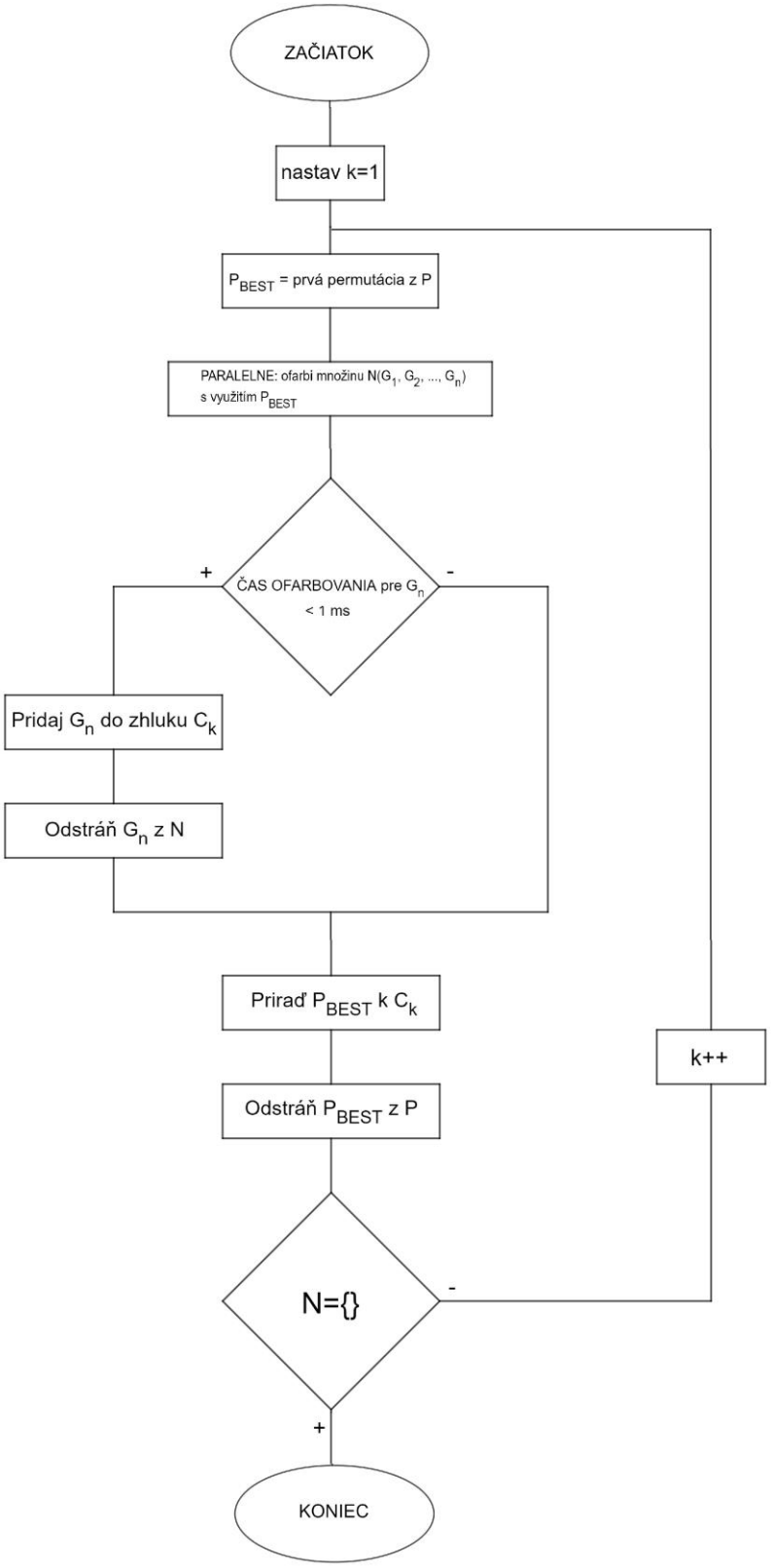
Pri analýze výsledkov tohto zhlukovania sme pozorovali, že grafy, ktorých hranové ofarbovanie trvalo viac ako 1 milisekundu boli umiestnené **na okrajoch zhlukov**. Tento fakt interpretujeme ako nešpecifickosť týchto grafov – môžeme povedať, že grafy, ktorých hranové ofarbovanie trvalo viac ako 1 milisekundu nie sú špecifické pre žiadny z vypočítaných zhlukov. Preto sme navrhli metodiku, ktorej vstupnými dátami sú dve množiny [IX]:

- **Množina permutácií *P***. Z už vypočítaných permutácií grafov vyberieme 500 takých, ktorých aplikácia na grafy viedla k najkratším časom ofarbovania. Vybrané poradia hrán zoradíme vzostupne podľa času ofarbovania.
- **Vstupná množina grafov *N*** zakódovaná vo formáte graph6.

Samotná metodika pracuje nasledovne:

- Z množiny *P* vyberieme **permutáciu s najnižším časom ofarbovania**, označíme ju ako P_{BEST} a aplikujeme ju na vstupnú množinu grafov *N*. Permutovanú množinu grafov následne paralelne hranovo ofarbíme rovnako ako v podkapitolách 4.1 a 4.2.
- Pre každý graf po ofarbení **overíme zvolenú podmienku**. Podmienku sme postavili na základne meraní z experimentu 8: **Bol graf G_n ofarbený v čase pod 1 milisekundu.**
    - **TRUE:** Graf G_n priradíme do zhluku C_k a odstránime ho z množiny *N*.
    - **FALSE:** Algoritmus pokračuje ďalším krokom bez akcie.
- Permutáciu P_{BEST} **priradíme ku zhluku** *Ck* a **odstránime** ju z množiny *P*.
- V prípade, že v množine *N* ešte ostávajú grafy, vytvoríme nový zhluk a postupujeme v algoritme od kroku 1. Ak je množina *N* prázdna, znamená to,

že všetky grafy boli ofarbené v čase nižšom ako jedna milisekunda a algoritmus končí.



Obrázok 4.11 – Schéma algoritmu ofarbovania s využitím najlepších permutácií

Výstupom algoritmu je množina *k* súborov – každý súbor obsahuje zhluk ofarbených grafov s pridelenou permutáciou. Algoritmus je znázornený schémou na obrázku 4.11.

Využitím vytvorenej metodiky predpokladáme potvrdenie alebo zamietnutie **pracovných hypotéz H3 a H4,** definovaných v tejto podkapitole. Navrhnutý algoritmus **nedokončí svoj beh úspešne** v prípade, že nie je možné všetky grafy zo vstupnej množiny hranovo ofarbiť v čase nižšom ako jedna milisekunda. Pod úspešným dokončením behu programu v tomto prípade rozumieme vytvorenie *k* zhlukov, do ktorých sú rozdelené všetky grafy zo vstupnej množiny.

4.3.2 EXPERIMENTY ZAMERANÉ NA PARALELNÉ OFARBOVANIE GRAFOV A NAJLEPŠIE PERMUTÁCIE

Metodiku prezentovanú v podkapitole 4.3.1 sme experimentálne overili na rovnakej množine 19 935 snarkov ako pri predošlom algoritme. Z experimentu 8 predošlej metodiky sme vedeli, že táto množina grafov obsahuje veľkú časť, ktorá bude ofarbená v čase pod jednu milisekundu.

Vstupnú množina permutácií *P* sme vytvorili rovnako ako v experimente 7 predošlého algoritmu.

Obe vstupné množiny *N* a *P* sme zadali algoritmu na spracovanie. Z predošlého experimentu bolo zrejmé, že pri úspešnom behu programu bude výsledkom **2 až 40 zhlukov**. Prvý zhluk bude obsahovať všetky grafy, ktoré boli ofarbené v čase pod 1 milisekundu v experimente 8 – to znamená 19 896 zo vstupných 19 935 grafov. Zvyšných 39 grafov môže byť rozdelených do **1 až 39 zhlukov** podľa času ich hranového ofarbenia.

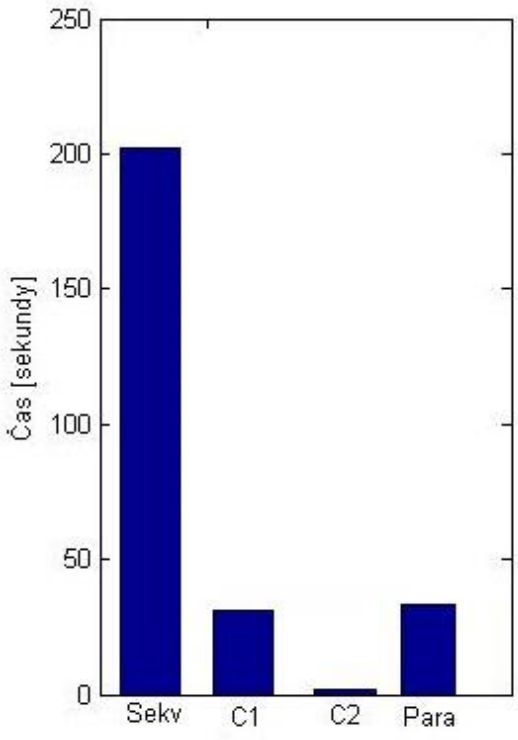
Algoritmus v experimente našiel nasledovné zhluky grafov s priradenými permutáciami:

- Zhluk 1, $P_{BEST} = P_1$
    - 19 896 grafov bolo hranovo ofarbených v čase nižšom ako 1 milisekunda,
    - N obsahuje 39 grafov neofarbených v čase nižšom ako 1 milisekunda.
- Zhluk 2, $P_{BEST} = P_2$
    - 39 grafov bolo hranovo ofarbených v čase nižšom ako 1 milisekunda,
    - N = {} a algoritmus skončil.

Na nájdenie zhlukov s ich priradenými permutáciami (postupnosťami ofarbovania hrán) boli potrebné len 2 cykly algoritmu. Výsledky z meraní spojených s experimentom a použitý výpočtový systém sú uvedené v tabuľke 4.11.

Tabuľka 4.11 – Výsledky meraní z experimentu zameraného na najlepšie permutácie

Výpočtový systém	5 uzlov, 12 procesorov na uzle, 4 GB RAM
Veľkosť problému	19 935 snarkov
Sekvenčný čas ofarbovania [hodiny]	3:22
Paralelný čas ofarbovania [hodiny]	Prvá skupina grafov: 0:31 Druhá skupina grafov: 0:02
Zrýchlenie výpočtu	6,122



Obrázok 4.12 – Porovnanie časov výpočtu hranového ofarbenia 19 935 grafov

Na obrázku 4.12 prezentujeme porovnanie časov výpočtu hranového ofarbenia pre celú množinu 19 935 grafov. Porovnávame čas ofarbovania sekvenčným spôsobom (Sekv), paralelným spôsobom s využitím algoritmu navrhnutého v 4.3.1 (Para) a aj čas potrebný na ofarbenie jednotlivých zhlukov (C1 a C2).

Z experimentu vyplýva, že je možné nájsť množinu permutácií grafov, ktoré redukujú čas výpočtu celej množiny vstupných grafov na požadovanú úroveň. Zároveň je možné nájsť také poradie ofarbovania hrán grafu, že čas ofarbovania všetkých grafov z vybranej množiny je približne podobný – konkrétne pod jednu milisekundu. Tieto zistenia zamietajú hypotézy H3 a H4.

4.4 PARALELNÉ OFARBOVANIE GRAFOV S VYUŽITÍM NAJLEPŠÍCH PERMUTÁCIÍ A DÁVKOVÉHO SPRACOVANIA DÁT

Pre potreby spracovania množín grafov, ktoré obsahujú milióny grafov, sme vytvorili metodiku, ktorá spája princíp využitia vhodnej permutácie grafu a dávkového spracovania dát [IX].

Metodika má na vstupe vybranú množinu grafov. Táto množina je predspracovacím skriptom rozdelená na maximálny možný počet častí obsahujúcich 500 000 grafov a jednu časť s ostatkom grafov. Každá takáto časť je skriptom distribuovaná na paralelné spracovanie v niekoľkých krokoch:

- **Načítanie dát z príslušnej časti do matice *M*.** Grafy sú uložené vo formáte graph 6 čo umožňuje vytvoriť maticu, ktorá bude reprezentovať celú množinu vstupných dát. Počet riadkov tejto matice je rovný počtu grafov v danej časti, počet stĺpcov matice je daný veľkosťou grafu vo formáte graph 6.
- **Paralelné ofarbovanie grafov.** Algoritmus spracováva grafy uložené v matici M v dávkach. Počet dávok je priamo závislí od počtu paralelných vlákien, ktoré sme schopní súčasne spustiť:

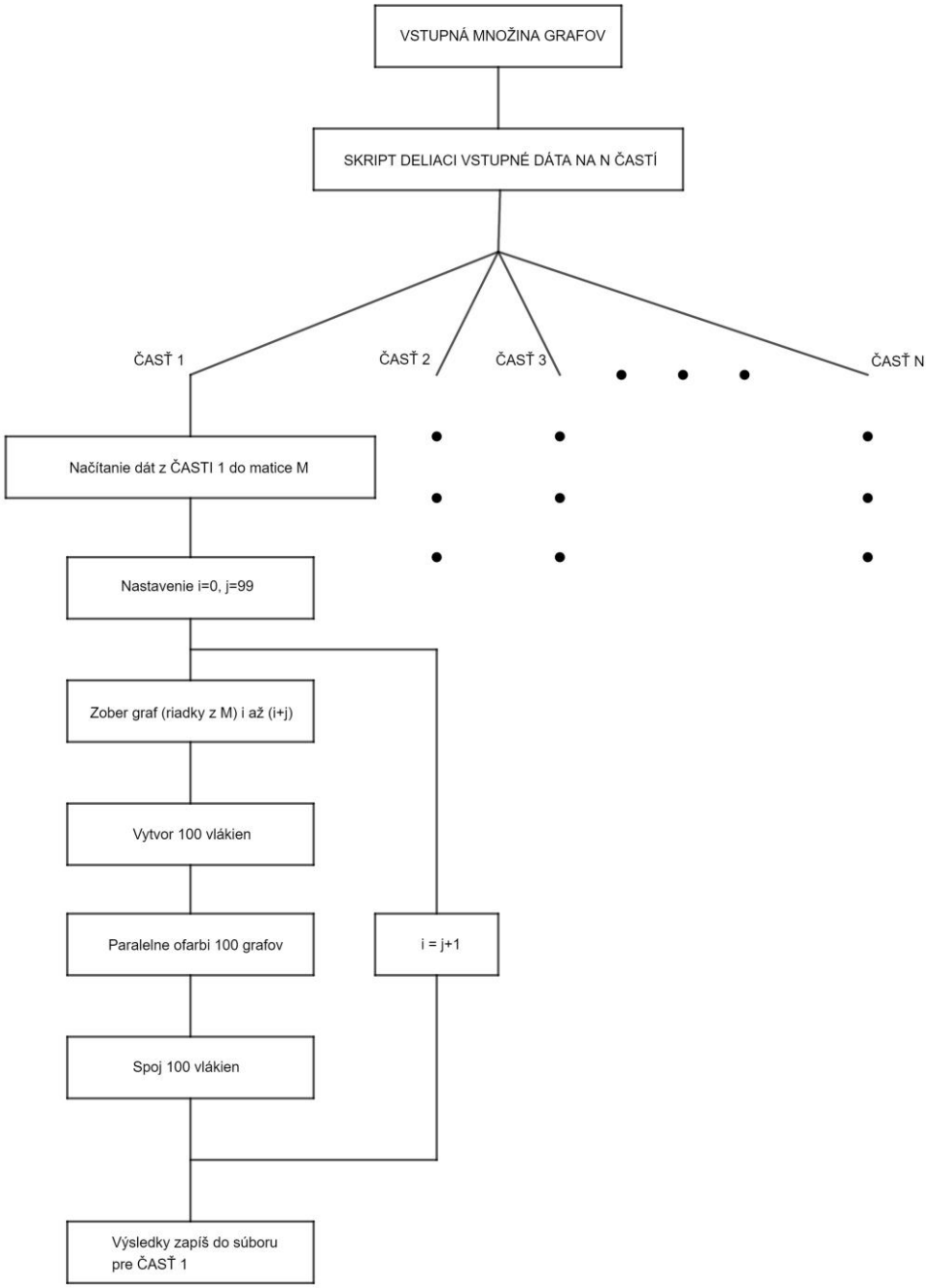
  $$počet\ dávok = počet\ grafov\ v\ matici\ M\ /\ počet\ paralelných\ vlákien \quad (4.2)$$

  Algoritmus vytvorí daný počet paralelných vlákien, z ktorých každé dostane na vstupe jeden graf vo formáte graph6 a následne:
  - graf preloží z formátu graph6 na maticu susedností,
  - graf permutuje s využitím permutácie s najnižším časom ofarbovania nájdenej v 4.2,
  - ofarbí graf.

  Po ofarbení všetkých grafov v dávke sa vlákna spoja, algoritmus zoberie ďalšie grafy z matice *M* a opakuje tento krok.

- Po ofarbení všetkých grafov zo vstupnej časti dát algoritmus zapíše ID grafu a čas potrebný na hranové ofarbenie grafu do súboru, ktorý je spoločný pre všetky grafy v samotnej časti programu.

Výstupom z algoritmu je niekoľko súborov (počet súborov závisí od počtu častí, na ktoré je pôvodná množina dát rozdelená). Schéma algoritmu je uvedená na obrázku 4.13.



Obrázok 4.13 – Schéma paralelného programu zameraného na najlepšie permutácie a dávkové spracovanie dát

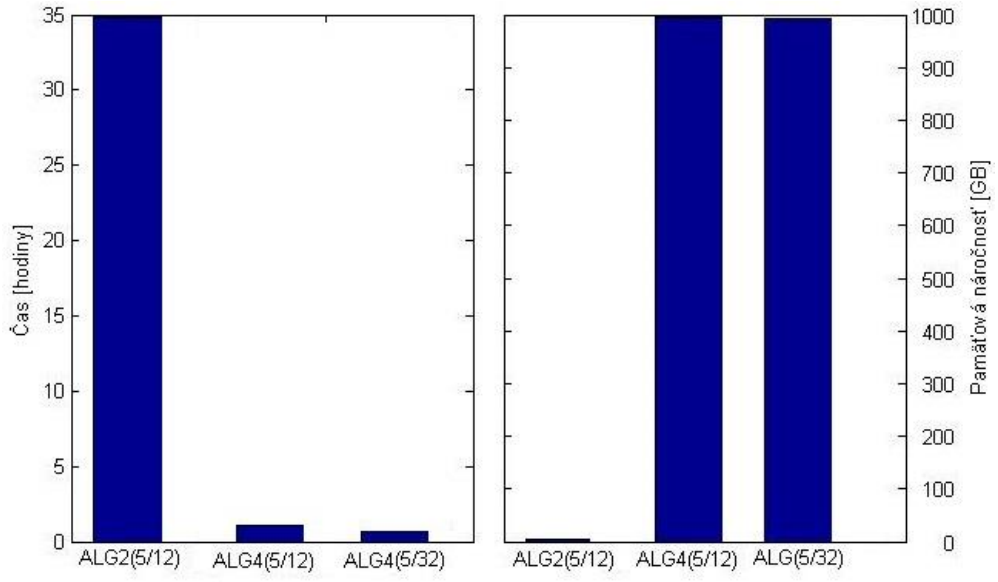
4.4.1 EXPERIMENTY ZAMERANÉ NA PARALELNÉ OFARBOVANIE GRAFOV, NAJLEPŠIE PERMUTÁCIE A DÁVKOVÉ SPRACOVANIE DÁT

Navrhnutý algoritmus sme experimentálne otestovali na množine grafových dát, ktorých časová náročnosť bola pri využití predchádzajúcich algoritmov príliš vysoká na to aby boli algoritmy použiteľné. Táto množina dát obsahuje **3 833 587 kubických grafov** s 34 vrcholmi. Množina bola rozdelená na sedem častí po 500 000 grafov a jednu časť s 333 587 grafmi. Samotné časti boli ofarbované v dávkach po 100 grafov. Výsledky merania a porovnanie časovej a pamäťovej náročnosti s algoritmom 2 prezentovaným v kapitole 4.2 sú uvedené v tabuľke 4.12 a na obrázku 4.14.

Tabuľka 4.12 – Porovnanie časovej a pamäťovej náročnosti ofarbenia množiny 3 833 587 snarkov

	ALG2	ALG4	ALG4
Výpočtový systém	5/12ppn	5/12ppn	5/32ppn*
Čas ofarbovania [hh:mm:ss]	34:47:53	01:06:58	00:42:14
Potrebná pamäť RAM [kb]	3 450 900	994 598 552	993 818 164
Zrýchlenie výpočtu	-	31,178x	49,437x
Zvýšenie pamäťovej náročnosti	-	288,116x	287,988x

***virtualizované**

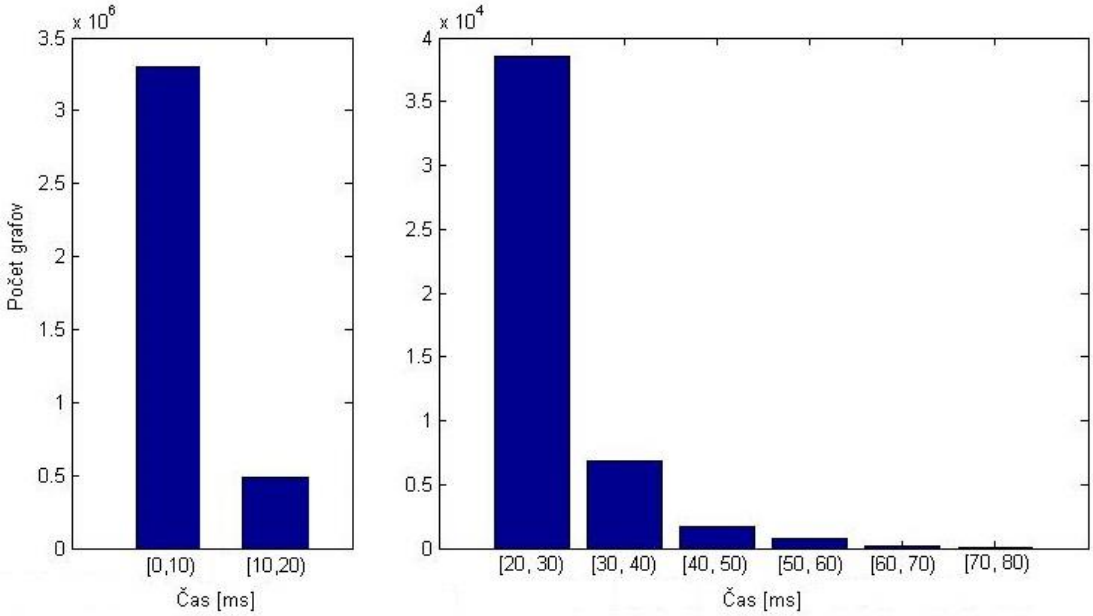


Obrázok 4.14 - Porovnanie časovej a pamäťovej náročnosti ofarbenia množiny 3 833 587 snarkov

Z tabuľky 4.12 a obrázku 4.14 je zrejmé, že pri výpočtoch obsahujúcich milióny grafov dochádza k výraznému zrýchleniu výpočtu, ale zároveň sa dramaticky zvyšuje aj pamäťová náročnosť algoritmu. Pri využití rovnakého výpočtového systému sme dosiahli oproti algoritmu prezentovanému v podkapitole 4.2 približne **31,2 násobné zrýchlenie** výpočtu pričom sa pamäťová náročnosť zvýšila približne **288,1 násobne**.

Bližšia analýza výsledkov ukazuje, že zo vstupnej množiny 3 833 587 grafov bolo:

- 3 301 216 grafov bolo hranovo ofarbených v čase nižšom ako 1 milisekunda,
- 532 364 grafov v čase vyššom ako 1 milisekunda, pričom:
    - 484 267 grafov je v časovom rozmedzí 10 – 20 ms
    - 38 486 grafov v rozmedzí 20 – 30 ms
    - 6 881 grafov je v časovom rozmedzí 30 – 40 ms
    - 1 750 grafov ofarbených v čase 40 – 50 ms
    - 735 grafov je v časovom rozmedzí 50 – 60 ms
    - 175 grafov je v rozmedzí 60 – 70 ms
    - 70 grafov v rozmedzí 70 – 80 ms



Obrázok 4.15 - Porovnanie počtu grafov v jednotlivých nameraných časových intervaloch

Na ďalšie spracovanie dát navrhujeme metodiku odprezentovanú v tejto kapitole spojiť s metodikou prezentovanou v 4.3 a vytvoriť zhluky grafov s priradenými permutáciami.

4.5 PARALELNÉ OFARBOVANIE GRAFOV VYUŽÍVAJÚCE ZHLUKOVANIE GRAFOV A DÁVKOVÉ SPRACOVANIE DÁT

Na základe algoritmov prezentovaných v kapitolách 4.3 a 4.4 sme navrhli **kombináciu** týchto algoritmov [IX], ktorá zabezpečuje zníženie času výpočtu ofarbovania množiny grafov a splní podmienku, ktorá vyplýva z hypotézy 4 (rovnakosť resp. podobnosť času ofarbovania všetkých grafov z vybranej množiny).

4.5.1 METODIKA ZAMERANÁ NA PARALELNÉ OFARBOVANIE GRAFOV, ZHLUKOVANIE GRAFOV A DÁVKOVÉ SPRACOVANIE DÁT

Ako **vstup pre algoritmus** sme zvolili rovnaké množiny ako pri algoritme prezentovanom v 4.3 – množinu grafov N a množinu permutácií P. Algoritmus paralelného ofarbovania grafov využívajúci zhlukovanie grafov a dávkové spracovanie dát je opísaný nasledovne:

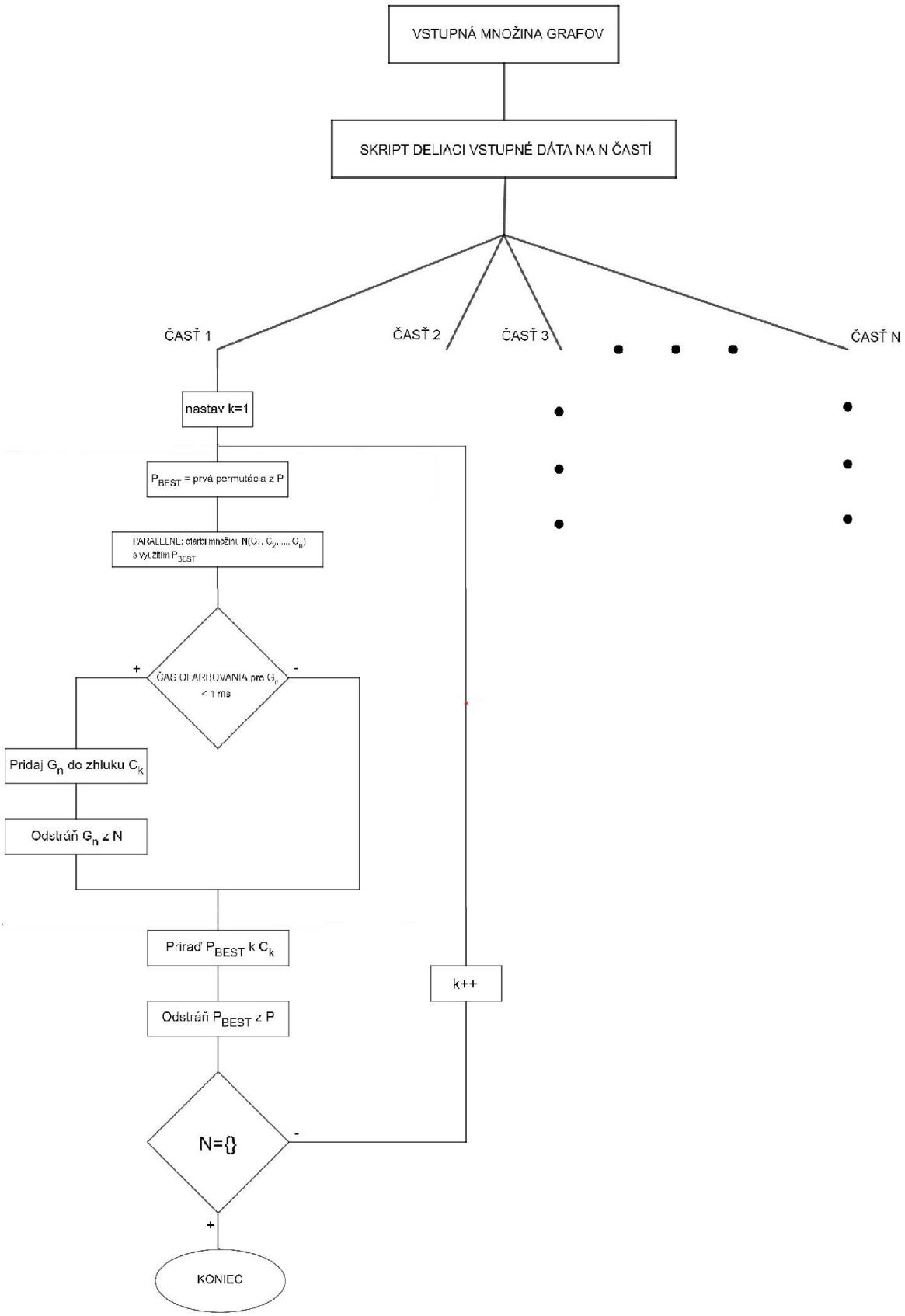
- **Rozdelenie množiny grafov N na dátové časti.** Rovnako, ako v prípade algoritmu zameraného na dávkové spracovanie dát (kapitola 4.4), algoritmus pomocou rozdeľovacieho skriptu rozdelí vstupnú množinu grafov N na n menších častí. Presnejšie, skript rozdelí vstupné dáta na približne rovnako veľké podmnožiny obsahujúce cca. 500 000 prvkov (grafov).
- **Distribuovanie dátových častí.** Po rozdelení množiny grafov na dátové časti postupujeme podľa algoritmu prezentovaného v podkapitole 4.4. Jednotlivé dátové časti (podmnožiny) sú distribuované na dostupný, špecifikovaný výpočtový systém.
- V ďalšom kroku navrhnutý algoritmus vykoná jednotlivé kroky algoritmu pre paralelné ofarbovanie grafov s využitím najlepších permutácií (prezentovaného v podkapitole 4.3). Pre každú z n dátových častí vytvorených zo vstupnej množiny grafov N:
    - **Výber permutácie s najnižším časom ofarbovania** P_{BEST} zo vstupnej množiny permutácií P a jej následné aplikovanie na celú dátovú časť podľa vzťahu 4.1.
    - **Paralelné hranové trojofarbenie grafov** v dátovej časti s využitím výpočtového modelu paralelných vlákien.

- **Overenie podmienky** pre všetky grafy v dátovej časti. Podmienka je zostavená nasledovne: Bol graf *G* hranovo ofarbený v čase nižšom ako jedna milisekunda?
- **Vytvorenie zhluku grafov**, ktoré splnili podmienku uvedené v predchádzajúcom kroku.
- **Pridelenie P_{BEST}** k vytvorenému zhluku grafov.
- **Odstránenie P_{BEST}** z množiny permutácií P, odstránenie grafov s časom ofarbovania nižším ako jedna milisekunda z dátovej časti.

Tieto kroky sú opakované kým daná dátová časť nie je prázdna alebo pokiaľ algoritmus neúspešne nevyskúša všetky permutácie zo vstupnej množiny *P* (takáto situácia môže nastať v prípade, že sa v množine *N* nachádza graf, ktorý nemôže byť ofarbený v čase nižšom ako jedna milisekunda).

Výstup z výpočtu algoritmu sa skladá z množiny súborov, ktorá obsahuje zhluky grafov s ich pridelenými permutáciami pre každú z vytvorených *n* dátových častí. Na získanie celkových výsledkov z výpočtu algoritmu je potrebné na množinu súborov použiť aglomeračnú funkciu. Táto funkcia spája súbory (zhluky), ktorým bola v behu programu pridelená rovnaká permutácia.

Schéma algoritmu, ktorý využíva paralelné ofarbovanie grafov, zhlukovanie grafov a dávkové spracovanie dát je prezentovaná na obrázku 4.16.



Obrázok 4.16 - Schéma paralelného programu zameraného na najlepšie permutácie, dávkové spracovanie dát a zhlukovanie grafov

4.5.2 EXPERIMENTY PRE PARALELNÉ OFARBOVANIE GRAFOV, ZHLUKOVANIE GRAFOV A DÁVKOVÉ SPRACOVANIE DÁT

Cieľom experimentov uvedených v tejto podkapitole je overenie funkčnosti algoritmu navrhovaného v 4.5.1 pričom sa sústredíme na minimalizáciu počtu vytvorených zhlukov. Na úvod sme funkčnosť a vlastnosti algoritmu navrhovaného v 4.5.1 experimentálne overili na dvoch rozmerných množinách dát:

- množina 3 833 587 grafov, ktorá obsahuje 34 vrcholové snarky,
- množina 25 286 953 snarkov s 34 vrcholmi.

Namerané výsledky z týchto experimentov sú uvedené v tabuľkách 4.13 a 4.14. Všetky uvádzané hodnoty boli namerané na výpočtovom systéme pozostávajúcom z 6 uzlov, pričom bolo na každom uzle 12 procesorov. Z predchádzajúcich experimentov (najmä toho, uvedeného v 4.4.1), bolo zrejmé, že algoritmus založený na dávkovom spracovaní dát má vysoké požiadavky na pamäť systému. V prípade 3 833 587 snarkov algoritmus potreboval takmer 1 TB pamäte, v prípade 25 286 953 grafov algoritmus zužitkoval 4 TB pamäte.

V nižšie uvedených tabuľkách uvádzame merania pre nasledovné vlastnosti výpočtov:

- **Číslo zhluku** - ID číslo zhluku, ktorý obsahuje dané grafové dáta,
- **Číslo P_{BEST}** - ID číslo permutácie priradenej k danému zhluku,
- **Čas výpočtu** – čas ofarbovania a režijné náklady pre všetky grafy v danom zhluku,
- **Potrebná pamäť** – pamäť potrebná pri ofarbovaní daného zhluku,
- **Počet grafov ofarbených** v čase vyššom ako jedna milisekunda. Tieto grafy je potrebné ofarbiť s využitím inej permutácie.

Tabuľka 4.13 – Zhluky grafov množiny 3 833 587 snarkov

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	01:06:58	994.598	532 364
2	2	00:07:00	32.059	57 274
3	3	00:00:49	1.457	5 775
4	4	00:00:07	0.254	0

Tabuľka 4.14 – Zhluky grafov množiny 25 286 953 snarkov

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	04:19:09	3219.341	1 356 175
2	2	00:11:43	12.242	127 775
3	3	00:01:26	3.362	15 032
4	4	00:00:16	0.845	6
5	5	00:00:01	0.017	0

Pri bližšom pohľade na dosiahnuté výsledky môžeme uviesť nasledovnú analýzu. Pre množinu **3 833 587 snarkov** sme vytvorili 4 zhluky, pričom:

- Zhluk 1, $P_{BEST} = 1$
  - 3 301 216 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
  - 532 364 grafov bolo ofarbených v čase medzi 10 a 80 ms.
- Zhluk 2, $P_{BEST} = 2$
  - 475 090 zo zostávajúcich 532 364 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
  - 57 274 grafov bolo ofarbených v časoch medzi 10 a 60 milisekúnd.
- Zhluk 3, $P_{BEST} = 3$
  - 51 499 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
  - 5775 grafov bolo ofarbených v časoch v rozmedzí 10 a 30 milisekúnd.
- Zhluk 4, $P_{BEST} = 4$
  - 5775 grafov bolo ofarbených v čase nižšom ako jedna milisekunda. Každý zo vstupnej množiny grafov bol ofarbený v čase nižšom ako jedna milisekunda.

Pre množinu **25 286 953 snarkov** algoritmus vytvoril 5 zhlukov, v ktorých:

- Zhluk 1, $P_{BEST} = 1$
  - 23 930 778 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
  - 1 356 175 grafov bolo ofarbených v časoch v rozmedzí 10 a 100 milisekúnd.

- Zhluk 2, $P_{BEST} = 2$
    - 1 228 400 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
    - 127 775 grafov bolo ofarbených v čase medzi 10 a 30 ms.
- Zhluk 3, $P_{BEST} = 3$
    - 112 743 zo zostávajúcich grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
    - 15 032 grafov bolo ofarbených v časoch 10 až 30 ms.
- Zhluk 4, $P_{BEST} = 4$
    - 15 026 grafov bolo ofarbených v čase nižšom ako jedna milisekunda,
    - 6 grafov bolo ofarbených v čase medzi 10 a 20 milisekúnd.
- Zhluk 5, $P_{BEST} = 5$
    - Zostávajúcich 6 grafov bolo ofarbených v čase nižšom ako jedna milisekunda.

4.5.3 EXPERIMENTY PRE PARALELNÉ OFARBOVANIE GRAFOV, ZHLUKOVANIE GRAFOV A DÁVKOVÉ SPRACOVANIE DÁT NA SADÁCH SNARKOV S VÄČŠÍM POČTOM VRCHOLOV

Pre ďalšie overenie algoritmu a princípu ofarbovania permutovaných grafov (a zároveň overenia hypotéz H1 - H4 na iných množinách snarkov) sme pracovali aj s množinami 36, 38 a 40 vrcholových snarkov (tabuľka 2.1). Pre každú z týchto množín sme postupovali nasledovne:

- **Náhodné permutovanie množín grafov** – pre grafy s 36, 38 a 40 vrcholmi potrebujeme nájsť permutácie, ktorých aplikovanie na množinu grafov znižuje čas ofarbenia grafov v množine. Postupovali sme rovnako v podkapitole 4.2.
- **Vytvorenie množiny najlepších permutácií** – podobne ako v podkapitole 4.2 sme pre množinu 36, 38 a 40 vrcholových grafov náhodným permutovaním našli množinu 500 najlepších permutácií, ktorých aplikovanie na množinu grafov znižovalo čas ofarbenia všetkých grafov v množine.
- **Využitie dávkového spracovania grafov a zhlukovania** – po nájdení vhodného počtu permutácií sme využili algoritmus prezentovaný v 4.5.1 a ofarbili sme všetky podmnožiny 36, 38 a 40 vrcholových snarkov.

V tabuľke 4.15 uvádzame počet grafov v jednotlivých podmnožinách 36, 38 a 40 vrcholových snarkov. Snarky jednotlivých veľkostí sú do týchto podmnožín rozdelené na základe matematických vlastností jednotlivých grafov, tzv. veľkosti najmenšej kružnice v grafe a cyklickej súvislosti. Pri množinách označených znakom '+' zatiaľ nebolo možné vygenerovanie všetkých grafov danej veľkosti (resp. daných vlastností).

Tabuľka 4.15 – Podmnožiny 36, 38 a 40 vrcholových snarkov

Počet vrcholov grafu	Veľkosť najmenšej kružnice ≥ 4	Veľkosť najmenšej kružnice ≥ 5	Veľkosť najmenšej kružnice ≥ 6	Cyklická súvislosť ≥ 5
36	404 899 916	60 167 732	1	180 612
38	-	19 775 763+	39	35 429+
40	-	-	25	-

Začali sme podmnožinou 36 vrcholových grafov s cyklickou súvislosťou vyššou alebo rovnou 5. Táto množina obsahuje 180 612 grafov a pomocou nášho algoritmu sme ju rozdelili do štyroch zhlukov (tabuľka 4.16).

Tabuľka 4.16 – Zhluky grafov množiny 180 612 snarkov s 36 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	00:01:10	1.326	4008
2	2	00:00:07	0.839	241
3	3	00:00:07	0.376	12
4	4	00:00:01	0.017	0

Pri 36 vrcholových snarkoch existuje množina grafov s veľkosťou najmenšej kružnice v grafe rovnou alebo vyššou ako 6, ktorá obsahuje jeden graf. Tento graf bol ofarbený v čase nižšom ako jedna milisekunda pomocou permutácie $P_{BEST} = 2$. Pokračovali sme množinou 60 167 732 snarkov s 36 vrcholmi. Túto množinu náš algoritmus rozdelil na 7 zhlukov (tabuľka 4.17).

Tabuľka 4.17 – Zhluky grafov množiny 60 167 732 snarkov s 36 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	08:21:18	4372.881	4 229 360
2	2	00:34:31	35.237	256 902
3	3	00:03:11	1.245	53 487
4	4	00:01:07	0.891	16 343
5	5	00:00:33	0.873	8 901
6	6	00:00:31	0.754	8 042
7	7	00:00:04	0.019	0

Najrozsiahlejšia množina 36 vrcholových snarkov obsahuje 404 899 916 prvkov. Túto množinu náš algoritmus spravoval do 16 zhlukov (tabuľka 4.18).

Tabuľka 4.18 – Zhluky grafov množiny 404 899 916 snarkov s 36 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	47:16:14	8472.230	17 642 937
2	2	03:32:49	56.671	12 876 022
3	3	00:56:11	30.290	4 038 765
4	4	00:40:26	12.128	1 844 397
5	5	00:38:39	9.133	1 000 036
6	6	00:07:12	2.465	321 009
7	7	00:05:31	1.026	162 932
8	8	00:00:22	0.891	21 329
9	9	00:00:20	0.806	17 890
10	10	00:00:18	0.650	12 993
11	11	00:00:17	0.521	9 636
12	12	00:00:13	0.360	4109
13	13	00:00:10	0.332	1572
14	14	00:00:03	0.196	60
15	15	00:00:02	0.183	38
16	16	00:00:03	0.067	0

Pokračovali sme množinami snarkov, ktoré sa skladajú z 38 vrcholov. K dispozícií sú 3 takéto množiny – množina s 35 429 grafmi, množina s 39 grafmi a množina s 19 775 768 prvkami. Výsledky rozdelenia do zhlukov pre tieto množiny grafov sú uvedené v tabuľkách 4.19, 4.20 a 4.21.

Tabuľka 4.19 – Zhluky grafov množiny 35 429 snarkov s 38 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	00:00:42	0.630	1053
2	2	00:00:11	0.439	0

Tabuľka 4.20 – Zhluky grafov množiny 39 snarkov s 38 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	00:00:03	0.371	0

Tabuľka 4.21 – Zhluky grafov množiny 19 775 768 snarkov s 38 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	03:16:02	3018.03	594 945
2	2	00:01:33	1.139	10 709
3	3	00:00:03	0.732	4 283
4	4	00:00:01	0.424	0

Poslednou množinou snarkov sú 40 vrcholové grafy, ktorých je 25. Aj napriek nízkemu počtu týchto grafov, bola táto množina rozdelená do troch zhlukov uvedených v tabuľka 4.22.

Tabuľka 4.22 – Zhluky grafov množiny 25 snarkov so 40 vrcholmi

Číslo zhluku	Číslo P_{BEST}	Čas výpočtu [hh:mm:ss]	Potrebná pamäť [GB]	Počet grafov ofarbených > 1 ms
1	1	00:00:04	0.230	9
2	2	00:00:01	0.089	2
3	3	00:00:01	0.019	0

4.5.4 VYHODNOTENIE EXPERIMENTOV PRE PARALELNÉ OFARBOVANIE GRAFOV, ZHLUKOVANIE GRAFOV A DÁVKOVÉ SPRACOVANIE DÁT

Celkový čas výpočtu potrebný na ofarbenie celej množiny grafov pričom prihliadame na podmienku plynúcu z hypotéz H3 a H4 môžeme vypočítať nasledovne:

$$T_o = \sum_{i=1}^{k} Tc_i \tag{4.3}$$

kde *To* je celkový čas výpočtu ofarbovania danej množiny grafov, *k* je počet zhlukov a Tc_i je výpočtový čas pre zhluk číslo *i*.

Pre **34 vrcholové snarky** sme namerali nasledovné výsledky. Celkový čas ofarbovania pre množinu 3 833 587 bol 01:14:54. Pre množinu 25 286 953 snarkov bol tento čas rovný 04:32:35.

36 vrcholové snarky obsahujú ako jediné zo skúmaných množín grafov štyri podmnožiny. Prvá podmnožina obsahuje 180 612 a celkový čas výpočtu jej ofarbenia bol 00:01:25. Podmnožina grafov s veľkosťou najmenšej kružnice ≥ 6 obsahuje len jeden graf, ktorý bol ofarbený v čase nižšom ako jedna sekunda. Ďalšie dve podmnožiny 36 vrcholových snarkov obsahujú viac než 60 miliónov a viac než 404 miliónov grafov. Celkový čas výpočtu ofarbenia pre množinu 60 167 732 bol 09:01:15. Pre množinu 404 899 916 snarkov bol čas výpočtu ofarbenia celej množiny grafov 53:18:50.

Pri **38 vrcholových snarkoch** sme skúmali 3 skupiny grafov. Prvá z nich obsahovala 35 429 prvkov a prezentovaný algoritmus ju ofarbil v čase 53 sekúnd. Druhou množinou 38 vrcholových snarkov bola malá množina 39 grafov, ktoré boli ofarbené za 3 sekundy. Poslednú podmnožinu 38 vrcholových snarkov, ktorá obsahuje 19 775 768 grafov, algoritmus ofarbil za 03:17:39.

V prípade **40 vrcholových snarkov** poznáme len jednu množinu 25 grafov, ktorá však bola prekvapivo rozdelená do troch zhlukov a ofarbená za 6 sekúnd.

Aj keď algoritmus pre svoj beh vyžaduje vysokú pamäťovú náročnosť, umožňuje paralelné zrýchlenie výpočtu a vytvára zhluky, ktoré majú pridelené permutácie, pomocou ktorých vieme graf ofarbiť uspokojivo z pohľadu hypotézy H4. Ide teda o **efektívnu dekompozíciu problému**.

V tabuľkách 4.13 – 4.22 môžeme vidieť, že nie je potrebné vysoké množstvo vhodne zvolených permutácií na to aby sme dosiahli čas ofarbovania každého grafu z ofarbovanej množiny nižší ako jedna milisekunda. Konkrétne počty permutácií (resp. klastrov) sú uvedené v tabuľke 4.23.

Tabuľka 4.23 – Počet potrebných zhlukov (permutácií) pre všetky testované množiny snarkov

34 vrcholové snarky				
Počet grafov	19 935	-	3 833 587	25 286 953
Počet zhlukov	2	-	4	5
36 vrcholové snarky				
Počet grafov	180 612	1	60 167 732	404 899 916
Počet zhlukov	4	1	7	16
38 vrcholové snarky				
Počet grafov	35 429	39	19 775 763+	-
Počet zhlukov	2	1	4	-
40 vrcholové snarky				
Počet grafov	-	25	-	-
Počet zhlukov	-	3	-	-

Tabuľka 4.23 reprezentuje sumarizáciu zaujímavých údajov, ktoré sa týkajú zhlukovania snarkov.

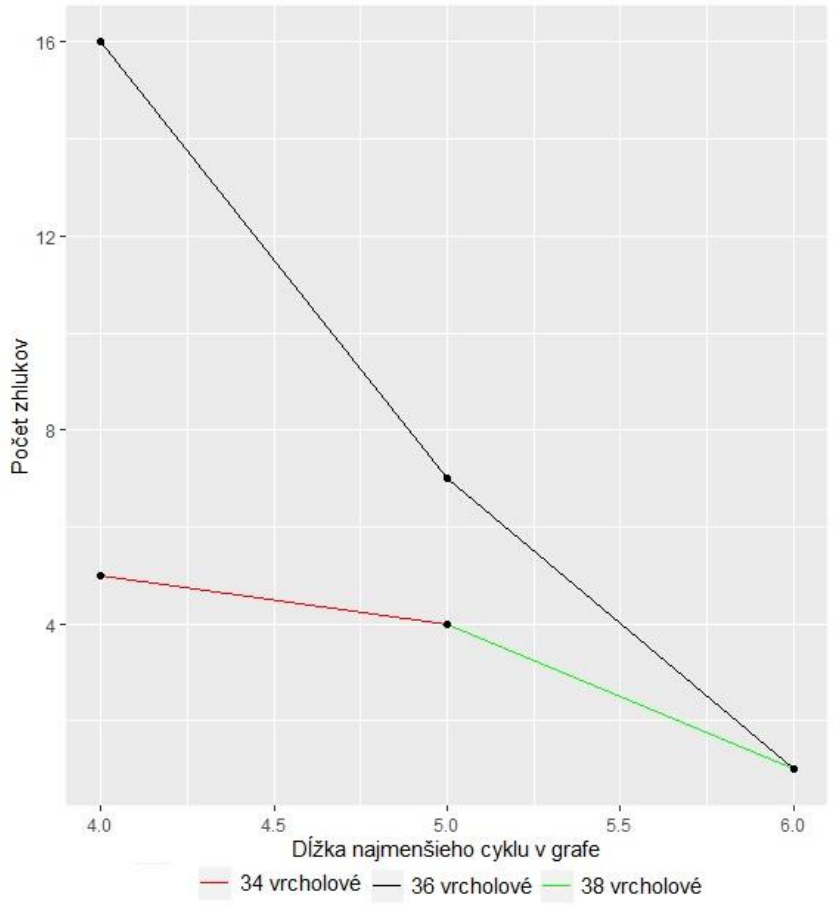
Keďže konvenčné metódy zhlukovania (metóda k-priemerov, k-najbližších susedov a pod.) potrebujú na svojom vstupe množinu dát a počet zhlukov, do ktorých majú byť dané dáta rozdelené, je potrebné aby sme vedeli potrený počet zhlukov aspoň odhadnúť.

4.5.5 POČET POTREBNÝCH ZHLUKOV PRE SNARKY S 34 – 38 VRCHOLMI

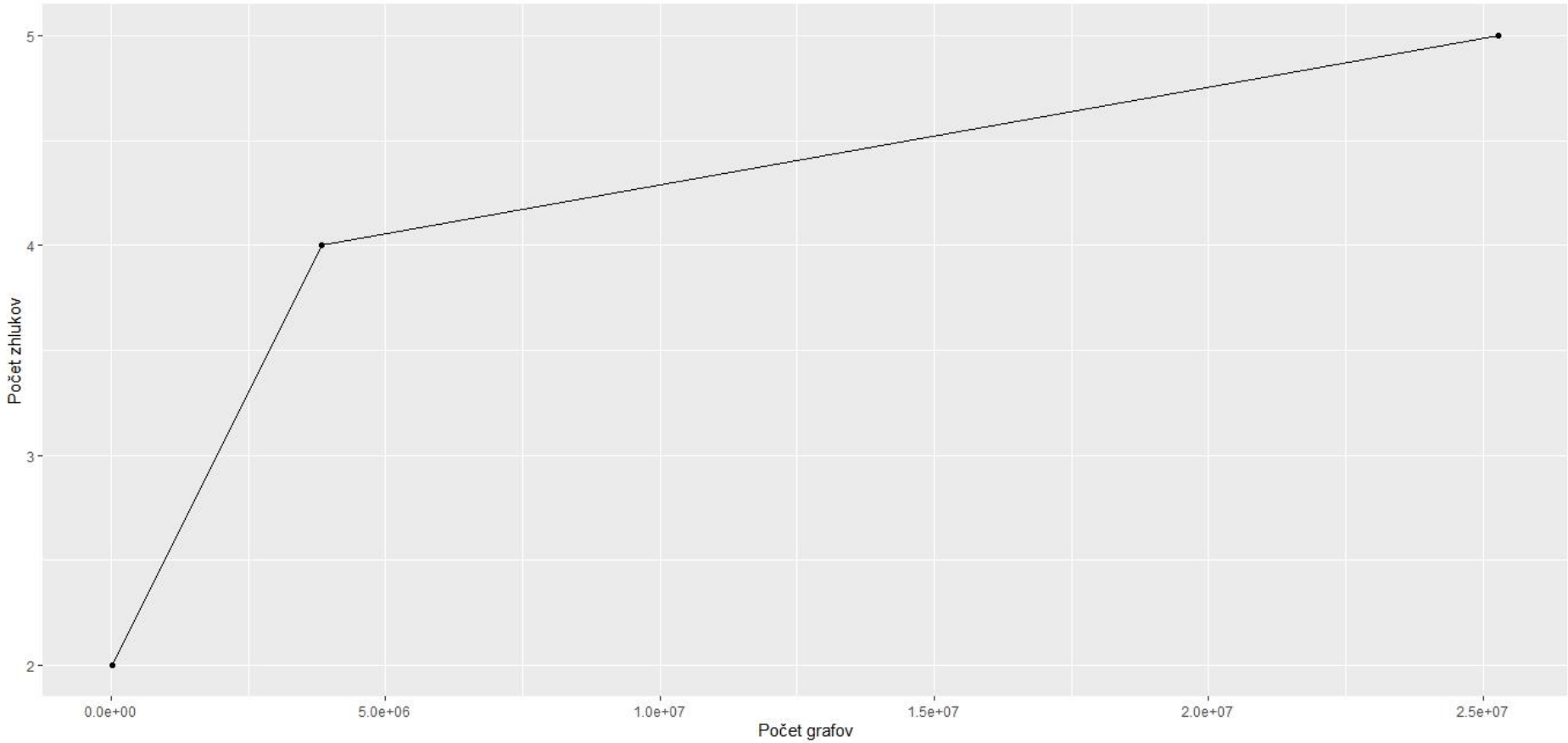
Aj keď je táto časť práce nad rámec stanovených cieľov, dosiahnuté výsledky nás nabádajú ku zovšeobecneniu počtu potrebných zhlukov pre grafové dáta s využitím **interpolačných funkcií**. Preto sme vytvorili skript v jazyku R, pomocou ktorého vieme znázorniť potenciálny počet potrebných permutácií pre rôzne veľké sady dát.

Prvá metóda odhadu počtu potrebných zhlukov sme založili na interpolácii, ktorá využívala počet potrebných zhlukov pre jednotlivé podmnožiny snarkov a matematické vlastnosti, pomocou ktorých boli snarky do týchto množín roztriedené (tabuľka 4.15). Interpolácia na základe počtu potrebných zhlukov a dĺžky najkratšieho cyklu v grafe je uvedená na obrázku 4.17.

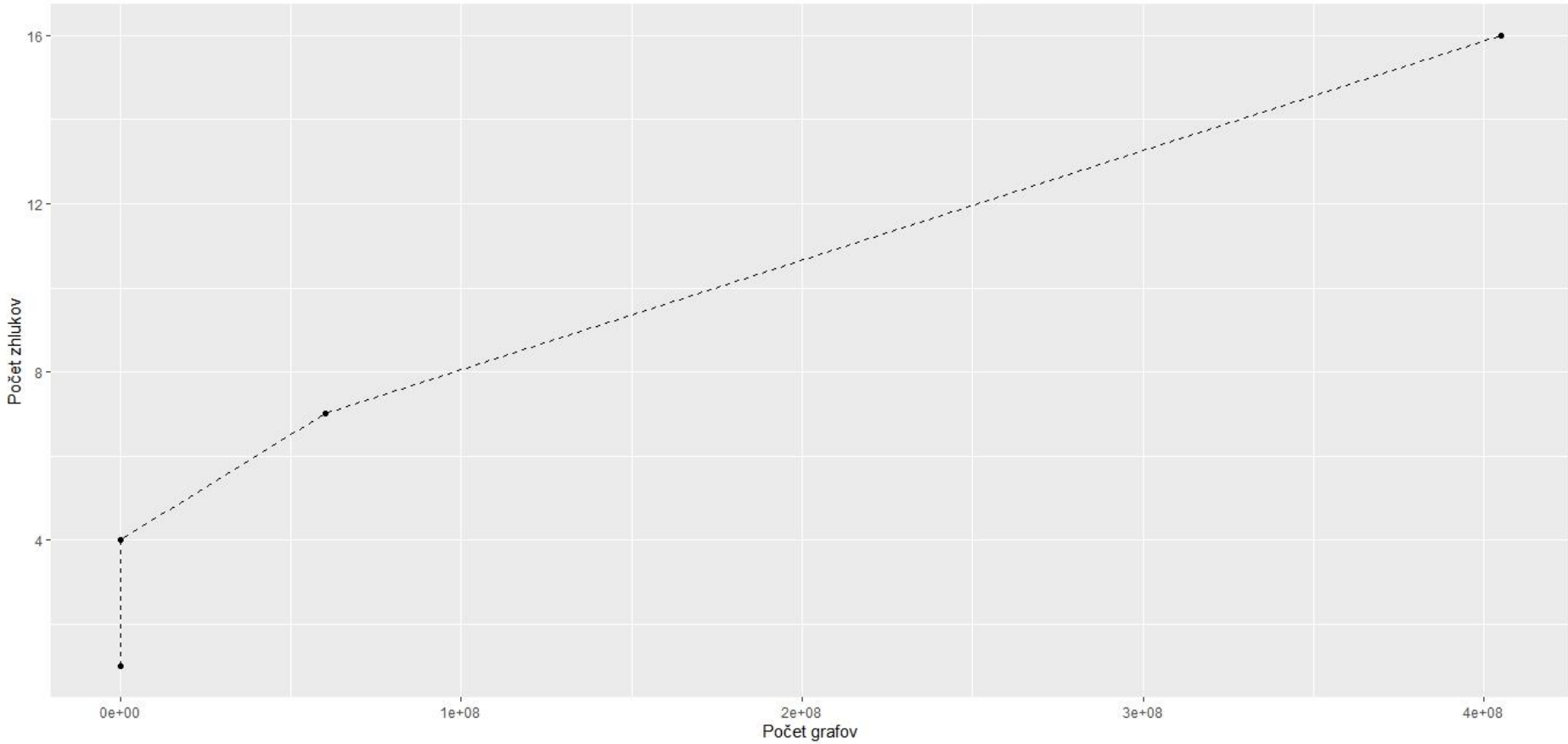
Ako druhé sme vyhotovili interpolácie počtu zhlukov na základne porovnania počtu grafov v jednotlivých podmnožinách a odmeraného počtu potrebných zhlukov pre tieto podmnožiny (tabuľka 4.23). Na obrázkoch 4.18, 4.19 a 4.20 prezentujeme interpolácie pre 34, 36 a 38 vrcholové snarky. Porovnanie kriviek týchto grafov uvádzame na grafe zobrazenom na obrázku 4.21.



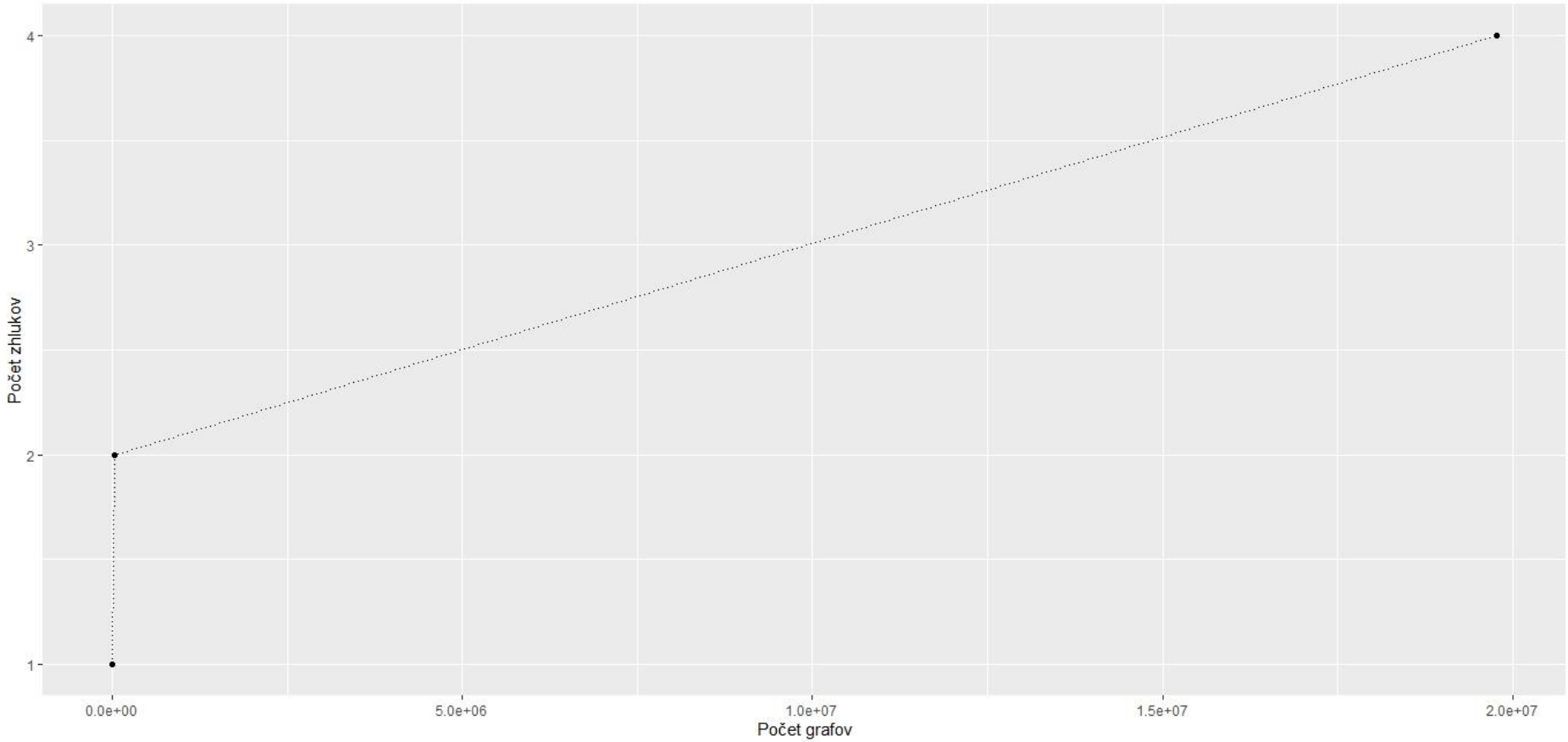
Obrázok 4.17 – Interpolácia počtu zhlukov pre 34, 36 a 38 vrcholové snarky založená na veľkosti najmenšej kružnice v grafe



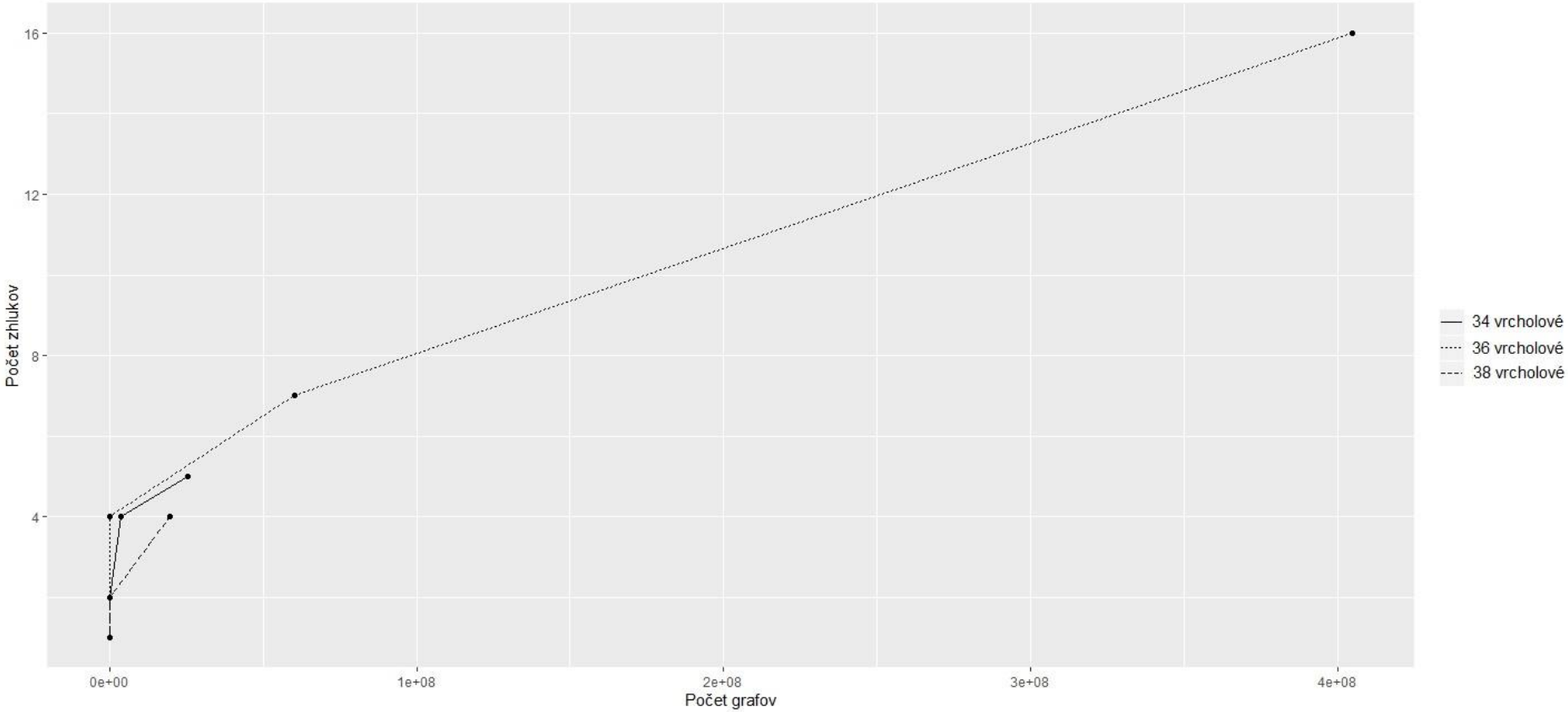
Obrázok 4.18 – Interpolácia počtu zhlukov pre 34 vrcholové snarky



Obrázok 4.19 – Interpolácia počtu zhlukov pre 36 vrcholové snarky



Obrázok 4.20 – Interpolácia počtu zhlukov pre 38 vrcholové snarky



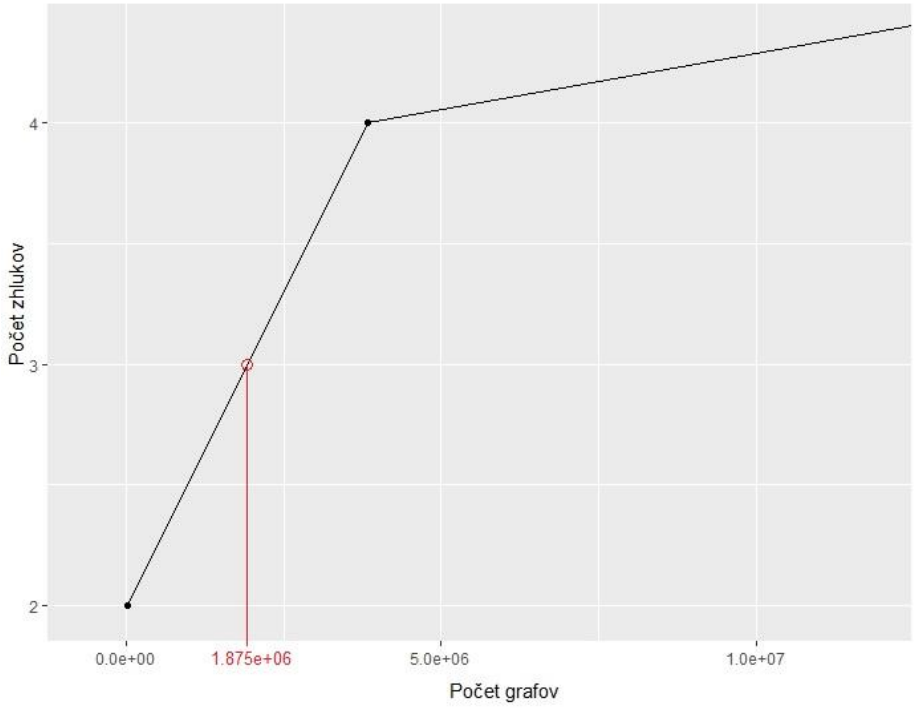
Obrázok 4.21 – Porovnanie interpolácií počtu potrebných zhlukov pre 34, 36 a 38 vrcholové snarky

Prvý model interpolácie založený na nameranom počte zhlukov a dĺžke najmenšieho cyklu v grafe nepriniesol zaujímavé výsledky. Dôvodom je najmä nekompletnosť údajov, pri ktorých nebola dĺžka najmenšieho cyklu v grafe nameraná (ide o jednu hodnotu pre každú z vybraných množín snarkov).

Interpolácia, ktorá využíva počet grafov a nameraný počet zhlukov poukazuje na tieto poznatky:

- Podľa tohto typu interpolácie **nie je možné určiť predpis funkcie**, pomocou ktorej by sme vedeli vypočítať počet potrebných zhlukov pre ľubovoľný počet grafov.
- Na základne interpolácií uvedených na obrázkoch 4.18 – 4.21 vieme **odhadnúť počet potrebných zhlukov** pre sady snarkov, ktoré obsahujú:
    - 19 935 – 25 286 953 grafov s 34 vrcholmi,
    - 1 – 404 899 916 grafov s 36 vrcholmi,
    - 39 – 19 775 763 grafov s 38 vrcholmi.

Ako príklad uvádzame obrázok 4.22, na ktorom je priesečníkom dvoch priamok znázornený počet potrebných zhlukov (3) pre približne 1 875 000 snarkov s 34 vrcholmi.



Obrázok 4.22 – Počet zhlukov pre 1 875 000 snarkov s 34 vrcholmi

ZÁVER

Hranové ofarbovanie grafov je operácia, ktorá je často opakovanou časťou výpočtu pri skúmaní rôznych vlastností grafov, ako napríklad hľadanie hranovej a vrcholovej kritickosti alebo kokritickosti grafu. Hlavným cieľom práce bol návrh, implementácia a overenie algoritmov, metodík a nástrojov, ktoré prispejú k optimalizácii dekompozície paralelných aplikácií pri práci s hranovým ofarbovaním kubických grafov. Konkrétnymi cieľmi práce bolo potvrdenie alebo zamietnutie nasledujúcich hypotéz:

- *Algoritmus hranového backtrackingu nie je citlivý na počiatočný vrchol ofarbovania grafu. To znamená, že ak pri výpočte použijeme rôznu postupnosť hrán bude doba výpočtu algoritmu pre zadaný graf rovnaká.*
    - Vo všetkých experimentoch uvedených v kapitole 4 sme ukázali, že doba výpočtu algoritmu hranového backtrackingu je pre rôzne permutácie **výrazne odlišná**, čím sme hypotézu zamietli. Namerané hodnoty sa pohybovali v intervale od časov pod 1 milisekundu až po časy približne 1300 ms.
- *Neexistuje také poradie ofarbovania hrán grafu (permutácia), že po jeho aplikovaní na množinu grafov, bude čas ofarbenia celej množiny grafov redukovaný.*
    - Našli sme permutáciu aplikovateľnú na grafy z vybranej množiny, pomocou ktorej sme **znížili časy hranového ofarbovania grafov.** Tento výsledok bol experimentálne overený v podkapitole 4.2.2 a poukazuje na zamietnutie pracovanej hypotézy H2.
- *Nie je možné rozdeliť množinu grafov na podmnožiny tak, že pre každú podmnožinu grafov sme schopní nájsť jej vlastné poradie ofarbovania hrán, ktoré znižuje čas potrebný na ofarbenie každého grafu v danej podmnožine.*
    - Navrhli, implementovali a experimentálne sme overili algoritmus, pomocou ktorého sme našli množinu permutácií grafov, ktoré **redukujú čas výpočtu** celej množiny vstupných grafov na požadovanú úroveň. Experimenty zhlukovania grafov sú uvedené v podkapitole 4.3.2.

- *Neexistuje také poradie ofarbovania hrán grafu, že čas hranového ofarbovania je pre každý graf z množiny rovnaký.*
    - Je možné nájsť také poradie ofarbovania hrán grafu, že čas ofarbovania všetkých grafov z vybranej množiny je približne podobný – v našom prípade sme v experimentoch uvedených v kapitole 4.3.2 **použili množinu permutácií**, pomocou ktorých sme ofarbili jednotlivé grafy z vybranej množiny **pod jednu milisekundu.** Čím sme optimalizovali dekompozíciu paralelných a distribuovaných výpočtov čo bol hlavný cieľ tejto práce.

Okrem cieľov práce formalizovaných v hypotézach sme preskúmali možnosti **paralelizácie výpočtov** týkajúcich sa hranového ofarbovania grafov. V rámci práce sme optimalizovali (minimalizovali) čas ofarbovania množín vybraných grafov s využitím vhodne zvolených permutácií grafov. **Dekomponovali** sme problém ofarbenia rozmernej množiny grafov na menšie podproblémy, ktoré boli distribuované na výpočtovú architektúru a počítané s využitím paralelných algoritmov. Pomocou algoritmu prezentovaného v podkapitole 4.5 sme vypočítali hranové trojofarbenie všetkých veľkých množín snarkov z portálu House of Graphs tak, aby sme zabezpečili podobný (resp. rovnaký) čas ofarbovania pre každý graf z vybranej množiny – čas nižší ako jedna milisekunda. Táto podobnosť (resp. rovnosť) času je kľúčovým konceptom efektívnej dekompozície akéhokoľvek problému na menšie podproblémy. Výpočty tak môžu byť vykonávané na akomkoľvek výpočtovom systéme a trvanie ofarbenia jednotlivých grafov bude vždy podobné (resp. rovnaké). Tým sme ukázali, že využitie paralelných a distribuovaných výpočtov je možné do istej **úrovne granularity danej úlohy**.

Využitie paralelného výpočtového systému v kombinácii s paralelným programovaním pre potreby problému ofarbovania množín grafov je vhodné a dôležité. Na výpočtovom systéme, ktorý obsahoval len fyzické zdroje sme výpočet tohto problému **niekoľkonásobne zrýchlili**.

Na záver výskumnej časti práce sme uviedli koncept **zhlukovania grafov** na základe permutácií. Štandardné metódy zhlukovania potrebujú pre svoj beh poznať počet zhlukov, do ktorých majú byť dáta rozdelené. V prípade grafových dát, s ktorými sme v práci pracovali je počet zhlukov ťažko odhadnuteľný. Využili sme

preto interpolačné funkcie na odhadnutie vhodného počtu potrebných zhlukov a tým sme otvorili nové potenciálne oblasti skúmania vo zvolenej problematike.

Algoritmy prezentované v [13, 14] sú zamerané na optimalizáciu času hranového ofarbovania jedného grafu. Aj keď je tento cieľ podobný ako cieľ našej práce, autori nevyužívajú žiadny z nami použitých konceptov (dekompozícia, paralelné počítanie alebo permutovanie grafov). V prácach, ktoré sú s týmito algoritmami spojené, tiež nie sú počítané ofarbenia veľkých množín grafov.

Paralelné prístupy k problematike ofarbovania grafov sú stručne opísané v [45] a optimalizované v [46]. Autori publikácií však nedbajú na dekompozíciu problému, permutovanie grafov a ofarbovanie veľkých množín grafov.

Vzhľadom na odlišnosti týchto prístupov nie je možné porovnať výsledky autorov s výskumom prezentovanom v tejto práci.

ĎALŠÍ VÝSKUM

V rámci ďalšieho výskumu, spojeného s prezentovanými výsledkami, je možné sústrediť sa na preskúmanie niekoľkých oblastí:

- **Hľadanie vhodných zhlukovacích metód.** Využitie zhlukovacej metódy k-priemerov poukázalo na zaujímavé pozorovania. Metóda však neposkytla vhodné riešenie zhlukovania pre grafové dáta. V ďalšom výskume je potrebné preskúmať viac konvenčných zhlukovacích metód a analyzovať ich použiteľnosť pri zhlukovaní grafových údajov. Tiež je potrebné sústrediť sa na presný odhad potrebných zhlukov pre rôzne množiny grafov – úvod do tejto problematiky bol prezentovaný v podkapitole 4.5.
- **Využitie výsledkov získaných v práci.** Grafové ofarbovanie je využiteľné v niekoľkých oblastiach výskumu – rozvrhovanie, prideľovanie rádiových frekvencií, optimalizácia kompilátorov alebo pri SAT solveroch. Metodiky navrhnuté v práci je vhodné overiť na niektorých z týchto úloh a porovnať nami dosiahnuté výsledky s výsledkami štandardných metód.
- **Algoritmická generácia postupností ofarbovania hrán.** Je potrebné hľadať vlastnosti, ktoré definujú postupnosť ofarbovania hrán s čo najnižším (resp. najvyšším) časom ofarbovania. V prípade, že bude možné tieto vlastnosti postupností rozoznať, je potrebné vytvoriť algoritmus, pomocou ktorého bude možné generovať takéto postupnosti ofarbovania hrán pre rôzne grafy.

ZOZNAM BIBLIOGRAFICKÝCH ODKAZOV

[1] ŠKRINÁROVÁ, J. 2017. *Elastický klaster*. Banská Bystrica: Belianum: Univerzita Mateja Bela v Banskej Bystrici, 2017, 106 s. ISBN 978-80-557-0642-9

[2] KOLLÁR, J. 2003. *Metódy a prostriedky pre výkonné paralelné výpočty*. Košice : Technická Univerzita v Košiciach, 2003. 100 s. ISBN 80-89066-70-4

[3] KARABÁŠ, J., MÁČAJOVÁ, E., NEDELA, R.: 6-decomposition of snarks. 2013. In European Journal of Combinatorics: 20th International workshop on combinatorial algorithms (IWOCA), Hradec nad Moravicí, 28 June - 2 July, 2009. Elsevier, 2013, Volume 34, Issue 1, s. 111-122. ISSN 0195-6698

[4] ŠKRINÁROVÁ, J.: Implementation and evaluation of scheduling algorithm based on PSO HC for elastic cluster criteria. 2014. In: Central European Journal of Computer Science. Volume 4, Issue 3, s. 191-201. ISSN 1896-1533

[5] ŠKRINÁROVÁ, J., HURAJ, L., SILÁDI, V.: A neural tree model for classification of computing grid resources with PSO tasks scheduling. 2013. Neural networks world. ISSN 1210-0552. Web of Science, Current Contents http://www.nnw.cz/doi/2013/NNW.2013.23.014.pdf

[6] ŠKRINÁROVÁ, J., VESEL, E.: Model of education and training strategy for the high performance computing. 2016. In Proceedings of 14th IEEE International Conference on Emegring eLearing Technologies and Applications. ISBN 978-1-5090-4699-7

[7] PALÚCH S., PEŠKO, Š. 2006. *Kvantitatívne metódy v logistike.* Žilina: EDIS – vydavateľstvo ŽU, 2006, 185 s. ISBN 80-8070-636-0

[8] MARTINCOVÁ, P., GRONDŽÁK K. 2004. *Operačné systémy*. EDIS – vydavateľstvo ŽU, 2004, 294 s. ISBN 80-8070-242-X

[9] DIESTEL R. 2016. *Graph theory.* Piate vydanie. Springer - Verlag, Heidelberg, 2016, 447 s. ISBN 978-3-662-53621-6

[10] TOKHI M.O., HOSSAIN M.A, SHAHEED M.H. 2003. *Parallel Computing for Real-time Signal Processing and Control.* Springer, 2003, 254 s. ISBN 978-1-85233-599-1

[11] HÄGGLUND, J.: On snarks that are far from being 3-edge-colorable. 2016. In The Elcectronic Journal of Combinatorics, Volume 23, Issue 2, Paper #P2.6. ISSN: 1077-8926

[12] HOLYER I.: The NP-Completeness of Edge-Colouring. 1981. In SIAM J. COMPUT, Volume 10, Issue 4, s. 718-720. ISSN 0097-5397

[13] KOWALIK, L.: Improved edge-coloring with three colors. 2009. In theoretical computer science, Volume 410, Issues 38-40, s. 3733-3742. ISSN 0304-3975

[14] FIOL, M. A., VILALTELLA J.: A Simple and Fast Heuristic Algorithm for Edge-coloring of Graphs. 2013. In AKCE International Journal of Graphs and Combinatorics, Volume 10, Issue 3, s. 263-272

[15] BEIGEL, R., EPPSTEIN, D.: 3-coloring in time O(1.3289n). 2005. Journal of Algorithms, Volume 54, Issue 2, s. 168-204. ISSN 0196-6774

[16] STEFFEN, E.: Non-bicritical critical snarks. 1999. Graphs Combin., Volume 15, s. 473-480

[17] BRINKMANN, G., COOLSAET, K., GOEDGEBEUR, J., MÉLOT H.: House of Graphs: a database of interesting graphs, Discrete Applied Mathematics, 161:311-314, 2013. Available at http://hog.grinvin.org

[18] BRINKMANN, G., GOEDGEBEUR J., HÄGGLUND J., MARKSRÖM K.: Generation and properties of snarks. 2013. Journal of Combinatorial Theory, Series B, Volume 103, Issue 4, s. 468-488, ISSN 0095-8956

[19] SIPSER, M. 2006. *Introduction to the Theory of Computation*. Course Technology Inc., 2006, 431 s. ISBN 0-619-21764-2

[20] DEVICI, M., BOMAN, E.G., DEVINE, K.D., RAJAMANICKAM, S.: Parallel Graph Coloring for Manycore Architectures. 2016. In proceedings of 2016 IEEE 30th Internation Parallel and Distributed Processing Symposium, IPDPS 2016, s. 892 – 901. ISBN 978-150902140-6

[21] FOSTER, I. 2003. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Boston: Addison-Wesley Longman Publishing, Boston, USA. ISBN 0201575949

[22] SOUROURI, M., RAKNES, E. B., REISSMANN, N., LANFFUTH, J., HACKENBERG, D., SCHÖNE, R., KJELDSBERG, P. G.: Towards fine-grained dynamic tuning of HPC applications on modern multi-core architectures. 2017. In proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC' 17). Article No. 41. ISBN 9781450351140

[23] EDWARDS, H. C., TROTT, C. R., SUNDERLAND, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. 2014. J Parallel Distrib Comp, Volume 74, Issue 12, s. 3202–3216.

[24] VINOD REDDY I.: Parameterized algorithms for conflict-free colorings of graphs. 2018. Theoretical Computer Science. ISSN 0304-3975

[25] GEBREMEDHIN, A. H., MANNE, F.: Scalable parallel graph coloring algorithms. 2000. Concurrency: Practice & Experience, Volume 12, Issue 12, s. 1131–1146

[26] CATALYÜREK, U. V., FEO, J., GEBREMEDHIN, A. H., HALAPPANAVAR, M., POTHEN, A.: Graph coloring algorithms for multi-core and massively multithreaded architectures. 2012. Parallel Comput, Volume 38, Issue 10, s. 576– 594.

[27] MCKAY, B.D., PIPERNO, A.: nauty and Traces User's Guide. Verzia 2.6. s. 77-78. 2016.

[28] LUMSDAINE, A., GREGOR, D., HENDRICKSON, B., BERRY, J.: Chalanges in parallel graph processing. 2007. Parallel Processing Letters, Volume 17, Issue 1, s. 5-20.

[29] AMDAHL, G.M.: Validity of the single processor approach to achieving large scale computing capabilities.1967. In AFIPS '67 (Spring) Proceedings of the April 18-20, 1967, spring joint computer conference, s.485-487.

[30] GUSTAFSON, J.L.: Reevaluating Amdahl's law. 1988. Communications of the ACM, Volume 31, Issue 5, s.532-533.

[31] KIRK, D., HWU W. 2016. *Programming Massively Parallel Processors*. Tretie vydanie. Elsevier, 2016, 576 s. ISBN 9780128119860

[32] BAESENS, B. 2014. *Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications*. Wiley, 2014, 256 s. ISBN 1118892704

[33] AGGARWAL C.C., REDDY C.K. 2013. *Data clustering: Algorithms and Applications.* Chapman and Hall/CRC, 2013, 652 s. ISBN 9781466558212

[34] BOYD, S., VANDENBERGHE, L. 2004. *Convex Optimization.* Cambridge University Press, 2004, 716 s. ISBN 0521833787

[35] GEBALI, F. 2011. *Algorithms and parallel computing.* Wiley, 2011, 364 s. ISBN 978-0470902103

[36] WEHNER, M., OLIKER, L., SHALF, J.: A real cloud computer. 2009. IEEE Spectrum, 46(10), s. 24 – 29.

[37] TOMASEVIC, M., MILUTINOVIC, V.: Hardware approaches to cache coherence in shared - memory multiprocessors: Part 1. 1994. IEEE Micro, 14(5), s. 52 – 59.

[38] KHAILANY, B.K., WILLIAMS, T., LIN, J., LONG, E.P., RYGH, M., TOVEY, D.W., DALLY, W.J.: A programmable 512 GOPS stream processor for signal image, and video processing. 2008. IEEE Journal of Solid - State Circuits, 43(1), 202 – 213 s.

[39] BISCHOF, CH., BUCKER, M., GIBBON, P., JOUBERT, G., LIPPERT, T., MOHR, B., PETERS, F. *Parallel computing: Architectures, Algorithms and Applications*. 2017. NIC-Directors, Volume 38, 830 s. ISBN 978-3-9810843-4-4

[40] COULOURIS, G., DOLLIMORE, J., KINDBERG, T., BLAIR, G. *Distributed systems – concepts and designs*. 2012. Addison-Wesley, 5. Vydanie, 1065 strán. ISBN 978-0-13-214301-1

[41] *The advanced network systems architecture (ANSA) reference manual*. 1989. Castle Hill, Cambridge, England: Architecture Project Management.

[42] PBS professional. PBS professional: Industry-leading workload manager and jobscheduler for high-performance computing. [Online]. Available: https://www.pbspro.org/.

[43] BOUVEVRON, CH., CELEUX, G. , MURPHY, T.B., RAFTERY A.E. *Model-Based Clustering and Classification for Data Science*. 2019. Cambridge University Press, 427 strán. ISBN 978-1-10-864418-1

[44] GRAMA, A., GUPTA, A., KARYPIS, G., KUMAR, V. *Introduction to Parallel Computing*. 2003. Addison Wesley, Druhé vydanie, 856 strán. ISBN: 978-0201648652

[45] CATALYÜREK, U. V., FEO, J., GEBREMEDHIN, A. H., HALAPPANAVAR, M., POTHEN, A.: Graph coloring algorithms for multi-core and massively multithreaded architectures. 2012. Parallel Comput, Volume 38, Issue 10, s. 576– 594.

[46] DEVICI, M., BOMAN, E.G., DEVINE, K.D., RAJAMANICKAM, S.: Parallel Graph Coloring for Manycore Architectures. 2016. In proceedings of 2016 IEEE 30th Internation Parallel and Distributed Processing Symposium, IPDPS 2016, s. 892 – 901. ISBN 978-150902140-6

ZOZNAM AUTORSKÝCH PUBLIKÁCIÍ

[I.] ŠKRINÁROVÁ, J., **DUDÁŠ, A.**, VESEL, E.: Model of education and training strategy for the management of HPC systems. In 2017 IEEE 14th International Scientific Conference on Informatics, Poprad, Slovakia, 14. – 17. November 2017. ISBN 978-1-5386-0889-0

- **Ohlasy:** MELICHERČÍK. M., SILÁDI, V., SVÍTEK, M., HURAJ, L.: Spreading High Performance Computing Skills with E-Learning Support. In 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, 2018, pp. 361-366.

[II.] ŠKRINÁROVÁ, J., **DUDÁŠ, A.**: A Methodology for the professional training of the management and evaluation of HPC systems. Open Computer Science, Volume 8, Issue 1, s. 68-79, ISSN 2299-1093

[III.] **DUDÁŠ, A.**, ŠKRINÁROVÁ, J., VOŠTINÁR, P., SILÁČI, J.: Improved process of running tasks in the high performance computing. In ICETA 2018: Proceedings: 16th IEEE International Conference on Emerging eLearning Technologies and Applications. - New Jersey: Institute of Electrical and Electronics Engineers. - ISBN 978-1-5386-7912-8. - s. 133-140

[IV.] ŠKRINÁROVÁ, J., **DUDÁŠ, A.**: Procesy operačného systému vo vyučovaní informatiky. In Didinfo 2019: medzinárodná konferencia o vyučovaní informatiky: 25. ročník konferencie. - ISSN 2454-051X (online). - 1. vyd. - Banská Bystrica: Univerzita Mateja Bela v Banskej Bystrici, 2019. - ISBN 978-80-557-1533-9 (online). - s. 152-156

[V.] CHOVANCOVÁ, O., PIATRIKOVÁ, L., **DUDÁŠ, A.**: Improving fuzzy c-means algorithm using particle swarm optimalization and fuzzy c-means++. In Information and digital technologies 2019: proceedings of the international conference. - 1. vyd. - Danvers: Institute of Electrical and Electronics Engineers, 2019. - ISBN 978-1-7281-1401-9. - s. 173-179.

[VI.] **DUDÁŠ, A.**, ŠKRINÁROVÁ, J., VESEL, E.: Optimization design for parallel coloring of a set of graphs in the High-Performance Computing. In 2019 IEEE 15th International Scientific Conference on Informatics, Poprad, Slovakia, 20. – 22. November 2019.

[VII.] VESEL, E., ŠKRINÁROVÁ, J., **DUDÁŠ, A.**: Comparison of in-house HPC calculation with public cloud computing for parallel algorithm containing recursive functions. In ICETA 2019 : Proceedings : 17th IEEE International Conference on Emerging eLearning Technologies and Applications. - New Jersey: Institute of Electrical and Electronics Engineers.

[VIII.] CHOVANCOVÁ, O., LAUKOVÁ, J., **DUDÁŠ, A.**, KOSTOLNÝ, J.: Interactive Data Visualization of Fuzzy Membership Functions and Fuzzy Clustering. In ICETA 2019 : Proceedings : 17th IEEE International Conference on Emerging eLearning Technologies and Applications. - New Jersey: Institute of Electrical and Electronics Engineers.

V recenznom konaní:

[IX.] ŠKRINÁROVÁ, J., **DUDÁŠ, A.**: Optimization of Functional Decomposition of Parallel and Distributed Computations in Graph Coloring with the use of High Performance Computing. In Computing and Informatics (CAI), 2020. ISSN 1335-9150 *(podané: 29.2.2020)*

[X.] **DUDÁŠ, A.,** ŠKRINÁROVÁ, J.: Edge coloring of set of graphs with the use of data decomposition and clustering. In IPSI BgD Transactions on Internet Research. ISSN 1820 - 4503 *(podané: 5.5.2020)*

PRÍLOHA A: PERMUTÁCIA S NAJNIŽŠÍM ČASOM OFARBOVANIA

V prílohe ponúkame permutáciu, pomocou ktorej sme pre 34 vrcholové grafy v kapitole 4 dosiahli ofarbovanie s najnižším časom. Permutáciu uvádzame vo vektorovej forme a vo forme matice.

VEKTOROVÝ FORMÁT PERMUTÁCIE

Vektorový formát permutácie je zjednodušeným zápisom maticového formátu permutácie. Predpokladom pre jeho zostavenie je fakt, že permutačná matica má práve jednu hodnotu 1 v každom riadku, ostatné hodnoty v permutačnej matici sú rovné nule. Vektor permutácie sme zostavili nasledovne:

- Index položiek vektora je rovný indexu riadkov matice permutácie,
- Hodnoty položiek vektora sú rovné indexu stĺpca matice permutácie kde sa nachádza hodnota 1.

Napríklad:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

*Túto maticu vieme zapísať ako vektor **[1,0]** – v prvom riadku je hodnota 1 na pozícii 1, v druhom riadku je hodnota 1 na pozícii 0.*

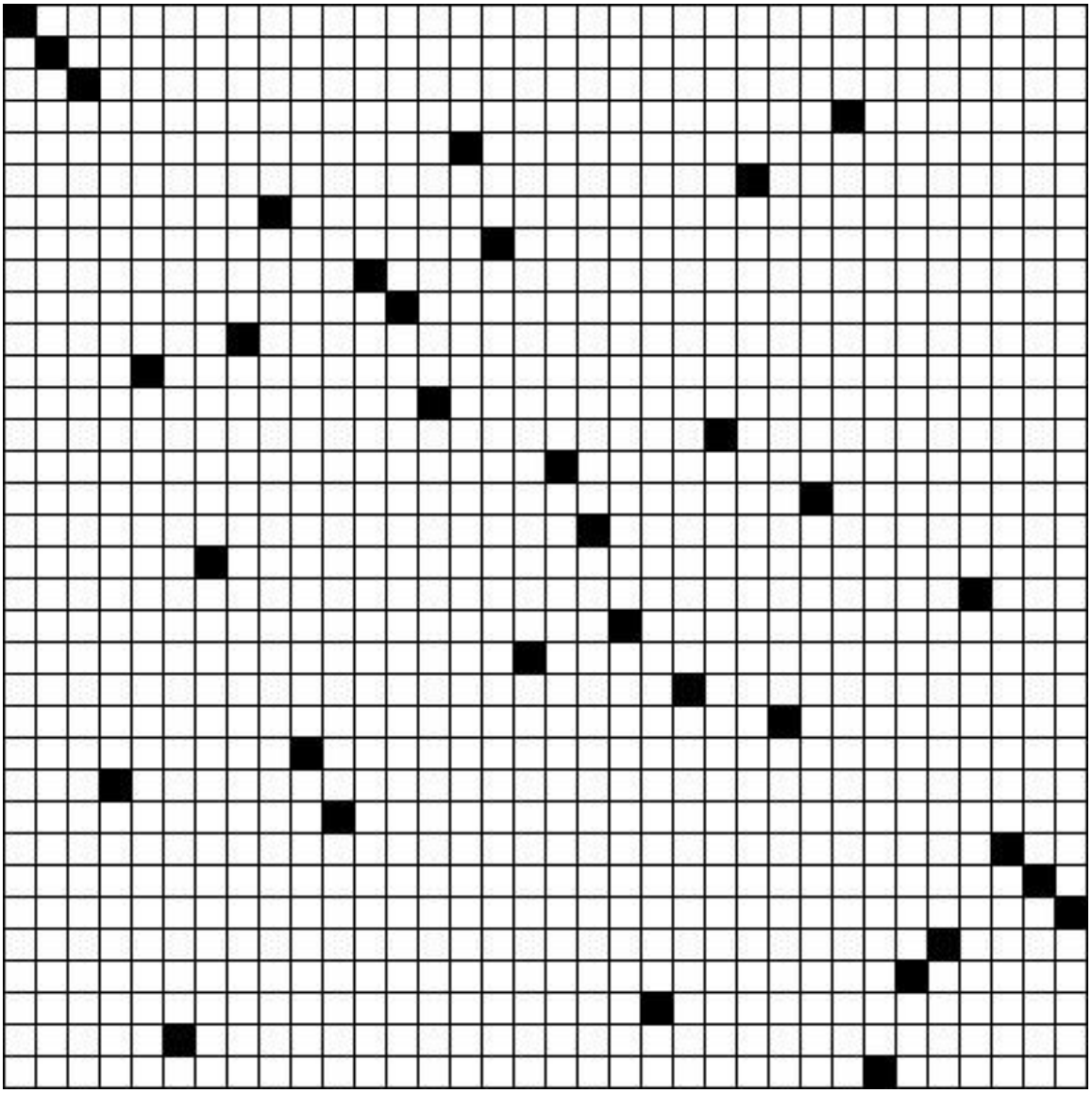
Vektorový formát permutácie s najnižším časom ofarbovania pre skúmané 34 vrcholové grafy je:

[0, 1, 2, 26, 14, 23, 8, 15, 11, 12, 7, 4, 13, 22, 17, 25, 18, 6, 30, 19, 16, 21, 24, 9, 3, 10, 31, 32, 33, 29, 28, 20, 5, 27]

MATICOVÝ FORMÁT PERMUTÁCIE

Permutačná matica má práve jednu hodnotu 1 v každom riadku, ostatné hodnoty v permutačnej matici sú rovné nule. Pre zlepšenie čitateľnosti označujeme hodnotu 1 čiernym poľom a hodnotu 0 bielym.

Maticový formát permutácie s najnižším časom ofarbovania je:



Obrázok A.1 – Maticový formát permutácie s najnižším časom ofarbovania