NI-PDP Úloha 1 (Mincut)

1. Sekvenční implementace

Úvodní fází řešení úlohy bylo vytvoření sekvenční verze algoritmu pro nalezení minimálního hranového řezu mezi dvěma disjunktními podmnožinami uzlů pevně zadané velikosti. Hledání optimálního rozdělení bylo realizováno pomocí algoritmu typu **Branch-and-Bound s prohledáváním do hloubky (BB-DFS)**, který systematicky prochází možné konfigurace a současně efektivně ořezává výpočetní prostor na základě odhadované dolní meze.

Optimalizace

Pro zvýšení efektivity výpočtu byly v algoritmu použity dvě klíčové optimalizace:

- **Dolní odhad řezu (lower bound):** V každém kroku rekurze je spočítána dolní mez na minimální možnou váhu řezu, která by mohla vzniknout v závislosti na aktuálním stavu rozdělení uzlů. Pro každý dosud nezařazený uzel je vypočtena hypotetická cena, pokud by byl přiřazen buď do podmnožiny (X), nebo do (Y), a do odhadu je zahrnuta ta z těchto možností, která je výhodnější. Součet těchto minim umožňuje algoritmu efektivně identifikovat konfigurace, které nemohou vést k lepšímu řešení než dosud nalezené, a předčasně je ořezat.
- **Heuristický odhad výchozího řešení (guesstimate):** Před samotným spuštěním prohledávání je provedeno několik náhodných rozdělení uzlů do podmnožin o požadované velikosti a pro každé z nich je spočítána váha řezu. Nejlepší nalezená hodnota je použita jako počáteční horní mez (`minCutWeight`). Bez této heuristiky by algoritmus začínal s hodnotou `INT_MAX`, případně se součtem vah všech hran, a v úvodní fázi výpočtu by nebylo možné provádět žádné ořezávání. V rámci tohoto projektu však dopad této optimalizace na výkon nebyl zásadní. Přesto se jedná o elegantní techniku, která může sehrát významnější roli v rozsáhlejších instancích problému nebo při větším paralelismu.

Výsledky

V porovnání s referenční sekvenční implementací dosahuje navržené řešení výrazně nižšího počtu rekurzivních volání i kratší doby výpočtu. Největší přínos optimalizací je patrný u grafů s vyšším počtem uzlů, kde efektivní dolní meze a počáteční heuristika významně zmenšují prostor, který je třeba rekurzivně procházet.

Tabulka výsledků referenčního řešení:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	218	#

file	n	a	recursion calls	time
graf_10_6b.txt	10	5	305	#
graf_10_7.txt	10	5	308	#
graf_15_14.txt	15	5	9000	#
graf_20_7.txt	20	7	34000	0.02
graf_20_7.txt	20	10	43000	0.02
graf_20_12.txt	20	10	142000	0.02
graf_20_17.txt	20	10	235000	0.02
graf_30_10.txt	30	10	9000000	1.9
graf_30_10.txt	30	15	17000000	3.9
graf_30_20.txt	30	15	59000000	13
graf_32_22.txt	32	10	66000000	6.3
graf_32_25.txt	32	12	268000000	24
graf_35_25.txt	35	12	662000000	65
graf_35_25.txt	35	17	21000000000	191
graf_40_8.txt	40	15	828000000	180
graf_40_8.txt	40	20	11000000000	275
graf_40_15.txt	40	15	41000000000	403
graf_40_15.txt	40	20	109000000000	2312
graf_40_25.txt	40	20	266000000000	5052

Tabulka výsledků sekvenčního řešení:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	183	0.000105423
graf_10_6b.txt	10	5	202	6.56e-05
graf_10_7.txt	10	5	277	8.83e-05
graf_15_14.txt	15	5	6019	0.00245165
graf_20_7.txt	20	7	10381	0.00776363
graf_20_7.txt	20	10	10500	0.00787336
graf_20_12.txt	20	10	39277	0.0245417
graf_20_17.txt	20	10	86411	0.0484901
graf_30_10.txt	30	10	965503	0.888179
graf_30_10.txt	30	15	733615	0.911922
graf_30_20.txt	30	15	6653893	6.93886
graf_32_22.txt	32	10	10752530	5.53075
graf_32_25.txt	32	12	39466809	20.5361
graf_35_25.txt	35	12	133293656	73.9693
graf_35_25.txt	35	17	147857192	109.697
graf_40_8.txt	40	15	15711139	16.795
graf_40_8.txt	40	20	15375162	19.2777
graf_40_15.txt	40	15	308323285	259.537
graf_40_15.txt	40	20	297291358	312.076

file	n	a	recursion calls	time
graf_40_25.txt	40	20	795689275	842.631

---

2. Paralelizace pomocí OpenMP – task-based přístup

Ve druhé fázi řešení byla sekvenční implementace rozšířena o paralelizaci prostřednictvím **OpenMP tasků**. Cílem bylo efektivně využít výpočetní kapacitu vícejádrového procesoru prostřednictvím rozdělení stavového prostoru algoritmu BB-DFS mezi více vláken.

Struktura paralelního řešení

Základní myšlenkou bylo rekurzivní větvení výpočtu ponechat, ale umožnit jeho souběžné zpracování. Pro tento účel byla zavedena prahová hloubka `TASK_DEPTH`, po jejíž úroveň se jednotlivé větve průzkumu paralelizují pomocí konstrukce `#pragma omp task`. Tímto způsobem vznikají úlohy, které pokrývají různá rozhodnutí přiřazení uzlů do podmnožin (X) a (Y), a mohou být zpracovány nezávisle.

Tedy pokud algoritmus v hloubce menší než `TASK_DEPTH` rozhoduje o zařazení dalšího uzlu, vytvoří dvě paralelní úlohy: jednu pro případ přiřazení do množiny (X), druhou pro množinu (Y).

Sdílené prostředky a synchronizace

Použití společné proměnné `minCutWeight` jako horní meze pro ořezávání stavového prostoru přináší potřebu synchronizace mezi jednotlivými tasky. K zajištění konzistence při jejím případném přepisování byla využita **kritická sekce (`#pragma omp critical`)**, v jejímž rámci dochází ke srovnání a případnému aktualizování globální nejlepší hodnoty řezu i příslušného rozdělení uzlů.

Další zajímavostí je řešení problému sledování celkového počtu rekurzivních volání v paralelním prostředí. Jednoduché sdílení čítače mezi vlákny by vedlo k výraznému zpomalení kvůli synchronizační režii. Proto bylo zvoleno efektivnější řešení pomocí **vektoru čítačů**, kde každý prvek odpovídá jednomu vláknu identifikovanému pomocí `omp_get_thread_num()`. Po skončení výpočtu se hodnoty agregují, což umožňuje zachovat přehled o komplexitě výpočtu bez negativního dopadu na výkon.

Výsledky

Paralelizace vedla k dalšímu zrychlení výpočtu. Výrazné zlepšení je patrné zejména u větších grafů a náročnějších hodnot parametru (a), kde dochází k masivnímu větvení stavového prostoru. Současné zpracování více větví umožňuje

rychleji nalézt kvalitní řešení a efektivněji provádět ořezávání pro všechny thready, ne jen ten, co ho našel.

Tabulka výsledků OpenMP task:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	193	0.000471052
graf_10_6b.txt	10	5	193	0.00681256
graf_10_7.txt	10	5	242	0.000427268
graf_15_14.txt	15	5	6020	0.00252995
graf_20_7.txt	20	7	9325	0.00191714
graf_20_7.txt	20	10	9876	0.00652727
graf_20_12.txt	20	10	38767	0.0034434
graf_20_17.txt	20	10	82052	0.010151
graf_30_10.txt	30	10	1179052	0.130106
graf_30_10.txt	30	15	793457	0.0981957
graf_30_20.txt	30	15	6888475	0.630512
graf_32_22.txt	32	10	11224584	1.04452
graf_32_25.txt	32	12	45701605	3.58045
graf_35_25.txt	35	12	121066624	10.2234
graf_35_25.txt	35	17	151615478	16.0082
graf_40_8.txt	40	15	14079045	2.66689
graf_40_8.txt	40	20	16678000	3.33941
graf_40_15.txt	40	15	438751243	51.2015
graf_40_15.txt	40	20	302219577	45.8628
graf_40_25.txt	40	20	798791158	114.198

---

3. Paralelizace pomocí OpenMP – data-paralelní přístup

Ve třetí fázi byla implementace změněna tak, aby cílila na **data-paralelní zpracování**. Cílem bylo dosáhnout dalšího zrychlení výpočtu prostřednictvím zpracování více nezávislých výpočetních jednotek ve větší míře najednou a na úrovni dat mimo rekurzi samotnou.

Struktura řešení

Zásadní změnou oproti předchozímu přístupu bylo zavedení **frontier-based strategie**, při níž se stavový prostor algoritmu předem „rozřeže“ na menší části. Konkrétně je pomocí funkce `generatePartialSolutions` vygenerována množina částečných konfigurací až do předem dané hloubky (`frontierDepth`). Každá z těchto konfigurací představuje alternativní výchozí stav pro další průchod rekurzivním BB-DFS algoritmem.

Následně je tato množina částečných řešení zpracována paralelně pomocí direktivy `#pragma omp parallel for`. Každý thread tak pracuje nezávisle na jiné části stavového prostoru, což umožňuje lepší využití výpočetních prostředků při současném zachování determinismu výpočtu.

Výpočet dolní meze (`parallelLB`) byl rovněž upraven: pro malé rozsahy je nadále zpracováván sekvenčně, zatímco pro větší vstupy se aktivuje paralelní verze založená na `#pragma omp for reduction`.

Praktické poznámky

Hodnota `frontierDepth` byla zvolena jednoduše jako minimum mezi velikostí požadované podmnožiny (a) a konstantou 16. Vzhledem k omezenému rozsahu projektu nebylo nutné její hodnotu detailně ladit a postačil tento *okometrický odhad*.

Dynamické rozdělování úloh pomocí fronty nebylo zavedeno – především z důvodu jednoduchosti implementace a časových omezení v rámci jiných studijních povinností. Vzhledem k tomu, že jednotlivé částečné konfigurace měly podobnou výpočetní náročnost, nebylo třeba dále řešit adaptivní plánování nebo přenos úloh mezi vlákny.

Uvažována byla rovněž možnost převést rekurzivní DFS na iterativní variantu, což by mohlo přinést zlepšení z hlediska cache locality nebo jednodušší správu paralelismu.

Výsledky

Data-paralelní přístup přinesl další zrychlení výpočtu ve srovnání s předchozí task-based verzí, i když rozdíl nebyl tak výrazný jako při přechodu ze sekvenční verze. Významný přínos je patrný u středně velkých / velkých grafů, kde počet částečných konfigurací umožňuje dostatečně rovnoměrné rozdělení práce mezi vlákna bez výrazné režijní zátěže.

Tabulka výsledků OpenMP data:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	115	0.00013593
graf_10_6b.txt	10	5	158	0.00699702
graf_10_7.txt	10	5	206	0.00770061
graf_15_14.txt	15	5	5990	0.00645705
graf_20_7.txt	20	7	5535	0.00152065
graf_20_7.txt	20	10	5628	0.00334884
graf_20_12.txt	20	10	17379	0.00476989
graf_20_17.txt	20	10	59182	0.00298345
graf_30_10.txt	30	10	370831	0.036411
graf_30_10.txt	30	15	328478	0.0657218

file	n	a	recursion calls	time
graf_30_20.txt	30	15	4011800	0.321835
graf_32_22.txt	32	10	9294131	0.587601
graf_32_25.txt	32	12	26570908	1.60171
graf_35_25.txt	35	12	90505714	5.73599
graf_35_25.txt	35	17	68918082	6.07222
graf_40_8.txt	40	15	2690770	0.422758
graf_40_8.txt	40	20	1481978	0.34287
graf_40_15.txt	40	15	75201953	8.10201
graf_40_15.txt	40	20	46390866	6.37945
graf_40_25.txt	40	20	527679964	60.2925

---

4. Implementace pomocí MPI

Poslední fází projektu byla implementace algoritmu s využitím knihovny MPI a hybridního přístupu kombinujícího MPI s OpenMP. Celá architektura byla navržena ve stylu **master–slave**: hlavní proces (master) dynamicky přiděloval úlohy podřízeným procesům (slaves), které paralelně zpracovávaly části stavového prostoru.

Implementace prošla třemi významnými verzemi, které se postupně zaměřovaly na správnost i výkon.

První verze: rychlá, ale příliš optimistická

První varianta byla založena na jednoduchém **master–worker** modelu: - Master proces načetl graf a vygeneroval počáteční sadu částečných řešení ("frontier"). - Každý worker dostal úlohu, provedl hluboké prohledávání DFS a odeslal nejlepší nalezený výsledek zpět. - K přenosu bylo využito blokující schéma (`MPI_Send`, `MPI_Recv`).

Použitý dolní odhad (`lowerBound`) byl poměrně přísný, což vedlo k **rychlému výpočtu**, ale také k **chybné eliminaci** některých větví obsahujících optimální řešení. Výsledné řezy byly mírně vyšší než skutečné optimum — rozdíl byl ale **konzistentní a malý**.

Tabulka výsledků MPI 1:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	83	0.0388391
graf_10_6b.txt	10	5	76	0.0476822
graf_10_7.txt	10	5	100	0.0330964
graf_15_14.txt	15	5	136	0.040798

file	n	a	recursion calls	time
graf_20_7.txt	20	7	268	0.0583538
graf_20_7.txt	20	10	1404	0.0513671
graf_20_12.txt	20	10	3068	0.0590673
graf_20_17.txt	20	10	3687	0.0598901
graf_30_10.txt	30	10	2699	0.0627989
graf_30_10.txt	30	15	69245	1.07399
graf_30_20.txt	30	15	102847	1.1142
graf_32_22.txt	32	10	1658	0.0713758
graf_32_25.txt	32	12	8812	0.211281
graf_35_25.txt	35	12	8027	0.221909
graf_35_25.txt	35	17	313911	2.6525
graf_40_8.txt	40	15	38985	1.54672
graf_40_8.txt	40	20	143840	2.80769
graf_40_15.txt	40	15	38448	1.57929
graf_40_15.txt	40	20	137670	2.80487
graf_40_25.txt	40	20	4872425	5.16683

Druhá verze: správná, ale pomalá

Ve druhé iteraci byl dolní odhad upraven tak, aby byl **volnější**: - U každého neobsazeného vrcholu se brala minimální cena mezi přiřazením do X a do Y. - Tím se výrazně omezilo předčasné prořezávání možných řešení.

Výsledkem bylo dosažení **správných minimálních řezů** (identických s referenčními výsledky), avšak za cenu **výrazného zpomalení**. U větších grafů narostla doba běhu na stovky sekund.

Tabulka výsledků MPI 2:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	241	0.038385
graf_10_6b.txt	10	5	252	0.0494458
graf_10_7.txt	10	5	316	0.049645
graf_15_14.txt	15	5	5598	0.0466972
graf_20_7.txt	20	7	13214	0.0775172
graf_20_7.txt	20	10	20462	0.0899592
graf_20_12.txt	20	10	59939	0.117091
graf_20_17.txt	20	10	119046	0.133543
graf_30_10.txt	30	10	959989	0.465937
graf_30_10.txt	30	15	1709511	4.01164
graf_30_20.txt	30	15	12693905	7.92533
graf_32_22.txt	32	10	9786022	3.48629
graf_32_25.txt	32	12	37036849	11.1633

file	n	a	recursion calls	time
graf_35_25.txt	35	12	121935274	36.6458
graf_35_25.txt	35	17	231280089	88.4059
graf_40_8.txt	40	15	23371224	17.0876
graf_40_8.txt	40	20	42062362	30.6209
graf_40_15.txt	40	15	311500404	120.68
graf_40_15.txt	40	20	742044041	295.644
graf_40_25.txt	40	20	#	600+(timed out on arm_long)

Třetí verze: finální hybridní řešení

Finální verze kombinovala výhody obou předchozích přístupů:

- **Těsný dolní odhad** vycházel z baseline, kdy všechny uzly šly do Y, a následně byly vybrány nejlepší přechody do X pomocí `nth_element`.
- Funkce generující "frontier" **neprováděla ořez** podle aktuálního nejlepšího řešení – bylo tak zajištěno, že žádná část prostoru nebyla předčasně vyřazena, jako tomu bylo u první MPI implementace.
- Při aktualizaci nejlepšího nalezeného řezu v rámci OpenMP DFS byla přidána **synchronizace pomocí `#pragma omp critical`**.
- Práce byla mezi procesy **dynamicky přerozdělována** podle potřeby.

Tato verze byla schopna nalézt správné výsledky a přitom si zachovala **výrazně lepší výkon** než druhá iterace.

Tabulka výsledků MPI 3:

file	n	a	recursion calls	time
graf_10_5.txt	10	5	199	0.700087
graf_10_6b.txt	10	5	181	0.0399271
graf_10_7.txt	10	5	259	0.042686
graf_15_14.txt	15	5	5636	0.0464536
graf_20_7.txt	20	7	5948	0.0430638
graf_20_7.txt	20	10	9030	0.0709878
graf_20_12.txt	20	10	23281	0.103996
graf_20_17.txt	20	10	61412	0.137302
graf_30_10.txt	30	10	94121	0.203179
graf_30_10.txt	30	15	202464	2.19649
graf_30_20.txt	30	15	3006829	5.59965
graf_32_22.txt	32	10	1007516	0.646947
graf_32_25.txt	32	12	5269356	3.06861
graf_35_25.txt	35	12	12568336	7.02315
graf_35_25.txt	35	17	47029498	33.2186
graf_40_8.txt	40	15	2956604	7.38073

file	n	a	recursion calls	time
graf_40_8.txt	40	20	5213256	14.561
graf_40_15.txt	40	15	23515327	21.7178
graf_40_15.txt	40	20	100363832	68.5089
graf_40_25.txt	40	20	154865031	98.7472

Architektura programu

- **Master proces**:
  - Načítá graf a připravuje částečné úlohy ("frontier").
  - Přiděluje úlohy workerům a přijímá od nich výsledky.
  - Dynamicky vyvažuje zátěž mezi workery.
- **Worker procesy**:
  - Přijímají částečné řešení a provádějí hluboké DFS s paralelizací pomocí OpenMP.
  - Odesílají nejlepší nalezený řez zpět masterovi.
- **OpenMP**:
  - Každý worker využívá task paralelizaci (`#pragma omp task`) při průchodu DFS stromem.
  - Dynamické rozdělování úloh umožňuje lepší využití dostupných CPU jader.

Možnosti dalšího vylepšení

Potenciálně by bylo možné implementaci dále optimalizovat například:

- **Zavedením work stealingu** mezi MPI procesy pro ještě lepší vyvážení zátěže.
- **Lepším návrhem frontier úrovně**, například preferováním konfigurací s vyšší odhadovanou složitostí.
- **Využitím asynchronní komunikace** v MPI (`MPI_Isend`, `MPI_Irecv`) k překrytí komunikace a výpočtu.

---

5. Diskuze výsledků

Shrnutí očekávání

V rámci projektu bylo očekáváno, že postupným přechodem od sekvenční implementace přes různé paralelní verze dojde ke znatelnému zlepšení výkonnosti řešení problému minimálního hranového řezu.

Konkrétně bylo předpokládáno:

- **Sekvenční verze** bude referenční základ bez paralelismu.
- **OpenMP task paralelismus** umožní zrychlení díky nezávislému prohledávání větví stavového prostoru.

- **OpenMP data paralelismus** přinese další optimalizaci díky lepšímu rozdělení práce.
- **MPI implementace** ve variantě master–slave bude schopná škálovat na více výpočetních uzlů a dosáhne nejvyšší výkonnosti.

Současně bylo počítáno s tím, že přechod na distribuované řešení (MPI) s sebou ponese vyšší náklady na synchronizaci a správu úloh, nicméně se očekávalo, že při dostatečně velkých instancích grafu budou tyto režie kompenzovány rozsáhlejší paralelizací. Tedy ideálně by bylo v grafech vidět *klesajíci schody* odrážející klesající čas běhu programu.

Co se týče rekurzivních volání, dalo by se odhadovat, že jejich počet v nějaké rozumné míře napříč implementacemi poroste, neboť překrytí více vláken pracujících najednou intuitivně nese nějaký pokles v *efektivitě za volání*. Důvodem k tomuto jevu může být neaktuální (a horší) globální minimum nějakého vlákna, což povede k více volání než by bylo nutně třeba. Ovšem trade-off je naprosto v pořádku, pokud vede ke zrychlení.

Logicky bylo očekáváno, že malé grafy budou rychlejší vzhledem k overheadu paralelizace. Přitom s rostoucí složitostí problému by měly paralelní instance být rychlejší a lepší – lepší škálování.

Přehled reálných výsledků

Grafy jsou prezentováný v následující kapitole (6).

Výsledky jednotlivých fází implementace přinesly několik poznatků. Co se týče času (grafy 1,2,3,4)

- **Sekvenční implementace** byla nejrychlejší pro nejmenší problémy (první 3), poté začala výrazně prohrávat.
- **První verze MPI implementace** nabídla nejrychlejší výpočty ze všech, avšak za cenu mírně vyšší hodnoty nalezeného řezu oproti optimu. Přesto by se tato varianta hodila v případech, kdy je prioritou rychlost nad absolutní přesností výsledku. Výsledek byl konzistentě méně než 10% odchýlen od optima a vždy vracel validní řez.
- **OpenMP data paralelismus** se stal nejrychlejší 100% korektní implementací.
- **Druhá verze MPI** korigovala problém s přesností, avšak vedla ke znatelnému zpomalení výpočtu v důsledku méně efektivního ořezávání stavového prostoru.
- **Finální MPI verze** díky zavedení těsného dolního odhadu a úplné enumerace frontier poskytla správné výsledky s přijatelným časem běhu (srovnatelný s OpenMP task).

Již zmíněné *schody* u časů jsou vidět v grafech 2 a 3 (až na malé grafy, kde dominuje sekvenční běh). První verze MPI implementace u větších grafů tento trend dodržuje, avšak nejrychlejší plně korektní varianta – MPI 3 – je bohužel na úrovni task paralelizace.

Výsledky rekurzivních voláni (grafy 5 a 6) vcelu odpovídají předpokladům, ale zajímavý je pokles rekurzivních volání u větších problémů. Zde paralelismus a následná synchronizace globálně nejlepšího nalezeného řešení vede k tomu, že více vláken může skončit dříve než déle tak, jak bylo původně uvažováno.

Detailní rozbor podle variant

Sekvenční verze Vlastní sekvenční implementace dosáhla velmi dobrého zrychlení oproti referenčnímu řešení.
Důvody lepší efektivity byly především:

- Zavedení **rychlejšího dolního odhadu** (`betterLowerBound`), který přesněji odhadl zbývající náklady budoucího přiřazování a umožnil včasné ořezání neplodných větví.
- Použití **heuristiky guesstimate**, která umožnila nalézt kvalitní horní mez ještě před spuštěním vlastního prohledávání, čímž se dramaticky snížil počet nutných rekurzí.

Výsledkem byla **mnohonásobně kratší** doba běhu u všech testovaných grafů v porovnání s referenční sekvenční verzí.

OpenMP task paralelismus První paralelní verze postavená na OpenMP task paralelismu přinesla další významné urychlení.
Hlavní myšlenkou bylo:

- Každou větev stavového prostoru prohledávat samostatným taskem, pokud byla hloubka rekurze pod určitou hranicí (`TASK_DEPTH`).
- Zachovat globální nejlepší řešení pomocí `#pragma omp critical`, čímž se zajistila správnost i v multithreaded prostředí.

Zajímavým aspektem bylo také agregování počtu rekurzivních volání na základě **thread ID** (místo synchronizovaných inkrementací), což přispělo k udržení vysokého výkonu.

Tato fáze přinesla znatelný skok ve výkonu, zejména na středně velkých grafech.

OpenMP data paralelismus Ve třetí fázi byl model upraven na **master–slave** přístup v rámci jednoho procesu:
nejprve se vygenerovalo více "startovních pozic" (`generatePartialSolutions`), a poté se každá z nich samostatně prohledávala pomocí DFS.

Výsledkem bylo:

- **Vyšší vyváženost zátěže** mezi thready.
- Možnost **efektivnější práce s velkými grafy**, protože průzkum disjunktních oblastí stavového prostoru probíhal nezávisle.

Zlepšení výkonu bylo v této fázi již mírnější než v předchozím přechodu, ale stále měřitelné.

MPI – první verze První pokus o distribuovanou variantu přes MPI následoval klasický **master–slave** model:

- Master generoval úlohy až do hloubky `frontierDepth` a dynamicky je přiděloval workerům.
- Úlohy obsahovaly kromě částečného přiřazení také aktuální globální nejlepší známou hodnotu (`globalBound`), což umožňovalo workerům okamžitě ořezávat neefektivní větve.

Díky velmi "agresivnímu" dolnímu odhadu tato verze běžela **extrémně rychle**, ale nalezla řezy o něco horší než optimum – což bylo dáno tím, že některé větve vedoucí k lepšímu řešení byly předčasně ořezány.

Přesto se tato implementace jeví jako **velmi vhodná pro úlohy, kde je klíčový čas a nevyžaduje se absolutní optimalita**.

MPI – druhá verze Ve snaze odstranit problémy s nepřesností byla ve druhé MPI verzi dolní odhad uvolněn:

- Místo výběru "optimálnějších" budoucích přiřazení bylo rozhodování více konzervativní.

Tím se podařilo odstranit chyby v hledání minima, ale:

- **Počet prozkoumaných stavů** dramaticky narostl.
- Výsledný čas výpočtu se výrazně prodloužil, v některých případech až několikanásobně oproti původní MPI verzi.

MPI – finální verze Finální verze spojila výhody obou přístupů:

- **Těsný dolní odhad** využívající výběr `remainX` nejlepších přírůstkových nákladů (`deltas`), který zároveň nezpůsoboval ořezání příliš brzy.
- **Úplná enumerace všech částečných přiřazení** do hloubky `frontierDepth` bez předčasného pruningu, čímž se eliminovalo riziko, že by byla přehlédnuta nějaká potenciálně optimální větev.
- **Kombinace OpenMP + MPI**, kdy každá úloha byla dále paralelně řešena na více jádrech jednoho uzlu.

Výsledkem byla správnost nalezených řezů a přijatelné časy běhu, i když u menších grafů se OpenMP verze stále ukazovala jako rychlejší.

---

6. Prezentace výsledků - Grafy

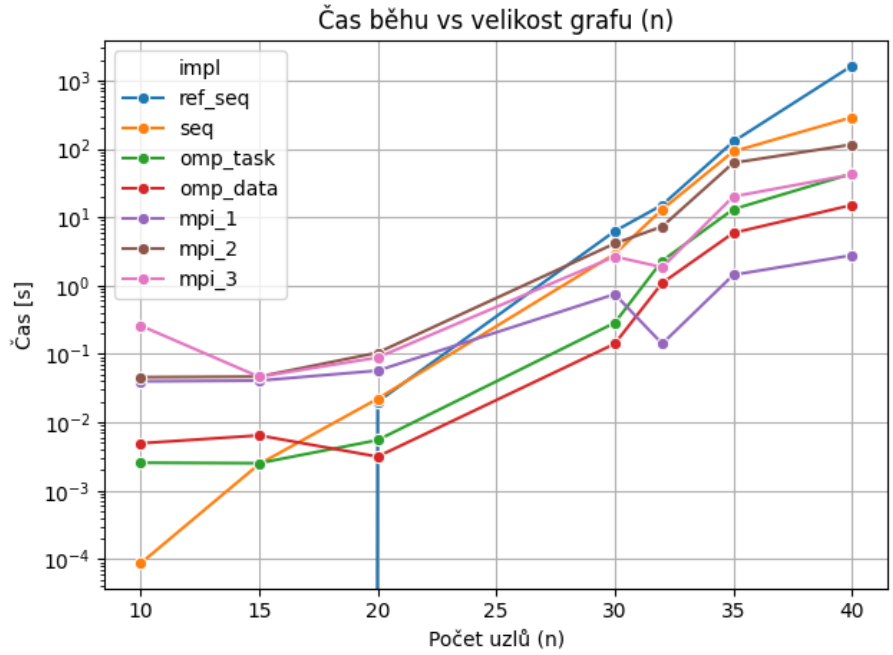

Figure 1: Porovnání časů (log scale) běhů jednotlivých implementací v závislosti na velikosti grafu `n`

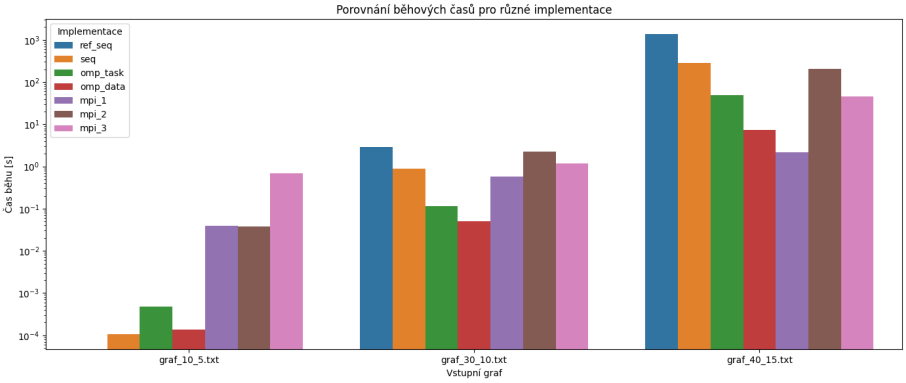

Figure 2: Porovnání časů (log scale) běhů pro 3 vybrané grafy

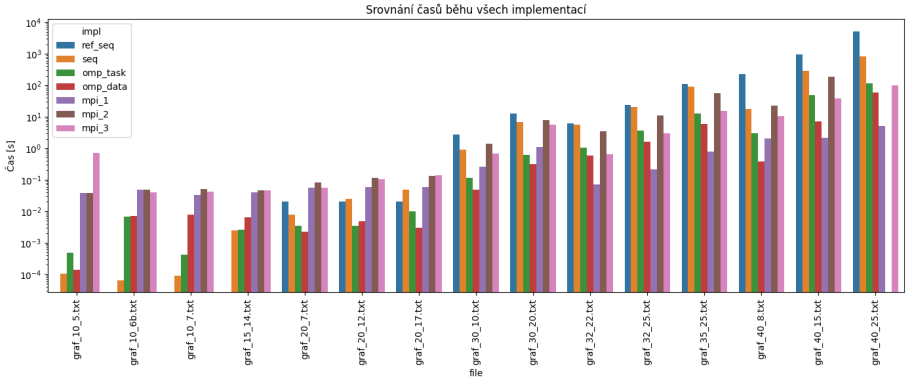

Figure 3: Porovnání časů všech implementací

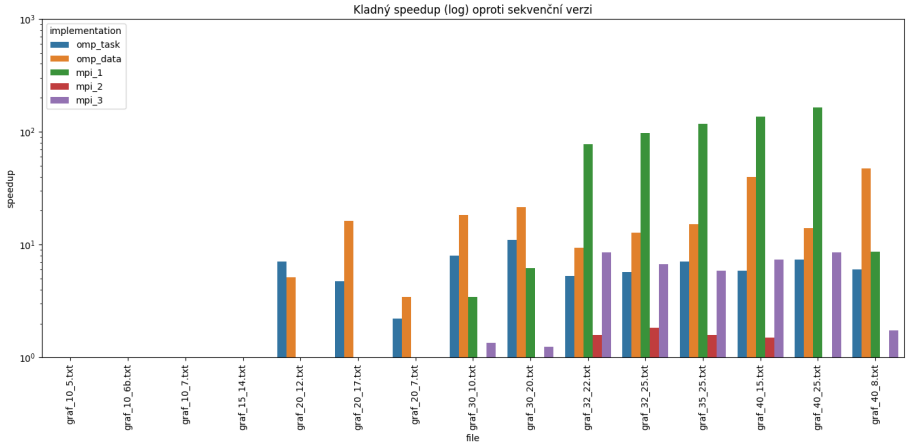

Figure 4: Speedup (log scale) ostatních oproti sekvenční implementaci

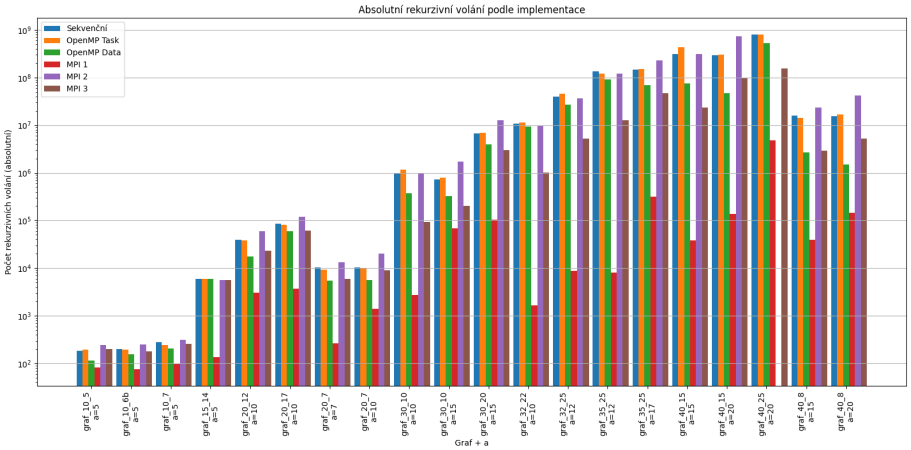

Figure 5: Rekurzivní volání běhů jednotlivých implementací

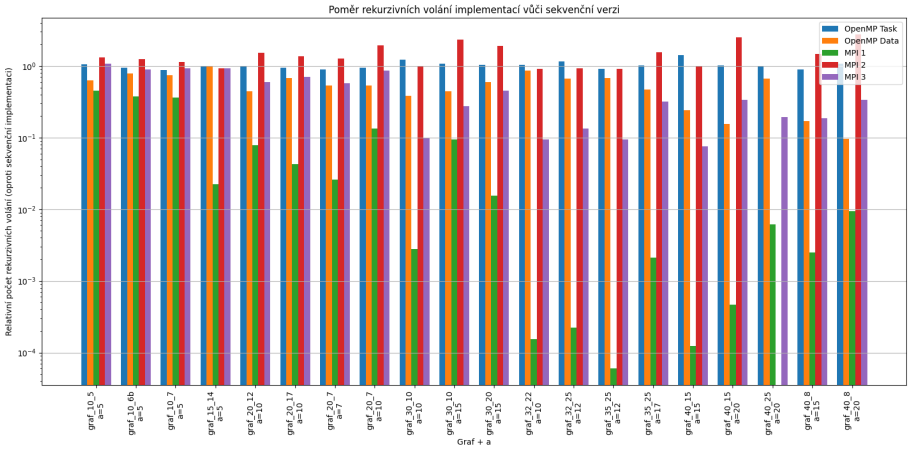

Figure 6: Porovnání rekurzivních volání oproti sekvenční implementaci

7. Závěr

V rámci semestrální práce byl implementován a analyzován algoritmus pro hledání minimálního hranového řezu grafu při fixní velikosti partice.
Implementace prošla několika fázemi:

- **Sekvenční řešení**, kde byla použita metoda Branch & Bound s heuristikou odhadu horní meze.
- **Paralelizace pomocí OpenMP**, nejprve na úrovni tasků, následně i s přípravou startovních konfigurací.
- **Distribuovaná verze pomocí MPI**, kombinovaná s OpenMP pro hybridní paralelismus.

Provedená měření ukázala:

- Výrazné zrychlení již při přechodu ze sekvenční na OpenMP verzi (task a data paralelismus).
- Finální verze MPI+OpenMP implementace dokázala najít správná řešení při zachování přijatelného výpočetního času.

Mezi klíčové optimalizace, které ovlivnily efektivitu a kvalitu výsledků, patřilo:

- Využití heuristiky pro počáteční odhad horní meze (guesstimate).
- Zavedení přesnějšího dolního odhadu v rámci Branch & Bound postupu.
- Dynamické rozdělování práce mezi procesy v MPI systému.

Během projektu bylo rovněž ukázáno, že příliš agresivní prořezávání může vést k nekorektním výsledkům, a proto bylo potřeba jemně balancovat mezi rychlostí průchodu a garancí správnosti.

Výsledky ukazují, že i relativně jednoduché hybridní paralelní přístupy mohou u nestandardních kombinatorických úloh přinést výrazné zlepšení výkonu, a že při správné kombinaci heuristik a dynamického řízení lze dosáhnout rozumného kompromisu mezi rychlostí a přesností.

Příloha

Příloha má následující strukturu:

```
./
|
|- src/
|  |- seq/         # Sekvenční implementace algoritmu
|  |- task/        # OpenMP task-based paralelizace
|  |- data/        # OpenMP data-paralelní paralelizace
|  |- mpi/         # MPI + OpenMP hybridní paralelizace
|
|- logs/
|  |- ref_seq.csv  # Výsledky referenční sekvenční implementace
|  |- seq.csv      # Výsledky vlastní sekvenční implementace
```

```
|  |- mp_task.csv  # Výsledky OpenMP task-based implementace
|  |- mp_data.csv  # Výsledky OpenMP data-paralelní implementace
|  |- mpi_1.csv    # Výsledky první verze MPI implementace
|  |- mpi_2.csv    # Výsledky druhé verze MPI implementace (bez posledního grafu)
|  |- mpi_3.csv    # Výsledky finální verze MPI implementace
|
|- scripts/ # scripty pro generování grafů, atd.
```