GAL - Dokumentace k projektu
Téma 21 - Paralelizace - Maximální tok v síti

Michal Dostál (xdosta51@stud.fit.vutbr.cz)
Oliver Kuník (xkunik00@stud.fit.vutbr.cz)

14. prosince 2022

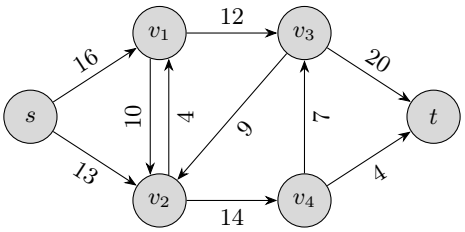
1 Úvod

Úkolem tohoto projektu bylo nastudovat problém maximálního toku v síti, konkrétně nastudovat možné paralelní algoritmy pro řešení tohoto problému a tyto algoritmy implementovat. Dále se tento projekt zabývá implementací sekvenčního algoritmu a porovnáním sekvenčního a paralelního algoritmu. Paralelní i sekvenční algoritmus `Ford Fulkerson` [1]. byl implementovaný podle článku *A Parallel Ford-Fulkerson Algorithm For Maximum Flow Problem* [3] a sekvenční algoritmus Edmond Karp podle knihy *Introduction to Algorithms* [2]. Byl pokus i o implementaci dalšího paralelního algoritmu, který je popsán v článku *Efficient Implementation of a Synchronous Parallel Push-Relabel Algorithm* [1], bohužel se v tomto dokumentu při popisu implementace vyskytují chyby a nebyly jsme schopni všechny tyto chyby odhalit a následně opravit, a proto se nám tento algoritmus nepovedlo implementovat.

2 Síť

V teorii grafů je síť, také známa jako provozní síť, orientovaný graf, kde každá hrana má kapacitu a každá hrana má tok. Velikost toku hrany nemůže přesáhnout velikost kapacity. Někdy bývá orientovaný graf pojmenovaný také jako síť, kde vrcholy jsou nazvány uzly a hrany jsou pojmenovány oblouky. Tok musí splňovat taková omezení, že velikost toku, který do uzlu vchází se musí rovnat toku, který z uzlu vychází, vyjímkou je zdroj, který může mít větší velikost odchozího toku, nebo spotřebič, který může mít větší velikost toku příchozího. Síť může být použita pro modelování dopravního provozu, modelování toku tekutin v potrubích, modelování elektrického proudu v elektrickém obvodu, nebo v čemkoli podobném, kde něco putuje v síti uzlů [3].

Na obrázku 1 můžeme vidět příklad sítě, která je vhodná pro řešení problému maximálního toku v síti.



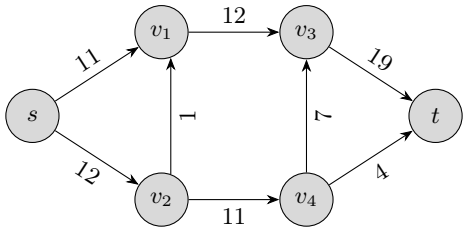
Obrázek 1: Příklad sítě

3 Maximální tok v síti

Problém maxímálního toku v síti zahrnuje hledání proveditelného toku skrze síť, která má jeden zdroj a jeden spotřebič a pro tento tok platí, že je maximální. Na problém maximálního toku v síti může být nahlíženo jako

[1] Ford Fulkerson v tomto dokumentu bude označovat pojmenování implementace podle článku [3].

na speciální případ více komplexního problému toku sítě, který se nazývá oběhový problém [3]. Na obrázku 2 je znázorněno, jak vypadá síť po provedení algoritmu `Ford Fulkerson` na síti z obrázku 1. V takovém případě je maximální tok v síti hodnota 23.



Obrázek 2: Síť po provedení algoritmu `Ford Fulkerson|`.

4 Sekvenční algoritmy

Byly implementovány sekvenční algoritmy Edmond Karp a `Ford Fulkerson`. Algoritmus Edmond Karp je stejný jako algoritmus `Ford Fulkerson`, liší se pouze ve způsobu, kdy si značí o kolik se daný tok zvýší. Algoritmus `FF` si zvýšení toku značí rovnou, ale EK ne, z tohoto důvodu musí algoritmus při úpravě reziduální sítě projít nalezenou cestu dvakrát. Tyto algoritmy pro následující vyhodnocení byly implementovány ve dvou variantách, kdy jedna varianta používá k ukládání hodnot matici sousednosti a druhá seznam sousednosti.

4.1 Implementace sekvenčního algoritmu

Sekvenční algoritmus `Ford Fulkerson` je implementován podle článku [3]. Algoritmus začíná jakýmkoli možným tokem (např. f = 0). Obecně je uzel v jednom ze tří stavů: neoznačený, označený a naskenovaný nebo označený a nenaskenovaný. Při vstupu do 1. kroku jsou všechny uzly neoznačeny. První krok nastaví zdroj na označený a nenaskenovaný.

Krok č.1 Nejprve se označí zdroj $s, l(s) = \infty$

Krok č.2 Vyberte libovolný uzel u, který je označený a není naskenovaný, pokud už neexistuje uzel, který je označený a není naskenovaný, pak aktuální tok je maximální tok. Pro všechny uzly $v \in N(u)$, kde $N(u)$ je množina všech sousedních uzlů u, tj. $(u, v) \in E$ nebo $(v, u) \in E$. Pokud v není označeno, pak:

- Pokud $(u, v) \in E$ a $f(u, v) < c(u, v)$, nastav label $(u, +, l(v))$ uzlu v. kde $l(v) = min(l(u), c(u, v) - f(u, v))$
- Jestliže $(v, u) \in E$ a $f(v, u) > 0$, nastav label $(u, -, l(v))$ uzlu v, kde $l(v) = min(l(u), f(v, u))$

Uzel u zůstane označený a naskenovaný a uzel v bude označený a nenaskenovaný. Pokud je uzel t označen, pak algoritmus pokračuje na krok č. 3, jinak na krok č. 2.

Krok č. 3 Nechť $x = t$, bude se provádět následující, dokud $x = s$

- Pokud označení $x = (y, +, l(x))$, pak nechť $f(y, x) = f(y, x) + l(t)$
- Pokud je označení $x(y, -, l(x))$, pak nechť $f(x, y) = f(x, y) - l(t)$
- Nechť $x = y$

Vrať se ke kroku č.1.

4.2 Časová složitost

- Pro implementace se seznamem sousednosti je časová složitost.

$$\text{DTIME} \approx \mathcal{O}(|V| * |E|^2)$$

- Implementace s maticí sousednosti mají časovou složitost.

$$\text{DTIME} \approx \mathcal{O}(|V|^3 * |E|)$$

4.3 Prostorová složitost

- Pro implementace se seznamem sousednosti je prostorová složitost.

$$\text{DSPACE} \approx \mathcal{O}(|V| + |E|)$$

- Implementace s maticí sousednosti mají prostorovou složitost.

$$\text{DSPACE} \approx \mathcal{O}(|V|^2)$$

5 Paralelní algoritmus

Byl implementován paralelní algoritmus `Ford Fulkerson` ve dvou verzích a to s ukládáním hodnot do matice sousednosti a druhá verze s ukládáním do seznamu sousednosti. Algoritmus je implementován stejným způsobem, jak je popsáno v 4.1 s tím rozdílem, že je zde paralelizováno hledání do šířky. Původní krok č. 2 se vykonává pro všechny hrany grafu paralelně a aby při označování uzlů nedošlo k vzájemnému přepisu, tak každý uzel je opatřený zámkem.

5.1 Časová složitost

- Implementace se seznamem sousednosti má časovou složitost (pouze odhad).

$$\text{DTIME} \approx \mathcal{O}\left(|V| * \frac{|E|^3}{n}\right)$$

- Implementace s maticí sousednosti má časovou složitost (pouze odhad).

$$\text{DTIME} \approx \mathcal{O}\left(|V|^5 * \frac{|E|}{n}\right)$$

Kde n je počet použitých vláken.

5.2 Prostorová složitost

- Implementace se seznamem sousednosti má prostorovou složitost.

$$\text{DSPACE} \approx \mathcal{O}(|V| + |E|)$$

- Implementace s maticí sousednosti má prostorovou složitost.

$$\text{DSPACE} \approx \mathcal{O}(|V|^2)$$

6 Generování sítí

Pro generování sítí byly vytvořeny skripty v adresáří `network_generator` a to skript s názvem `gen_graphs.py`, který vytvoří 50 různých sítí od velikosti sítě o 10 uzlech po velikost sítě s 500 uzly. Celkově je tedy vygenerováno 24500 náhodných sítí. Tyto sítě jsou vytvořeny pomocí nástroje pynetgen. [2] Pomocí funkce `netgen_generate` s parametry `mincost=1` a `maxcost=1` je vygenerována instance problému maximálního toku v síti, dále je zde potřeba nastavit parametr `density`, který udává hustotu v dané síti. Hustota je vypočítána a nastavena následujícím způsobem.

- Sítě, které obsahují méně než 50 uzlů:

$$|arcs| = 4 * |nodes|$$

[2] pynetgen je python modul pro vytváření instancí problémů v síti viz. Odkaz na pynetgen

- Sítě, které obsahují více než 50 uzlů:

$$|arcs| = \left(\frac{|nodes|}{3}\right)^2$$

Dále se zde nachází skript `gen_graphs_mean_degree.py`, který generuje různě velikou hustotu sítě na velikosti sítě o 500 uzlech. Tento skript postupně zvětšuje hustotu sítě od hodnoty $density = 25 * 500$ do hodnoty $density = 250 * 500$, pro různé hodnoty hustoty vygeneruje po každé 5 náhodných sítí. Tyto sítě jsou vygenerovány do složky `graphs/mean/degree`.

7 Použití programu

7.1 Návrh programu

Program pro vypočtení maximálního toku v síti je implementován v jazyce `C++`, jsou zde použity pouze standardní knihovny jazyka `C++`, pro paralelizaci algoritmů byla použita knihovna `OpenMP`.

Generování náhodných sítí je implementováno v jazyce `Python`, v této implementaci je použita knihovna `pynetgen`, která se jednoduše na serveru `merlin` doinstaluje pomocí příkazu `pip install pynetgen`.

Další součásti aplikace jsou skripty pro vyhodnocení rychlosti výpočtu maximálního toku v síti nad jednotlivými implementacemi s použitím všech vygenerovaných sítí. Výstup z těchto skriptů bude uložen pro další zpracování.

Poslední součást aplikace jsou skripty v jazyce `Python`, které z vystupních dat z předchozí části vykreslí grafy udávající rychlost jednotlivých algoritmů.

7.2 Program pro výpočet maximálního toku v síti

Všechny implementace výpočtu maximálního toku v síti jsou uloženy ve složce `bin/`. Pro vytvoření spustitelné binárky stačí v kořenovém adresáři spustit příkaz `make` a vznikne soubor `GAL`. Program se spouští pomocí příkazu `./GAL [PARAMS]`, kde parametry pro spuštění tohoto programu jsou následující.

- `-h --help`, program vypíše nápovědu a ukončí se.
- `-i, --input-graph=FILE`, kde `FILE` je název souboru obsahující vygenerovanou síť.
- `-s, --source=NODE`, kde `NODE` Udává index uzlu, který je zdrojem (indexováno od 0).
- `-t --sink=NODE`, kde `NODE` Udává index uzlu, který je spotřebičem (indexováno od 0).
- `-c --count=COUNT`, kde `COUNT` udává kolikrát se spustí výpočet nad danou sítí.
- `-n --num-threads=NUM`, kde `NUM` udává počet vláken pro paralelní algoritmy.
- `-o --csv-output`, výstup bude ve formátu pro vytvoření grafů.
- `-a --e-k-matrix`, výpočet se provede pomocí algoritmu Edomd Karp s matící sousednosti.
- `-b --e-k-list`, výpočet se provede pomocí algoritmu Edmond Karp se seznamem sousednosti.
- `-d --f-f-matrix`, výpočet se provede pomocí algoritmu `Ford Fulkerson` s maticí sousednosti.
- `-e --f-f-matrix-p`, výpočet se provede pomocí par. algoritmu `Ford Fulkerson` s maticí sousednosti.
- `-f --f-f-list` , výpočet se provede pomocí algoritmu `Ford Fulkerson` se seznamem sousednosti.
- `-g --f-f-list-p` , výpočet se provede pomocí par. algoritmu `Ford Fulkerson` se seznamem sousednosti.

7.3 Program pro generování sítí a vykreslení grafů

Programy pro generování sítí se nachází ve složce `network_generator/gen_graphs.py` a `network_generator/gen_graphs_mean_degree.py`. Tyto programy stačí spustit `shell` skriptem v kořenové složce s názvem `gen_graphs.sh` a sítě budou po vygenerování uloženy do složek `network_generator/graphs` a `network_generator/graphs_mean_degree`. Po vygenerování výsledků jednotlivých algoritmů nad všemi sítěmi, je možné spustit skript v kořenové složce s názvem `plot_graphs.sh`, který zpracuje výstupy jednotlivých algoritmů a vytvoří grafy, které můžeme vidět v kapitole 8.

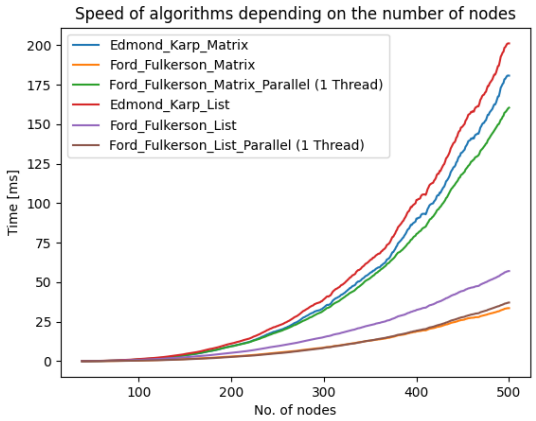
8 Vyhodnocení algoritmů

Všechny sekvenční i paralelní algoritmy byly vyhodnoceny na sítích, které jsou popsány v sekci 6 na superpočítači `Karolina`. `Karolina` je superpočítač v Ostravě, který má následující specifikace:

- Procesor: 2 x AMD Zen 2 EPYC™ 7H12, 2.6 GHz
- Paměť: 256 GB

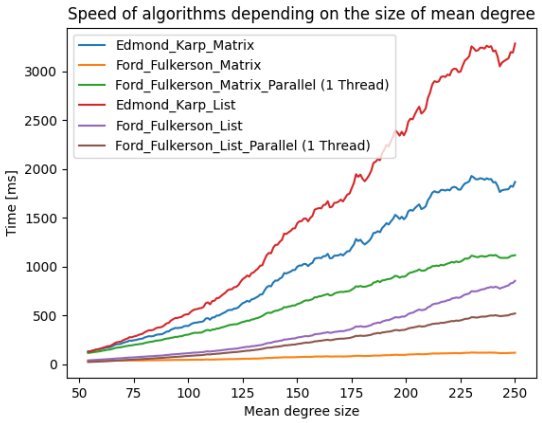
Algoritmy tedy byly vyhodnoceny na sítích o velikostech od 10 do 500 uzlů. Následně byly vyhodnoceny i na sítích, které všechny obsahovaly 500 uzlů, ale měly různou velikost hustoty. Byl porovnán i výkon sekvenčních algoritmu s paralelními, které byly spuštěny pouze na jednom vláknu. Paralelní algoritmy byly spuštěny s počtem vláken 1, 2, 4, 8, 16, 32, 64 v případě vyhodnocení sítí s různým počtem uzlů viz. 5 a v případě sítí, které měly různou velikost hustoty byly paralelní algoritmy spuštěny i s počtem vláken 128 viz. 6.

Na obrázku 3 můžeme vidět vyhodnocení všech sekvenčních algoritmů, které jsou Edmond Karp s použitím matice, Edmond Karp s použitím seznamu sousednosti, sekvenční i paralelní `Ford Fulkerson` s použitím matice, sekvenční i paralelní `Ford Fulkerson` s použitím seznamu sousednosti na různých velikostech sítě. Z grafu můžeme vyčíst, že algoritmus EK je méně efektivní než `FF`, což dává smysl, protože EK při úpravě reziduální sítě prochází nalezenou cestu dvakrát. Paralelní algoritmus `FFM` má horší výsledky, jelikož dělá spoustu nadbytečných výpočtů. U pomalejších algoritmů lze pozorovat složitost, která odpovídá teoretické složitosti.



Obrázek 3: Vyhodnocení sekvenčních a paralelních algoritmů (1 vlákno)

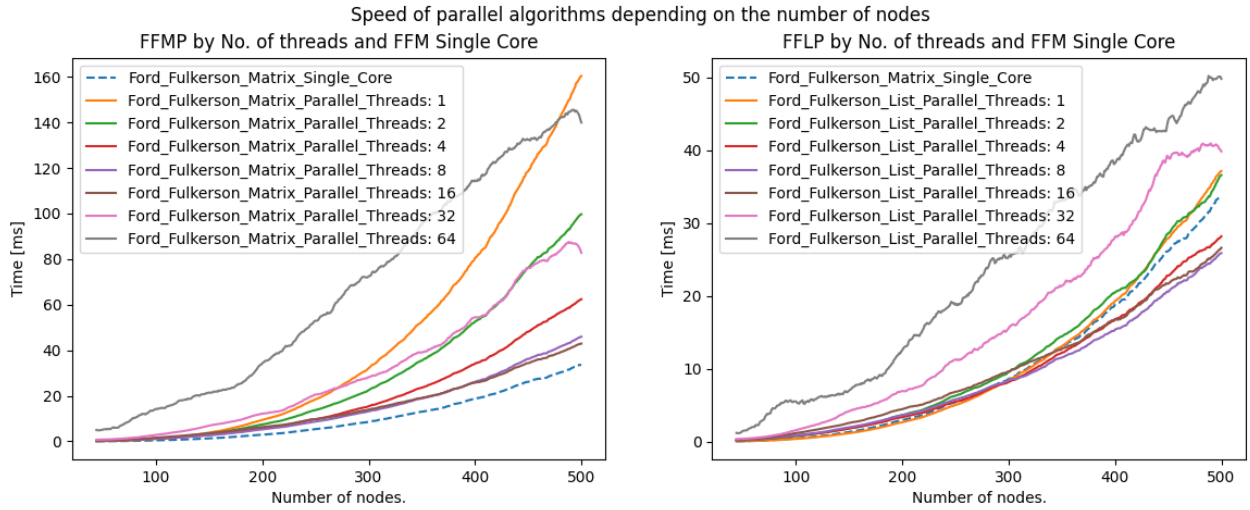
Na obrázku 4 můžeme vidět všechny algoritmy popsány výše, které jsou vyhodnoceny na síti o velikosti 500 uzlů s různou hustotou sítě. Na tomto grafu má nejlepší výsledek algoritmus `FFM`, předpokládáme, že je to právě kvůli tomu, že se jedná o přístup do matice a nemá overhead, i když podle teoretické složitosti by měl být `FFM` o dost pomalejší.



Obrázek 4: Vyhodnocení sekvenčních a paralelních algoritmů (1 vlákno)

Na obrázku 5 můžeme na levém grafu vidět paralelní algortimus `Ford Fulkerson` Matrix v sekvenční a paralelní verzi, která je spuštěna s různým počtem vláken. Z grafu je možno vyčíst, že nejrychlejší implementace je verze sekvenční a to z toho důvodu, že paralelní verze počítá spoustu nadbytečných výpočtů, které tak v rychlosti nepomáhají.

Na pravém grafu je vyhodnocen paralelní algortimus `Ford Fulkerson` se seznamem sousedů, zde můžeme vidět, že při použití počtu vláken 4, 8 a 16 je tato paralelní implementace rychlejší než sekvenční.

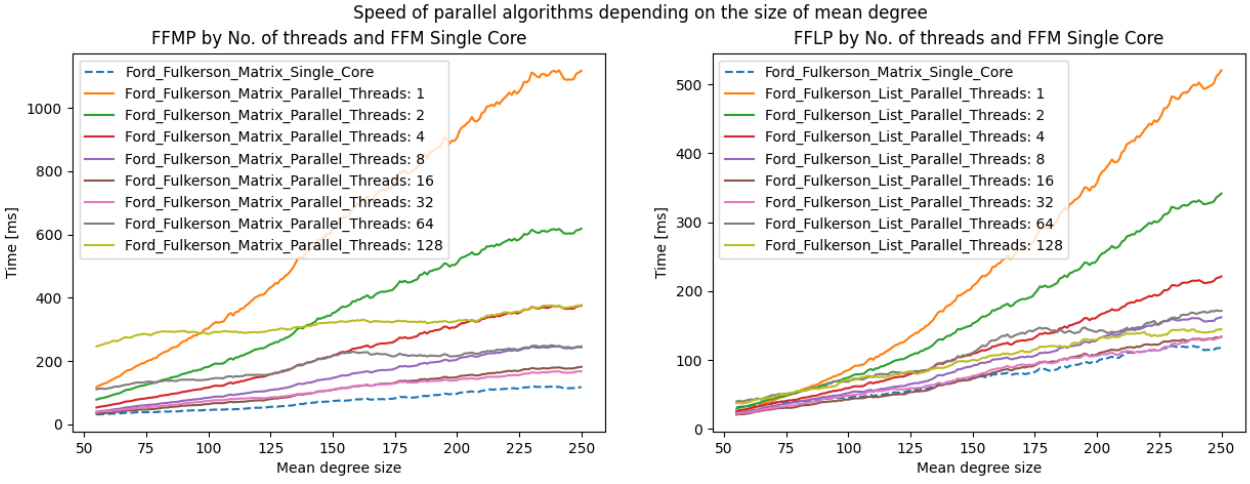


Obrázek 5: Rychlost paralelních algoritmů v závislosti na počtu uzlů

Na posledním obrázku 6 je vyhodnocení obou paralelních algoritmů s různým počtem vláken na různé velikosti hustoty sítě a toto vyhodnocení je porovnáno s nejrychlejší sekvenční implementací, která je `Ford Fulkerson` s použitím matice.

Zde opět na levém grafu můžeme vidět, že sekvenční implementace `FFM` je rychlejší než její paralelní verze, a to i přesto, že zde používáme vysoký počet vláken.

Na pravém grafu je vyhodnocení paralelní implementace `Ford Fulkerson` s použitím seznamu sousednosti, která je v její nejlepší verzi stejně rychlá jako sekvenční `FFM`. Paralelní verze není rychlejší, protože rychlost z teoretického hlediska závisí na počtu hran, která zde roste.



Obrázek 6: Rychlost paralelních algoritmů v závislosti na hustotě sítě

9 Závěr

Podařilo se implementovat dva sekvenční a jeden paralelní algoritmus pro výpočet maximálního toku v síti, pro tyto algoritmy bylo vygenerováno dostatečný počet sítí pro porovnání rychlosti mezi těmito algoritmy. Toto vyhodnocení se provedlo na více jádrovém superpočítačí `Karolina` a výsledky byly zdokumentovány. Tyto výsledky jsou zhodnoceny v kapitole č. 8. Při sekvenčních implementacích výsledky nedopadly podle očekávání s ohledem na jejich složitost, a to z toho důvodu, že algoritmus `Ford Fulkerson` s použitím matice sousednosti byl rychlejší než s použitím seznamu sousednosti, předpokládáme, že je to kvůli tomu, že se přistupuje do matice a kvůli tomu není overhead, co ovšem dopadlo podle očekávání je algoritmus Edmond Karp, který je pomalejší než alg. `Ford Fulkerson`, což vyplývá z toho, že algoritmus `FF` si hned značí o kolik se zvýší daný tok a tím pádem při úpravě reziduální sítě nemusí procházet nalezenou cestu dvakrát. Na grafu 3 lze vidět, že, závislost času na velikosti grafu odpovídá jejich teoretické složitosti. U paralelní implementace jsme ověřili, že paralelizovat algoritmus `Ford Fulkerson` s použitím matice sousednosti není efektivnější, a to z toho důvodu, že počítá spoustu nadbytečných výpočtu, avšak u verze se seznamem sousednosti se nám pro počet vláken 4, 8, 16 podařilo tento algoritmus zefektivnit, jak lze vidět na obrázku 5. Tyto paralelní algoritmy jsou pro více než 16 vláken neefektivní, jelikož paralelní úloha je zde malá a spouštění vláken má velký overhead, a proto zde začíná rychlost klesat. Pro jedno vlákno jsou paralelní algoritmy také neefektivní, protože pro každý krok BFS procházejí všechny hrany. Všechny vygenerované sítě, vstupní a výstupní soubory je možno zveřejnit.

Reference

[1] Baumstark, N.; Blelloch, G.; Shun, J.: Efficient Implementation of a Synchronous Parallel Push-Relabel Algorithm. 2015, doi:10.48550/ARXIV.1507.01926.
URL `https://arxiv.org/abs/1507.01926`

[2] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; aj.: *Introduction to Algorithms, Third Edition*. The MIT Press, tøetí vydání, 2009, ISBN 0262033844.

[3] Jiang, Z.; Hu, X.; Gao, S.: A Parallel Ford-Fulkerson Algorithm For Maximum Flow Problem. 2013.