Vysoké učení technické v Brně
Fakulta informačních technologií

Paralelizace - Algoritmus pro určení hranové konektivity grafu

Jiří Hon (xhonji01)
Martin Riša (xrisam00)

13. decembra 2014

Obsah

1 Úvod

Projekt sa venuje algoritmu [1] na určovanie hranovej konektivity v sekvenčnej a paralelizovanej variante.

2 Teoretický úvod

2.1 Sekvenčny algoritmus od Pánov Hiroshi Nagamochi a Toshihide Ibaraki

Nasledujúca sekcia čerpá z publikácie [1].

Algoritmus v porovnaní s ostatnými algoritmami pristupuje rozdielne k určovaniu hranovej konektivity G. Algoritmus analyzuje všetky vhodne vygenerované podgrafy G, ktoré vzniknú opakovanou konktrakciou dvoch uzlov, ktoré spájajú vybratú hranu. Výber hrany ku konktrakcií sa určuje procedúrou 1.

Algoritmus 1 Forest(G, n_{random},$\delta(G)$)

```
Label all nodes and all edges as unscanned
while exists unnscanned node do
  pick unscanned node n with largest r
  i = 1
  for all nodes t adjacent to n by unnscanned edge e do
    E_i = E_i ∪ e
    r_t = r_t + c(e)
    mark e scanned
    i = i + 1
  end for
  mark n scanned
end while
return an arbitrary edge from E_δ(G)
```

Táto procedúra má na vstupu G a náhodne vybratý uzol n_{start}. Postupne spracuje uzly, ktorým priraďuje hodnotu r. Spracovanie hrany uzlu spočívá vo inkrementácií hodnoty r o hodnotu kapacity hrany $c(e)$ a v zahrnutí hrany e do množiny E_i, pričom i značí poradie v akom bola hrana spracovaná v danom uzle.

Hrana vybratá ku kontrakcií je následne definovaná ako:

$$e_{contract} \in E_{\delta(G)} \quad (1)$$

Pričom $E_{\delta(G)} = E_i$ kde $i = \delta(G)$ a $\delta(G)$ je hodnota minimálnej kardinality spomedzi všetkých uzlov v G.

Samotný algoritmus zisťovania hranovej konnektivity G je založený na opakovanom volaní procedúry Forest a postupnej kontrakcií bodov na základe navrátenej hrany. Postup volania procedúry Forest a spôsob zisťovania hranovej konektivity je naznačený v procedúre 2.

Algoritmus 2 EdgeConnectivity(G)

$G^{'} = G$, $k = \delta(G)$, pick a random node n_{random}
while $|V^{'}| > 2$, $G^{'} = (V^{'}, E^{'})$ **do**
  call Forest(G, n_{random},k)
  let e be edge returned by Forest(G, n_{random})
  let $G^{'}$ be graph created by conraction by edge e
  analyze $\delta(G^{'})$ and set $k = min(k, \delta(G^{'}))$
end while
return k

Hľadná hodnota hranovej konektivity $\lambda(G)$ je naslédne definovaná ako:

$$\lambda(G) = min(\delta(G^1), \delta(G^2), \ldots, \delta(G^{|V|-1})) \quad (2)$$

Pričom G^i značí podgraf, ktorý vznikne kontrakciou G^{i-1} na základe vybranej hrany e.

Uvedený algoritmus sa dá upraviť k získaniu minimálneho rezu, resp. maximálneho toku v G. V projekte sa zaoberáme len problematikou zisťovania hodnoty hranovej konektivity G.

2.1.1 Teoretická zložitosť algoritmu

Procedúra Forest je volaná práve $|V| - 1$ krát. Časovú zložitosť kontrakcie dvoch bodov predpokladáme za konštantnú a zanedbateľne malú. V procedúre Forest sa prejde každým vrcholom práve jeden krát. Výber nasledujúceho uzlu k spracovaniu je realizovný prioritnou haldou. Za predpokladu použitia fibbonaciho haldy je inkrementácia hodnoty r_i realizovaná v priemere s konštantou časovou zložitosťou $\Theta(1)$. V každom kroku procedúry Forest dochádza k zmenšovaniu vytvorenej haldy za použitia operácie s priemernou časovou zložitosťou $\Theta(log(n))$. Taktiež v každom volaní procedúry Forest sa analyzuje graf menší oproti predošlému o jeden vrchol a minimálne jednu hranu. Najhoršia teoretická časová zložitosť je $O(|V||E|)$ [1, str. 59].

2.2 Paralény algoritmus na zisťovanie hranovej konektivity

Možnosti paralelizovania predošlého algoritmu nie sú zrejmé. Problémom je najmä procedúra Forest, v ktorej sa 'spracujú' iteratívne všetky uzly práve jeden-krát.

V prípade paralelizovania tohoto 'spracovania' uzlov sa k spracovaniu môzu vybrať dva susedné uzly, čo by viedlo k tomu, že ich spoločné hrany by patrili do rozdielnych množín E_i. Čo by potenciálne mohlo viesť k zostaveniu prázdnej množiny hrán $E_{\delta(G')}$. Dôkaz predošlého tvrdenia je uvedený na príklade v prílohe 1.

V prípade, ak by sa k paralélnemu spracovaniu uzlov vyberali také n-tice uzlov ktoré nie sú susedné. Znamenalo by to isté zväčšenie zložitosti algoritmu a taktiež nie je zaručená neprázdnosť množiny $E_{\delta(G')}$. Dôkaz je uvedený na príklade v prílohe 2.

Jedinou objavenou možnosťou paralelizovania uvedeného algoritmu, je paralelizovanie samotného volania procedúry Forest. Hlavné vlákno v prvom kroku spustí vlákno pre sekvenčný vypočet procedúry Forest a dopredne predikuje, aká hrana bude týmto volaním navrátená. Postupne spúšťa vlákna, ktorým na vstup generuje grafy G^{1*}. Pričom grafy G^{1*} sú výsledkom predikcie kontrakcie hrán. V momente, keď skončí vlákno pre sekvenčný výpočet procedúry Forest hlavné vlákno analyzuje, či už nepredikoval danú kontrakciu. Následne ukončí beh všetkých ostatných vlákien a v prípade, ak sa mu podarilo predikovať navrátenu kontrakciu označí toto vlákno za sekvenčný výpočet pre nasledujúcu iteráciu.

Možnosti predikcie kontrakcie sú rôzne a ich popis presahuje možnosti tejto práce.

Navrhované riešenie paralelizácie je z hľadiska implementácie príliš náročné a jeho prínos z hľadiska zložitosti algoritmu nejasný. Rozhodne by sa navrhované riešenie paralelizácie malo experimentálne otestovať. Bohužiaľ rozsah tohoho predmetu nám podrobnejšiu analýzu problematiky neposkytuje.

Iné možnosti paralelizácie tohoto algoritmu sa nám nepodarilo objaviť. Z tohoto dôvodu sme sa rozhodli upraviť si zadanie projektu a paralelizovať ľubovoľnú, paralizovateľnú, metódu na zisťovanie hranovej konnektivity.

Podľa MIN-CUT MAX-FLOW [4] teorému k zisteniu hranovej konektivity G postačuje výpočítať minimálnu hodnotu všetkých maximálných tokov z ľubovoľného uzlu do všetkých ostatných uzlov.

Zvolený paralélny algoritmus bol Edmonds Karp [2] na zisťovanie maximálneho toku v siety.

3 Návrh

Graf by mal byť reprezentovaný ako zoznam susedov(ang. adjacency list).

3.1 Sekvenčný algoritmus

K výberu nasledujúceho nespracovaného uzlu s najväčšou hodnotou r v procedúre Forest je potrebná binárna prioritná halda.

4 Implementácia

Algoritmy bolo zvolené implementovať v jazyku C++ za použitia knižnice OGDF. Pre reprezentáciu neorientovaných multigrafov bol zvolený vnútorný formát OGDF. Formát vstupných grafových súborov predpokladáme GML.

4.1 Sekvenčný algoritmus

Použitá binárna halda je *ogdf::BinaryHeap*. K priraďovaniu hrán do jednotlivých množín E_i je použitá *ogdf::EdgeArray*.

4.2 Paralélny algoritmus

K samotnému paralelizovaniu je použité rozhranie nástroja OpenMP vo verzií 4, ktorá je štandardnou súčasťou použíteho prekladača gcc.

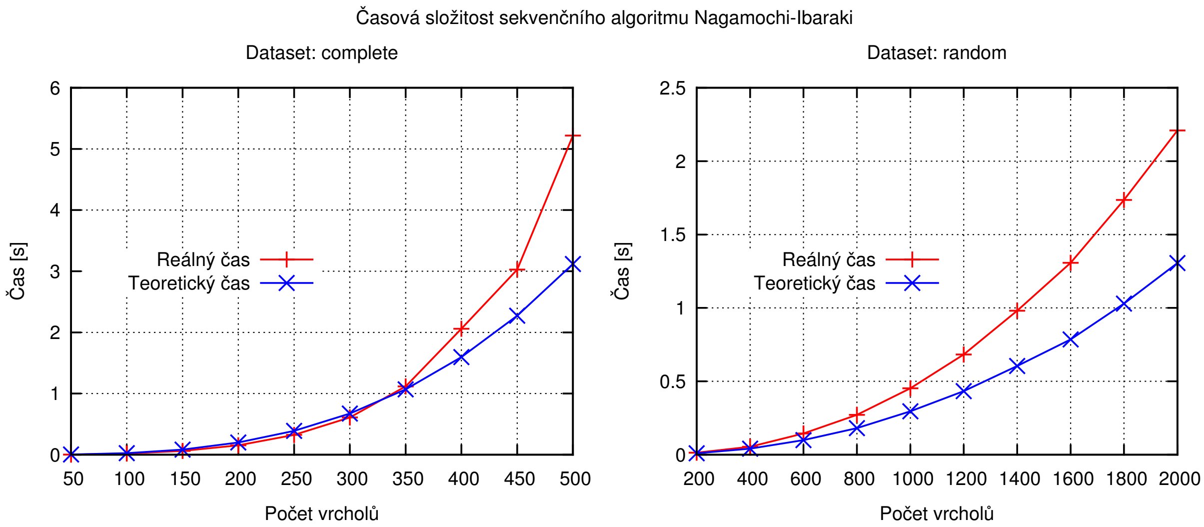
Nakoľko testovanie prebiehalo na relatívne malých grafoch, bolo stanovené jedným vláknom vykonávať desať rôznych výpočtov maximálných tokov medzi dvoma rôznymi bodmi v grafe.

5 Testovanie

Testovanie implementovaných algoritmov prebiehalo na stroji edesign1.fit.vutbr.cz s procesorom Intel Xeon X5650 2.67 GHz so šiestimi jadrami. Vstuné grafy boli zvolené na úplne grafy o N uzloch a náhodne generované grafy o N uzloch a s pravdepodobnosťou p výskytu hrany medzi dvoma uzlamy, pričom p je zvolené podľa modelu [3], tak aby boly vygenerované grafy celkom určite spojité.

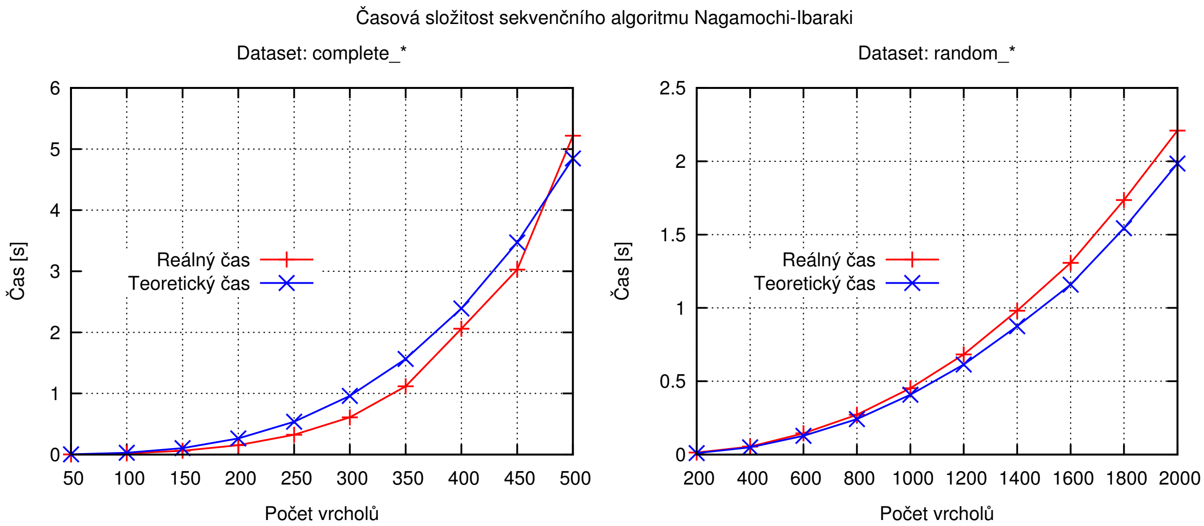
Všetky vstupné grafy boli otestované desať-krát a bol urobený priemer z nameraných časov.

5.1 Sekvenčný algoritmus

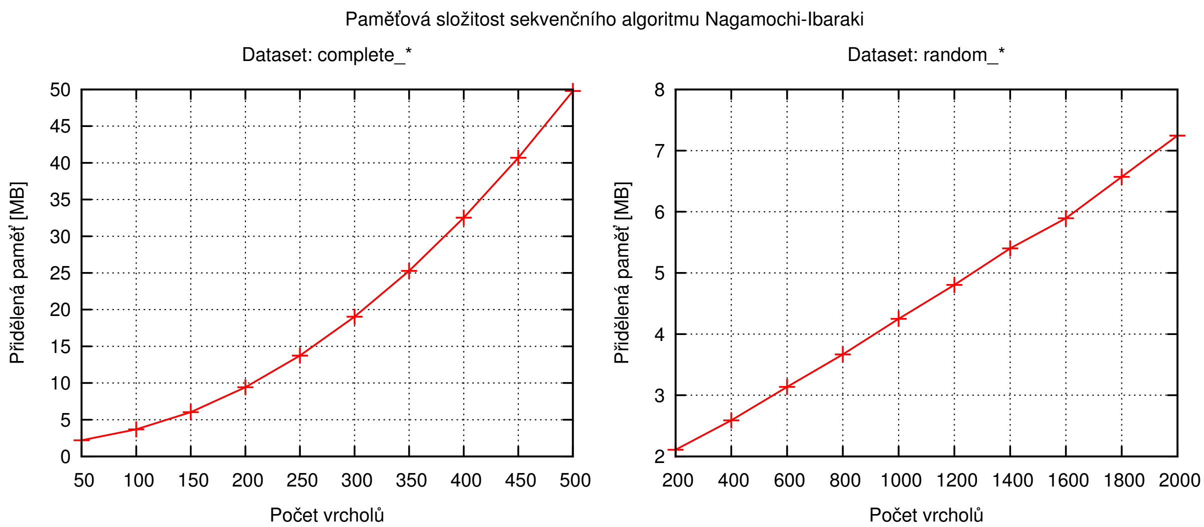


V predošlom grafe predpokladáme teoretickú zložitosť $O(|V||E|)$. Je zrejmé, že implementovaný algortimus pri väčších počtoch uzlov je pomašlí ako teoretická zložitosť, dôvodom tohoto spomalenia je pravdepobne použitie binárnej haldy namiesto odporúčanej Fibbonaciho haldy. V implementovanom algoritme sa prírastok zložitosti algoritmu použitím binárnej haldy zväčší v operácií vložení a zmenšení hodnoty prvku v halde a to v oboch prípadoch z priemerne konštantnej časovej zložitosti na $O(log(|V|))$. Z tohoto dôvodu odhadujeme priemernú časovú zložitosť implementovaného algoritmu na $O(|V||E|log(|V|))$.

Nasledujúci graf predpokladá teoretickú časovú zložitosť implementovaného algoritmu $O(|V||E|log(|V|))$.



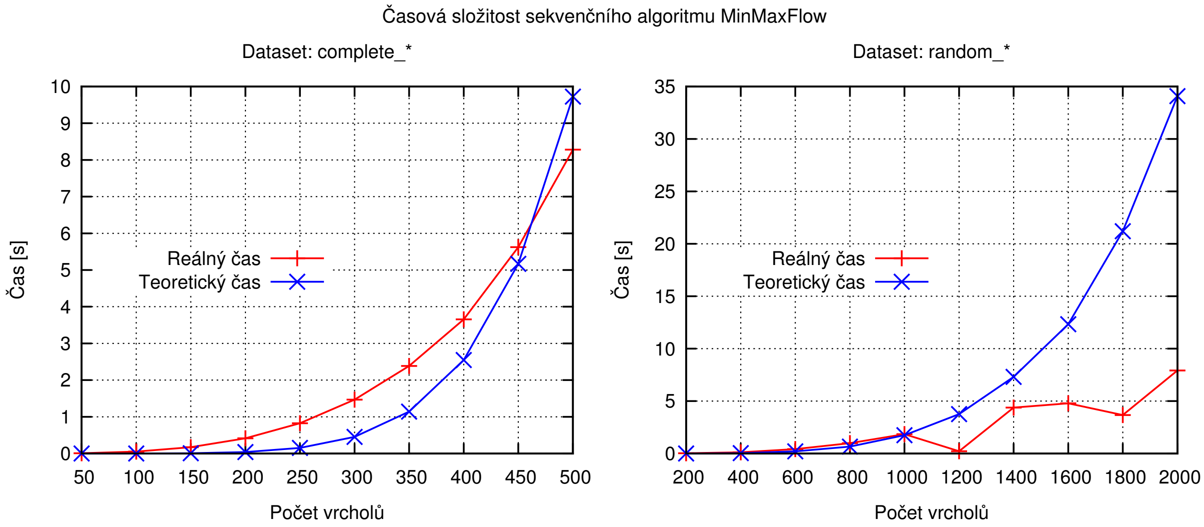
Nasledújci graf znázorňuje pameťovú náročnosť implementovaného algoritmu, ktorá je rovná $|V| + |E|$.



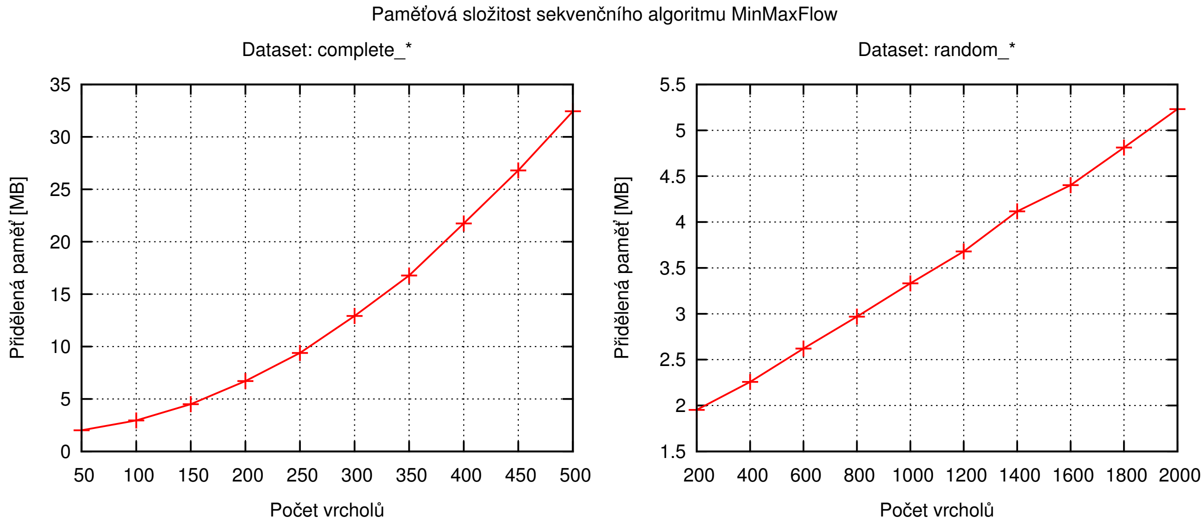
5.2 Paralélny algoritmus

Je zrejmé, že paralélny algoritmus závisí od počtu jadier, nakoľko je nutné volať procedúru Edmonds-Karp práce $|V| - 1$ krát a jednotlivé volania sú navzájom nezávislé.

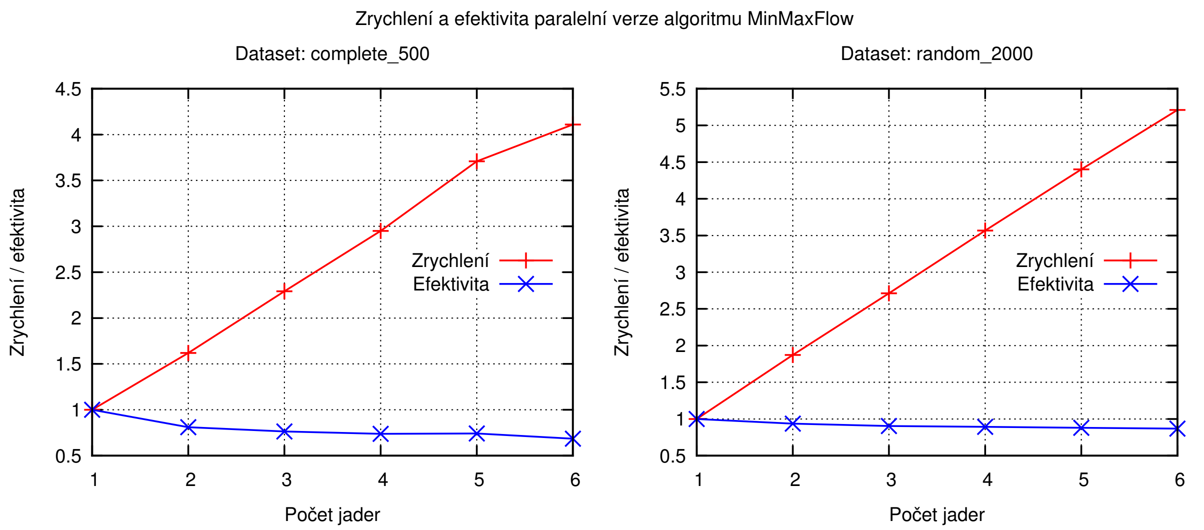
Procedúra Edmonds-Karp má priemernú časovú zložitosť $O(|V||E|^2)$. V prípade, keď máme k dispozícií aspoň $|V| - 1$ vlákien je celková zložitosť imlementovaného algoritmu $O(|V||E|^2)$. Je zrejmé že priemerná časová zložitosť algoritmu je $O(\frac{|V|^2|E|^2}{n})$, kde n je počet vlákien.



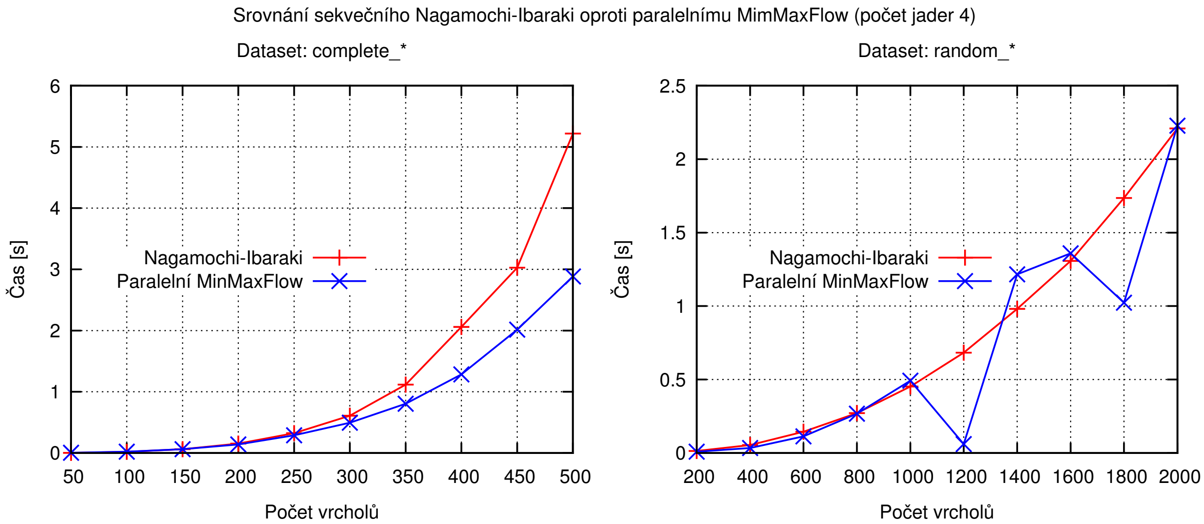
V predošlom grafe je na úplných grafoch vidieť, že implementácia Procedúra Edmonds-Karp má v priemere časovú zložitosť $O(|V||E|^2)$.



Namerá priestorová zložitosť je $|V| + |E|$. Nasledujúci graf znázorňuje efektivitu algoritmu.



5.3 Porovnanie algoritmov



Z Predošlého grafu možno usúdiť, že pre dané testovacie podmienky je výhodnejšie použiť paralélny algoritmus k zisťovaniu hranovej konektivity. Pre nižší počet vlákien je pre dané testovacie podmienky, vyhodnejšie použiť sekvenčný algoritmus.

5.4 Zdôvodnenie nameraných zložitostí

Nameraná časová zložitosť sekvenčného algoritmu je $O(|V||E|log(|V|))$. Táto zložitosť sa líši od $O(|V||E|)$ a to z dôvodu použitia binárnej haldy namiesto odporúčanej Fibbonaciho haldy.

Nameraná časová zložitosť paraléleho algoritmu je $\frac{|V|^2|E|^2}{n}$.

Sekvenčný algoritmus má v porovnaní s paralélnym väčšie pamäťové nároky, dôvodom je hlavne binárna fronta a zoznam hrán priradených do množín E_i.

6 Záver

V projekte sa prišlo na to, že algoritmus [1] pravdepodobne nemá priamočiaru paralélnu variantu.

Ďalej sa v projekte testovaním overilo, že imlpementované algoritmy implementujú navrhnuté algoritmy.

Testovaním sa prišlo na to, že sekvenčný algoritmus môže byť vhodný použiť na zisťovanie hranovej konektivity v prípade, ak nemáme k dispozícií dostatočný počet vlákien. Testovaním sa odhaduje táto minimálna hodnota jadier

Literatúra

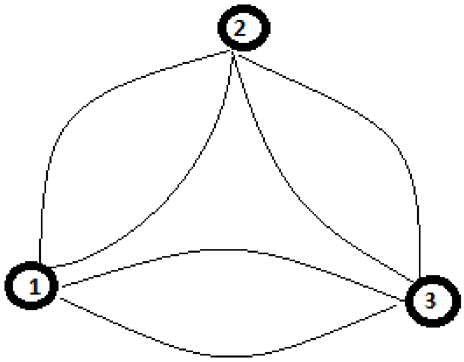
[1] H. Nagamochi, T. Ibaraki. *Computing edge-connectivity in multigraphs and capacitated graphs*, volume 5. Siam J. Discrete Math, 1992.

[2] Wikipedia. Edmonds karp algorithm — Wikipedia, the free encyclopedia, 2014. [Online; 10-december-2014].

[3] Wikipedia. Erdős–rényi_model — Wikipedia, the free encyclopedia, 2014. [Online; 10-december-2014].

[4] Wikipedia. Max-flow min-cut theorem — Wikipedia, the free encyclopedia, 2014. [Online; 10-december-2014].

A Príklad paralélneho spracovania susedných uzlov procedúrou Forest

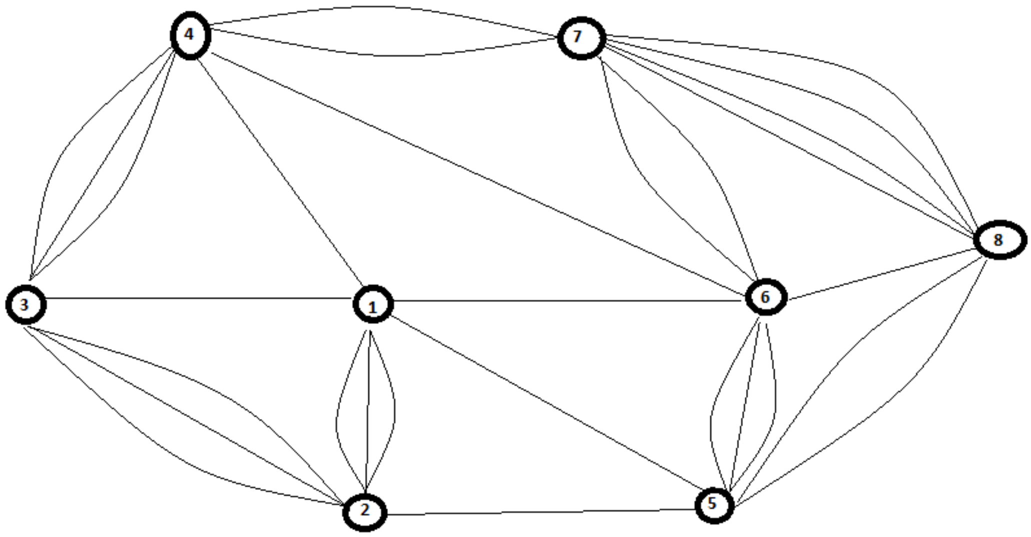


V nasledujúcom príklade predpokladáme, že procedúra Forest paralélne spracuváva dva uzly. Jeden z možných priebehov vlákien je taký, že jedno vlákno spracuje uzol 1 a druhé uzol 2. V aktuálnom stave sú hodnoty $r_i = 0$ a množiny $E_j = \emptyset$, pričom $i = 1 \ldots |V|$ a $j = 1 \ldots \delta(G)$.

Oboje vlákna sa musia synchronizovať v označovaní spoločných hrán a vo zvyšovaní hodnoty r_3. V prípade, ak jedno vlákno označí jednu spoločnú hranu hodnotou 1 a druhé vlákno označí ďaľšiu spoločnú hranu hodnotou 1 potom, ale $E_4 = \emptyset$ a nieje možné vybrať hranu ku kontrakcií.(Pozn. Nenechajte sa zmiasť tým, že sa v príklade jedná o uplný graf, jedná sa len o ilustráciu toho, že daný postup paralelizácie nieje valídny, existuje množsto takýchto prípadov, ktoré vedú k prázdnej množine $E_{\delta(G')}$.)

Riešením je zamedzenie paralélnemu číslovaniu spoločných hrán.

B Príklad paralélneho spracovania nesusediacich uzlov procedúrou Forest



V nasledujúcom príklade predpokládáme paralélne spracovanie uzlov 1 a 8, pričom $\delta(G) = 7$. A predpokladáme hodnoty $r_i = 0$ a množiny $E_j = \emptyset$, pričom $i = 1 \dots |V|$ a $j = 1 \dots \delta(G)$. V tomto prípade zostane množina $E_7 = \emptyset$ a teda nie je možné vybrať hranu ku kontrakcií.

Riešenie tohoto problému sa nám nepodarilo objaviť. Z predošlého usudzujeme, že zvolený algoritmus pravdepodobne nemá priamočiaru paralélnu variantu.