

Komparativna analiza BFS, WCC i PageRank algoritama u sekvencijalnoj i paralelnoj verziji

1. UVOD

Fast Transit Network upravlja kompleksnom mrežom čvorova i veza koja obuhvata stanice, kontrolne sisteme i komunikacione kanale. U slučaju incidenta, kao što su prekid veza, kvarovi čvorova ili degradacija servisa, operaterima je neophodna brza i precizna analiza stanja mreže. Sistem mora biti u stanju da odgovori na tri kritična pitanja: koji čvorovi su dostižni iz datog izvora i na kojoj udaljenosti, koji delovi mreže su međusobno povezani ako se zanemari smer veza, i koji čvorovi su najvažniji za prioritizaciju monitoringa.

Tradicionalni pristupi analizi grafova često nisu dovoljno efikasni za grafove velikih razmera kakvi se javljaju u modernim mrežnim infrastrukturama. Sa povećanjem broja čvorova i veza, potreba za paralelnom obradom postaje imperativ. Rust programski jezik, sa svojim karakteristikama bezbednosti memorije i performansi na nivou C/C++, predstavlja idealnu platformu za razvoj ovakvih sistema.

Cilj ovog projekta je razvoj sveobuhvatnog sistema za grafovsku analitiku koji koristi efikasne strukture podataka i paralelne algoritme kako bi obezbedio brze odgovore na upite nad velikim grafovima. Sistem je dizajniran sa naglaskom na tačnost, performanse i skalabilnost, pri čemu su sve implementacije temeljno testirane i verifikovane.

2. DIZAJN REPREZENTACIJE GRAFA

Izbor odgovarajuće strukture podataka za reprezentaciju grafa predstavlja fundamentalni aspekt sistema koji direktno utiče na performanse svih algoritama. U ovom radu je izabrana CSR (Compressed Sparse Row) reprezentacija, koja se pokazala kao idealna za grafove velikih razmera.

2.1 CSR format

CSR format kompresuje graf u dva glavna niza: niz offseta i niz ivica. Struktura CSRGraph u Rust jeziku sastoji se od četiri polja: `num_vertices` (broj čvorova), `num_edges` (broj ivica), `offsets` (niz offseta), i `edges` (niz ivica).

Niz `offsets` dužine $V+1$ (gde je V broj čvorova) sadrži indekse koji pokazuju gde u nizu `edges` počinju susedi svakog čvora. Niz `edges` dužine E (gde je E broj ivica) sadrži stvarne identifikatore susednih čvorova. Za čvor v , njegovi susedi se nalaze u opsegu `edges[offsets[v]..offsets[v+1]]`.

Ovakva reprezentacija omogućava pristup svim susedima čvora u konstantnom vremenu $O(1)$ za dobijanje početnog pokazivača, nakon čega sledi sekvencijalni pristup susedima. Budući da su susedi smešteni kontinualno u memoriji, ova struktura je izuzetno cache-friendly, što je ključno za performanse modernih procesora sa hijerarhiom keš memorije.

Dodatna prednost CSR formata je kompaktnost. Za graf sa V čvorova i E ivica, memorijska potrošnja iznosi približno $8(V+1) + 8E$ bajtova, što je značajno efikasnije od matrica susedstva čija je složenost $O(V^2)$. Ova karakteristika je posebno važna kod retko povezanih grafova koji su česti u mrežnim aplikacijama.

2.2 Bidirekciona CSR struktura

Za algoritme WCC i PageRank neophodna je informacija i o ulaznim i o izlaznim vezama čvorova. Usmereni graf prirodno sadrži informacije o izlaznim vezama, ali pristup ulaznim vezama zahteva dodatnu strukturu. Implementirana je bidirekciona CSR struktura koja održava dva nezavisna CSR grafa: *forward* graf koji predstavlja originalne ivice, i *reverse* graf koji sadrži transponovane ivice.

Konstrukcija ovih struktura vrši se tokom učitavanja grafa iz fajla, pri čemu se za svaku ivicu (u, v) u *forward* grafu dodaje ivica (v, u) u *reverse* graf. Ovakav pristup omogućava efikasan pristup kako odlaznim, tako i dolaznim susedima svakog čvora, što je esencijalno za WCC koji tretira graf kao neusmeren, i za PageRank koji mora iterirati preko prethodnika čvorova.

Dodatni memorijski trošak bidirekcionе strukture iznosi dva puta više od jednostavne CSR strukture, ali je ovaj trošak opravdan značajnim ubrzanjem algoritama koji inače ne bi imali efikasan način pristupa ulaznim vezama.

2.3 Procena memorijske potrošnje

Teorijska memorijska složenost za CSR graf sa V čvorova i E ivica može se precizno kvantifikovati. Niz offsets zahteva $(V+1) \times 8$ bajtova, dok niz edges zahteva $E \times 8$ bajtova, što daje ukupno $8V + 8 + 8E$ bajtova. Bidirekciona struktura udvostručava ovaj iznos na $16V + 16 + 16E$ bajtova.

Za konkretne primere, graf sa 50,000 čvorova i 60,000 ivica zahteva približno 1.2 MB za jednostavnu CSR strukturu i 2.4 MB za bidirekcionu strukturu. Eksperimentalna merenja pokazuju da je stvarna potrošnja u skladu sa teorijskim procenama, sa malim dodatnim overhead-om za metadata strukture i alokacije.

3. IMPLEMENTIRANI ALGORITMI

Sva tri algoritma implementirana su u dve varijante: sekvencijalna varijanta koja služi kao referentna implementacija za verifikaciju tačnosti, i paralelna varijanta optimizovana za maksimalne performanse na višejezgrenim procesorima.

3.1 BFS algoritam

Breadth-First Search algoritam predstavlja fundamentalni algoritam za pretragu grafova koji obilazi čvorove nivo po nivo, počevši od izvornog čvora. Cilj algoritma je određivanje najkraćeg rastojanja (u broju ivica) od izvornog čvora do svih ostalih dostižnih čvorova u grafu.

Sekvencijalna implementacija koristi standardni pristup sa *queue*-om. Algoritam inicijalizuje niz rastojanja sa vrednošću -1 za sve čvorove (označava nedostižnost), postavlja rastojanje izvornog čvora na 0, i dodaje ga u *queue*. U svakoj iteraciji, algoritam uklanja čvor sa početka *queue*-a, obilazi sve njegove susede, i za svaki neposećen susedni čvor postavlja rastojanje kao rastojanje trenutnog čvora uvećano za jedan, nakon čega ga dodaje u *queue*. Proces se nastavlja dok *queue* ne bude prazan.

Vremenska složenost sekvencijalnog BFS algoritma je $O(V + E)$ gde je V broj čvorova a E broj ivica, što je optimalno za ovaj problem. Prostorna složenost je takođe $O(V)$ za niz rastojanja i red.

Paralelna verzija koristi level-synchronous pristup. Umesto reda, algoritam održava frontijer čvorova na trenutnom nivou. U svakoj iteraciji, svi čvorovi u trenutnom frontijeru se obrađuju paralelno, pri čemu svaka nit obrađuje podskup čvorova. Susedni čvorovi koji se otkriju formiraju novi frontijer za sledeću iteraciju. Ključni izazov u paralelnoj implementaciji je thread-safety pri ažuriranju niza rastojanja, budući da više niti može istovremeno pokušati da postavi rastojanje istom čvoru. Ovo je rešeno korišćenjem atomskih operacija *compare_exchange* koja osigurava da samo jedna nit uspešno postavi rastojanje čvoru, dok ostale niti detektuju da je čvor već posećen i preskaču ga. Ovaj pristup eliminiše stetno preplitanje i garantuje korektnost algoritma.

3.2 WCC algoritam

Weakly Connected Components algoritam identifikuje slabo povezane komponente u usmerenom grafu. Slabo povezana komponenta definiše se kao maksimalan skup čvorova gde postoji put između bilo koja dva čvora ako se zanemari smer ivica. Praktično, algoritam tretira usmereni graf kao neusmereni.

Implementacija koristi BFS-baziran pristup. Algoritam inicijalizuje niz komponenti sa vrednošću koja označava da čvor nije posećen. Zatim iterira kroz sve čvorove, i za svaki neposećen čvor pokreće BFS koji dodeljuje isti identifikator komponente svim čvorovima dostižnim iz tog čvora (u neusmerenom smislu). Broj pokrenutih BFS pretraga odgovara broju komponenti u grafu.

Ključna razlika u odnosu na klasični BFS je što WCC mora da prati i ulazne i izlazne veze, što je omogućeno korišćenjem bidirekcionog CSR strukture. Metoda *all_neighbors* vraća iterator preko svih suseda (i *incoming* i *outgoing*), omogućavajući efikasnu iteraciju.

Paralelna verzija WCC algoritma dozvoljava da više niti istovremeno pokreće BFS sa različitih startnih čvorova. Svaka nit pokušava da zauzme neposećene čvorove korišćenjem atomskih operacija. Ako nit uspe da zauzme čvor, ona pokreće BFS iz tog čvora i dodeljuje komponentu svim dostižnim čvorovima. Ovaj pristup maksimalno iskorišćava paralelizam, posebno kod grafova sa više komponenti.

Važno je napomenuti da identifikatori komponenti mogu biti različiti između sekvencijalne i paralelne verzije (u sekvencijalnoj verziji komponente dobijaju uzastopne identifikatore, dok u paralelnoj zavisi od redosleda zauzimanja), ali grupacije čvorova su identične.

3.3 PageRank algoritam

PageRank algoritam računa numeričku vrednost koja predstavlja relativnu važnost svakog čvora u grafu. Originalno razvijen za rangiranje web stranica, algoritam se zasniva na ideji da je čvor važan ako mu pokazuju važni čvorovi. Matematički, PageRank vrednost čvora v definiše se kao: $PR(v) = (1-d)/N + d \times \sum (PR(u) / \text{out_degree}(u))$, gde je d damping factor (tipično 0.85), N ukupan broj čvorova, a suma se proteže preko svih čvorova u koji imaju izlaznu ivicu ka v .

Algoritam je iterativan i počinje sa inicijalnim vrednostima $PR(v) = 1/N$ za sve čvorove. U svakoj iteraciji, računa se nova vrednost PageRank-a za svaki čvor na osnovu trenutnih vrednosti njegovih prethodnika. Iteracije se nastavljaju dok razlika između uzastopnih iteracija ne padne ispod određenog praga tolerancije ϵ .

Implementacija koristi pull-based pristup gde svaki čvor povlači vrednosti od svojih prethodnika. Ovaj pristup je pogodan za paralelizaciju jer svaki čvor može nezavisno računati svoju novu vrednost. Sekvencijalna verzija jednostavno iterira kroz sve čvorove i ažurira njihove vrednosti.

Paralelna verzija koristi Rayon biblioteku za paralelnu iteraciju preko čvorova. Budući da svaki čvor samo čita vrednosti drugih čvorova (iz prethodne iteracije) i piše samo svoju vrednost, nema potrebe za atomskim operacijama tokom glavne faze računanja. Atomske operacije su potrebne samo za redukciju pri proveru konvergencije.

Posebna pažnja posvećena je tretmanu *dangling* čvorovima (čvorova bez izlaznih ivica). Kod takvih čvorova, njihova PageRank vrednost se ravnomerno distribuira svim čvorovima u grafu, što se implementira kao dodatni korak u svakoj iteraciji.

6. ANALIZA REZULTATA I DISKUSIJA

6.1 Poređenje sekvencijalnih i paralelnih verzija algoritama

Performanse sekvencijalnih i paralelnih verzija svih tri algoritma testirane su na dva grafa različitih veličina: mali graf sa 1,000 čvorova i veliki graf sa 5,000,000 čvorova. Paralelne verzije izvršavane su sa 16 niti. Rezultati su prikazani na Slici 1.

Rezultati na malom grafu

Na malom grafu, sekvenencijalne verzije su ubedljivo brže od paralelnih. Sva tri algoritma pokazuju ubrzanje manje od 1x, što znači da su paralelne verzije sporije. PageRank pokazuje najlošije rezultate sa ubrzanjem od samo 0.02x (preko 40 puta sporiji paralelno), dok BFS i WCC takođe pokazuju zanemarljive speedup-ove od 0.00x i 0.11x respektivno.

Ovi rezultati jasno demonstriraju fenomen overhead-a paralelizacije. Kod malih problema, fiksni troškovi kreiranja 16 niti, distribucije posla i atomskih sinhronizacionih operacija potpuno dominiraju nad stvarnim radom. Sa samo 1,000 čvorova, svaka nit dobija manje od 100 čvorova za obradu, što je daleko ispod praga gde paralelizacija postaje opravdana.

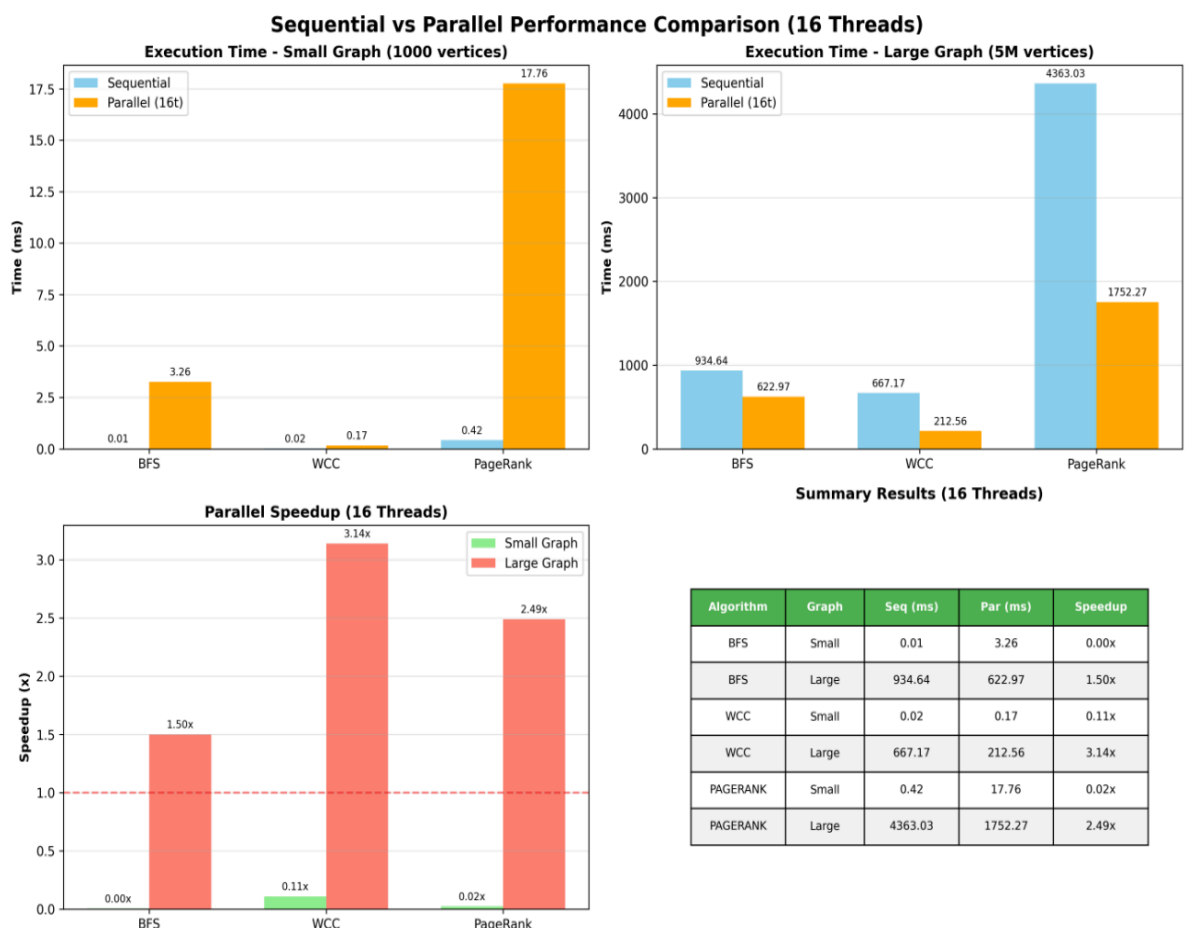
Rezultati na velikom grafu

Na velikom grafu sa 5,000,000 čvorova, paralelne verzije pokazuju značajno ubrzanje. WCC postiže najbolje rezultate sa ubrzanjem od 3.14x, PageRank dostiže 2.49x, dok BFS postiže 1.50x.

WCC pokazuje najbolju skalabilnost jer različite niti mogu nezavisno procesirati različite komponente grafa sa minimalnom međusobnom sinhronizacijom. Postignuto ubrzanje od 3.14x na 16 niti predstavlja efikasnost od približno 20%, što je solidan rezultat za grafovske algoritme gde memory bandwidth često postaje usko grlo.

PageRank sa speedup-om od 2.49x pokazuje solidne performanse, ali PageRank je memorijski zahtevan algoritam koji zahteva pristup rank vrednostima svih prethodnika svakog čvora, što generiše visok pritisak na memoriju. Dodavanje više niti nakon određene tačke ne donosi benefit jer sve niti konkurišu za isti resurs.

BFS postiže najslabije ubrzanje od 1.50x. Level-synchronous pristup zahteva sinhronizaciju nakon svakog nivoa grafa, što ograničava paralelizam. Atomske operacije koje se koriste za thread-safe pristup dodatno usporavaju izvršavanje zbog konkurencije između niti.



Slika 1. Poredjenje vremena izvršavanja sekvencijalnih i paralelnih verzija BFS, WCC i PageRank algoritama.

6.2 Skaliranje sa brojem niti

Analiza skalabilnosti paralelnih algoritama sa povećanjem broja niti pruža dublji uvid u karakteristike paralelizacije i identifikuje potencijalna uska grla. Testirane su konfiguracije sa 2, 4, 8, i 16 niti na oba grafa. Rezultati su prikazani na Slici 2, koja obuhvata vreme izvršavanja i ubrzanje za sve kombinacije algoritama i veličina grafova.

BFS skaliranje

BFS algoritam pokazuje značajne razlike u ponašanju između malog i velikog grafa. Na malom grafu od 1,000 čvorova, povećanje broja niti zapravo pogoršava performanse. Graf speedup analize pokazuje da sve paralelne konfiguracije imaju ubrzanje znatno ispod 1x (oko 0.01x-0.02x), što znači da su 50-100 puta sporije od sekvencijalne verzije.

Na velikom grafu od 5,000,000 čvorova, situacija je radikalno drugačija. Međutim, i ovde se uočava neočekivan trend. Konfiguracija sa 2 niti pokazuje dramatično povećanje vremena na 1250.48 ms (u odnosu na 905.62 ms sekvencijalno), što rezultuje ubrzanjem manjim od 1x. Tek sa 4 niti se postiže poboljšanje (831.51 ms, ubrzanje 1.09x), a trend se nastavlja sa 8 niti (644.36 ms, ubrzanje 1.41x) i 16 niti (613.94 ms, ubrzanje 1.48x).

Maksimalno postignuto ubrzanje od 1.48x sa 16 niti (efikasnost ~9%) daleko je od idealnog, što potvrđuje da je BFS inherentno težak problem za paralelizaciju zbog čestih sinhronizacionih tačaka i nepredvidivosti strukture frontijera između nivoa.

WCC skaliranje

WCC algoritam demonstrira najbolje skalabilne karakteristike od svih testiranih algoritama. Na malom grafu, overhead je prisutan ali manje izražen nego kod BFS-a. Vreme raste sa 0.02 ms (sekvencijalno) na 0.05 ms (2 niti), 0.05 ms (4 niti), 0.07 ms (8 niti), i 0.14 ms (16 niti). Iako su sve paralelne verzije sporije od sekvencijalne (speedup 0.38x-0.45x), razlike su manjih magnituda nego kod BFS-a.

Na velikom grafu, WCC pokazuje izuzetnu skalabilnost. Sekvencijalno vreme od 757.72 ms progresivno se smanjuje sa povećanjem broja niti: 468.02 ms (2 niti, ubrzanje 1.61x), 287.66 ms (4 niti, ubrzanje 2.63x), 224.40 ms (8 niti, ubrzanje 3.37x), i 211.36 ms (16 niti, ubrzanje 3.58x).

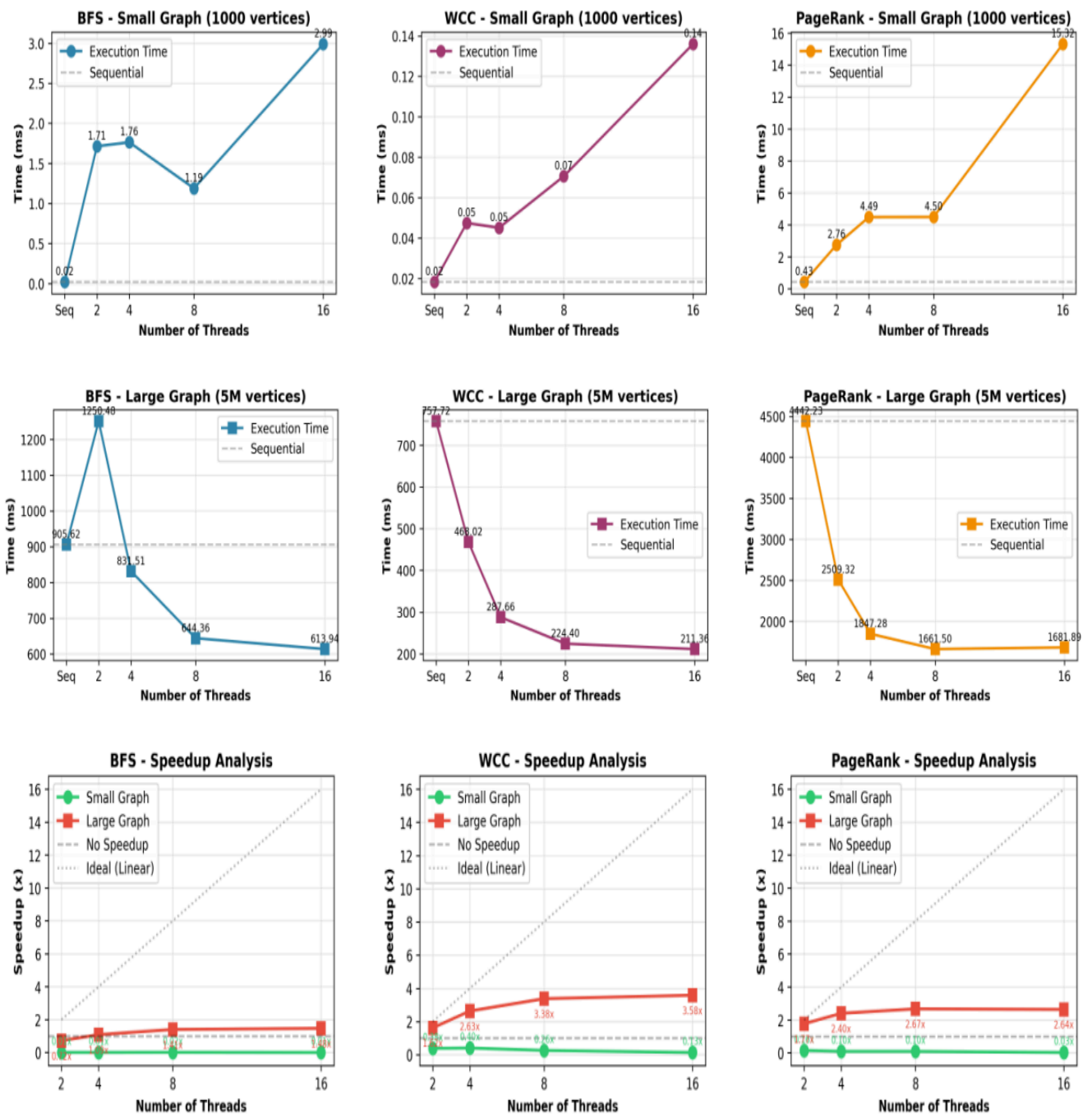
Ova bolja skalabilnost WCC algoritma može se objasniti prirodom algoritma i strukture grafa. Ako graf ima više manjih komponenti, različite niti mogu nezavisno raditi na različitim komponentama sa minimalnom interakcijom. Takođe, WCC koristi atomske operacije samo pri pokušaju zauzimanja čvorova, a nakon što nit uspešno zauzme startni čvor, BFS unutar te komponente je lokalna na toj niti bez potrebe za daljom sinhronizacijom sa drugim nitima.

PageRank skaliranje

PageRank pokazuje konzistentno i predvidljivo ponašanje skaliranja koje pada između BFS i WCC algoritama. Na malom grafu, vreme raste monotono sa brojem niti: 0.43 ms (sekvencijalno), 2.76 ms (2 niti), 4.49 ms (4 niti), 4.50 ms (8 niti), i 15.32 ms (16 niti). Ubrzanje ostaje ispod 1x za sve konfiguracije (0.03x-0.15x), potvrđujući da mali graf jednostavno nije dovoljno velik da opravda overhead paralelizacije.

Na velikom grafu, PageRank demonstrira dobro, ali ne spektakularno skaliranje. Od sekvencijalnog vremena od 4424.23 ms, vreme se smanjuje na 2509.32 ms (2 niti, ubrzanje 1.74x), 1847.28 ms (4 niti, ubrzanje 2.40x), 1661.50 ms (8 niti, ubrzanje 2.67x), i 1681.89 ms (16 niti, ubrzanje 2.64x).

Interesantno, između 8 i 16 niti uočava se blaga stagnacija i čak minimalno pogoršanje performansi (sa 1661.50 ms na 1681.89 ms). Ovo sugerše da je sa 8 niti dostignut memory bandwidth limit - dodavanje dodatnih niti ne pomaže jer sve niti konkurišu za isti resurs.



Slika 2. Skaliranje algoritama sa brojem niti.

6.3 Memorijsko zauzeće

Analiza memorijske potrošnje izvršena je na grafu sa 5,000,000 čvorova i 6,349,982 ivica, sa ciljem da se utvrdi overhead paralelizacije sa 16 niti. Rezultati, prikazani na Slici 3, pokazuju da su memorijske potrošnje sekvencijalnih i paralelnih verzija gotovo identične.

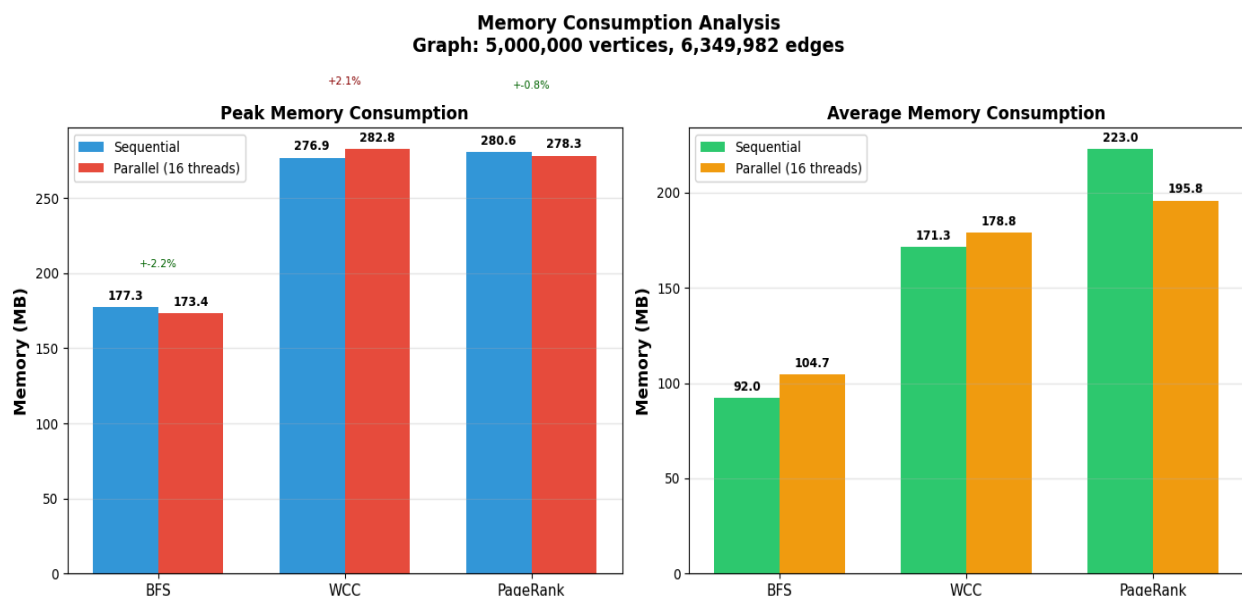
Minimalan overhead paralelizacije

Maksimalna memorijska potrošnja pokazuje izuzetno nizak overhead. BFS ima overhead od +2.2% (177.3 MB sekvencijalno naspram 173.4 MB paralelno), WCC pokazuje overhead od +2.1% (276.9 MB vs 282.8 MB), dok PageRank ima negativan overhead od -0.8% (280.6 MB vs 278.3 MB). Prosečan overhead za peak memoriju iznosi **manje od 2%**.

Prosečna memorijska potrošnja pokazuje sličan trend. BFS ima overhead od +13.8%, WCC od +4.4%, dok PageRank pokazuje smanjenje od -12.2%. Ove varijacije u prosečnoj potrošnji reflektuju dinamiku alociranja i dealociranja privremenih struktura tokom izvršavanja, pri čemu paralelne verzije često efikasnije upravljaju memorijom kroz kraći životni vek thread-local struktura.

Minimalan memorijski overhead od ~2% za maksimalnu potrošnju znači da nema značajnog trade-off-a između performansi i memorije. Za graf sa 5 miliona čvorova, svi algoritmi zahtevaju 177-283 MB memorije, što omogućava procesiranje velikih grafova na standardnom hardveru.

Ovako nizak overhead postignut je efikasnim deljenjem read-only grafovske strukture između svih niti, korišćenjem atomskih tipova iste veličine kao obični tipovi (AtomicI32 vs i32), i Rayon-ovim efikasnim thread pool-om koji minimizuje per-thread overhead.



7. ZAKLJUČAK

U ovom radu predstavljen je sveobuhvatan sistem za analitiku grafova velikih razmera, razvijen za potrebe *Fast Transit Network* mreže. Implementirane su efikasne CSR i bidirekzione CSR strukture podataka koje omogućavaju brz pristup susednim čvorovima. Razvijeni su sekvencijalni i paralelni algoritmi za BFS, WCC i PageRank, sa detaljnom verifikacijom korektnosti kroz skup testova.

CLI interfejs omogućava jednostavno korišćenje sistema sa mogućnošću podešavanja svih relevantnih parametara. Sistem je testiran na grafovima različitih veličina, od malih test grafova do velikih grafova sa milionima čvorova i ivica.

Implementacija u Rust programskom jeziku pokazala se kao izvrstan izbor zbog kombinacije performansi na nivou sistemskih jezika i bezbednosti memorije koja eliminiše čitavu klasu potencijalnih bagova. Rust-ov sistem vlasništva i borrowing pravila omogućili su implementaciju thread-safe paralelnih algoritama bez potrebe za garbage collection-om ili ručnim upravljanjem memorijom.

Rezultati pokazuju da je paralelizacija efikasna za grafove velikih razmera, dok mali grafovi pokazuju overhead paralelizacije. Ovaj rezultat je u skladu sa teorijom i praksom paralelnog programiranja, gde fiksni troškovi paralelizacije postaju zanemarljivi tek kad je posao dovoljno velik.

Buduća poboljšanja sistema mogu uključiti implementaciju adaptivne strategije koja automatski bira sekvencijalnu ili paralelnu verziju na osnovu veličine grafa, work-stealing pristup za bolju load balancing u WCC algoritmu, i potencijalnu SIMD optimizaciju za PageRank koji vrši veliku količinu floating-point računanja.

8. LITERATURA

1. Rayon: A data parallelism library for Rust. Dostupno na: <https://docs.rs/rayon/>
2. Klabnik, S., Nichols, C. (2023). The Rust Programming Language, 2nd Edition. No Starch Press.
3. https://en.wikipedia.org/wiki/Weak_component
4. https://en.wikipedia.org/wiki/Breadth-first_search
5. <https://en.wikipedia.org/wiki/PageRank>