Projektni zadatak – Računanje determinante matrice Laplasovim razvojem po prvoj vrsti

Mihajlo Kušljić
Softversko inženjerstvo i informacione tehnologije
Fakultet tehničkih nauka, Univerzitet u Novom Sadu
Novi Sad, Srbija
mihajlokusljic97@gmail.com

Motivacija problema

Determinante imaju široku primjenu u mnogim oblastima matematike. U algebri determinante se koriste za opisivanje invertibilnih matrica (imaju inverznu matricu) i da se opiše rješenje sistema linearnih jednačina pomoću Kramerovog pravila. Determinante se koriste i da se izračunaju zapremine u vektorskoj analizi: apsolutna vrijednost determinante realnih vektora jednaka je zapremini paralelopipeda koji grade ti vektori. Takođe na osnovu determinante može se utvrditi linearna zavisnost realnih vektora. U matematičkoj analizi, pri računanju integrala po podskupovima Euklidskog prostora R^n, koriste se tzv. jakobijani koji predstavljaju determinante Jakobijevih matrica...

Osnovni pojmovi

Da bi se definisale i proučile determinante, potrebno je uvesti pojam matrice, kao i osnovne pojmove o permutacijama:

- **Matrica** formata mn nad poljem F je funkcija Mmn koja preslikava skup uređenih parova $\{(i,j)|i \in \{1,2,\dots,m\} \wedge j \in \{1,2,\dots,n\}\}$ u skup F. Matrice formata nn, tj. matrice koje imaju isti broj redova i kolona, nazivamo kvadratne matrice **reda n**.
- Kvadratna **podmatrica** reda r matrice M_{mn} je kvadratna matrica reda r koja se dobija kada se iz matrice M_{mn} izbaci proizvoljnih m - r vrsta i n - r kolona.
- **Permutacija** skupa A je bijektivna funkcija koja preslikava skup A na samog sebe: $\sigma : A \xrightarrow{1-1,\mathrm{HA}} \mathrm{A}$. Na primjer, za skup $S = \{1,2,3\}$ jedna njegova permutacija bi bila: $\sigma = \binom{123}{231}$. Kako je skup S totalno uređen relacjom ≤, to će u prethodnom zapisu prva vrsta biti izostavljena, jer se podrazumjeva da su u njoj svi elementi skupa S u redoslijedu saglasno relaciji ≤ i permutacija će biti zapisana samo drugom vrstom: $\sigma = 231$. Pri tome ćemo sa $\sigma(i)$ označiti i-ti element u permutaciji, npr. u navedenom primjeru je $\sigma(2) = 3$. Skup svih permutacija skupa $S = \{1,2,\dots,n\}$ označavaće se sa S_n.
- Ako su $i < j$ и $\sigma(i) > \sigma(j)$, tada se uređeni par $(\sigma(i), \sigma(j))$ naziva **inverzija** permutacije σ. Broj svih inverzija permutacije σ označavaće se sa $Inv\ \sigma$.
- **Determinanta** u oznaci det je funkcija koja preslikava skup svih kvadratnih matrica u skup F, gdje je F polje nad kojim su definisane te matrice. Neka su A = $[a_{ij}]_{nn}$ proizvoljna matrica reda n i n proizvoljan prirodan broj. tada je det(A) definisana kao:

  $\det: M_{nn} \to F, \det(A) = \sum_{\sigma \in S_n} (-1)^{Inv\ \sigma} a_{1\sigma(1)} a_{2\sigma(2)} \dots a_{n\sigma(n)}$
- **Minor** reda r neke matrice M_{mn} je determinanta neke njene kvadratne podmatrice reda r

- **Laplasov razvoj** po vrsti/koloni je jedna od metoda izračunavanja determinante matrice. Ako razložimo matricu A po vrsi i tada je: $\det(A) = \sum_{j=1}^{n}(-1)^{i+j}A_{i,j}M_{i,j}$, gdje je $M_{i,j}$ minor, tj. determinanta podmatrica koje se dobije kada se iz matrice A ukloni i-ta vrsta i j-ta kolona.

Zadatak

Upotrebom programskih jezika Python i Go obezbjediti serijsku i paralelnu implementaciju računanja determinante kvadratne matrice reda n, primjenom Laplasovog razvoja po prvoj vrsti. Zadata matrica se učitava iz teksutalnog fajla koji prati slijedeći format:

- Prvi red u fajlu sadrži red matrice (n)
- Narednih n redova u fajlu sadrže vrste matrice, gdje svaki red sadrži n realnih brojeva razdvojenih barem jednim razmakom

Po učitavanju matrice izvršava se serijski i paralelni proračun determinante matrice. Rezultati se ispisuju na konzolu, a statistika o proračunima se dodaje u CSV fajl koji ima slijedeće zaglavlje:

- n - red matrice čija je determinanta izračunata
- exec_time_ms - vrijeme izvršavanja proračuna u milisekundama
- serial - da li je proračun serijski, moguće vrijednosti su: true (serijski propračun), false (paralelizovan proračun)
- implementation - jezik u kojem je propračun implementiran, moguće vrijednosti su: python, go

Argumenti programa su putanje do (validnih) datoteka koje sadrže matrice čije determinante treba izračunati i putanja do CSV fajla u kojem se čuvaju rezultati.

Pored glavnog toka programa, potrebno je obezbjediti i izvršavanje eksperimenata skaliranja u kojima se mjeri ubrzanje postignuto paralelizacijom, za različit broj niti/procesnih jedinica, u odnosu na serijsku implementaciju.

Sekvencijalno rješenje

Determinanta matrice računa se rekurzivnom primjenom Laplasovog razvoja po prvoj vrsti. Za računanje determinante matrice reda n razvojem po vrsti 0 potrebno je najprije odrediti vrijednosti n njenih minora: $M_{0,0}$, $M_{0,1}$, ... $M_{0,n-1}$. Računanje svakog od ovih minora svodi se na određivanje determinante podmatrice reda n – 1, ponovo primjenom Laplasovog razvoja po prvoj vrsti podmatrice... Baza rekurzije je slučaj kada se dođe do podmatrice reda 1. Tada je vrijednost determinante jednaka vrijednosti jedinog elementa podmatrice. Kada se završi računanje pomenutih minora, vrijednost determinante se određuje po formuli: $\det(A) = \sum_{j=0}^{n-1}(-1)^{0+j}A_{0,j}M_{0,j}$. Možemo primjetiti da u sumi prvi činilac naizmjenično uzima vrijednosti 1 i -1, drugi činilac je element polazne matrice a treći činilac je jedan od izračunatih minora.

Paralelno rješenje

Proračuni minora $M_{0,0}$, $M_{0,1}$, ... $M_{0,n-1}$, potrebnih za određivanje determinante matrice, su međusobno nezavisni i mogu se izvršiti paralelno. Za određivanje determinante matrice reda n koristeći p niti/procesnih jedinica, zadaci određivanja minora $M_{0,0}$, $M_{0,1}$, ... $M_{0,n-1}$, dodjeljuju se procesnim jedinicama po *Round-Robin* principu. Npr. za matricu reda 5 i 3 procesne jedinice (u oznaci p0, p1, p2)

računanje $M_{0,0}$ dodjeljuje se procesnoj jedinici p0, računanje $M_{0,1}$ dodjeljuje se p1, računanje $M_{0,2}$ dodjeljuje se p2, računanje $M_{0,3}$ dodjeljuje se p0 i računanje $M_{0,4}$ dodjeljuje se p1. Procesne jedinice izračunavaju u paraleli dodjeljene minore. Kada se izračunaju minori, početna procesna jedinica, p0, određuje vrijednost determinante. Svaki od pomenutih minora je reda n-1. U idealnom slučaju, kada je red matrice djeljiv brojem procesnih jedinica, sve procesne jedinice računaju isti broj minora istih dimenzija, pa svaka procesna jedinica dobija istu količinu posla. Ukoliko red matrice nije djeljiv brojem procesnih jedinica, neke procesne jedinice će imati više posla od drugih, što dovodi do neefikasnog korišćenja procesnih jedinica i ograničava ubrzanje. Npr., u navedenom primjeru procesne jedinice p0 i p1 treba da izračunaju 2 minora, dok procesna jedinica p2 treba da izračuna jedan minor. U opštem slučaju za određivanje determinante matrice reda n u paralelnoj implementaciji koristi se n procesnih jedinica. Tada svaka procesna jedinica treba da izračuna jedan minor, reda n – 1, čime se svim procesnim jedinicama dodjeljuje jednaka količina posla.
U Python implementaciji za procesne jedinice koriste se procesi iz standardne multiprocessing biblioteke, dok se u Go implementaciji koriste Go rutine.

Eksperimenti skaliranja

U eksperimentima skaliranja prati se ubrzanje postignuto paralelizacijom, o odnosu na dimenzije ulaznih podataka i/ili broj korišćenih procesnih jedinica. Ubrzanje se računa kao odnos trajanja serijskog proračuna i trajanja paralelizovanog proračuna za iste ulazne podatke. Ako je problem idealan za paralelizaciju (sav kod može da se izvršava paralelno) tada je maskimalno očekivano ubrzanje jednako broju procesnih jedinica, npr. ako imamo duplo više procesnih jedinica očekujemo da proračun bude duplo brži. Većina problema ne mogu idealno da se paralelizuju, pa se za predikciju ubrzanja koje je moguće postići paralelizacijom koriste 3 faktora:

- s - udio vremena serijskog proračuna provedenog u izvršavanju koda kojeg nije moguće paralelizovati,
- p - udio vremena serijskog proračuna provedenog u izvršavanju koda koje se može paralelizovati,
- N – broj procesnih jedinica.

Ako pogledamo problem računanja determinante Laplasovim razvojem po prvoj vrsti, serijski kod se svodi na podjelu zadataka za izračunavanje minora i sračunavanje vrijednosti determinante na osnovu dobijenih minora, dok paralelni kod obuhvata izračunavanje minora. U kontekstu vremena izvršavanja serijkski kod se može zanemariti u odnosu na paralelni, što je potvrđeno odgovarajućim mjerenjima u eksperimentu. Dakle za dati problem je s = 0 i p = 1, pa je teorijsko maksimalno ubrzanje jednako broju procesnih jedinica.

Eksperimenti skaliranja izvršeni su na arhitekturi slijedećih specifikacija:

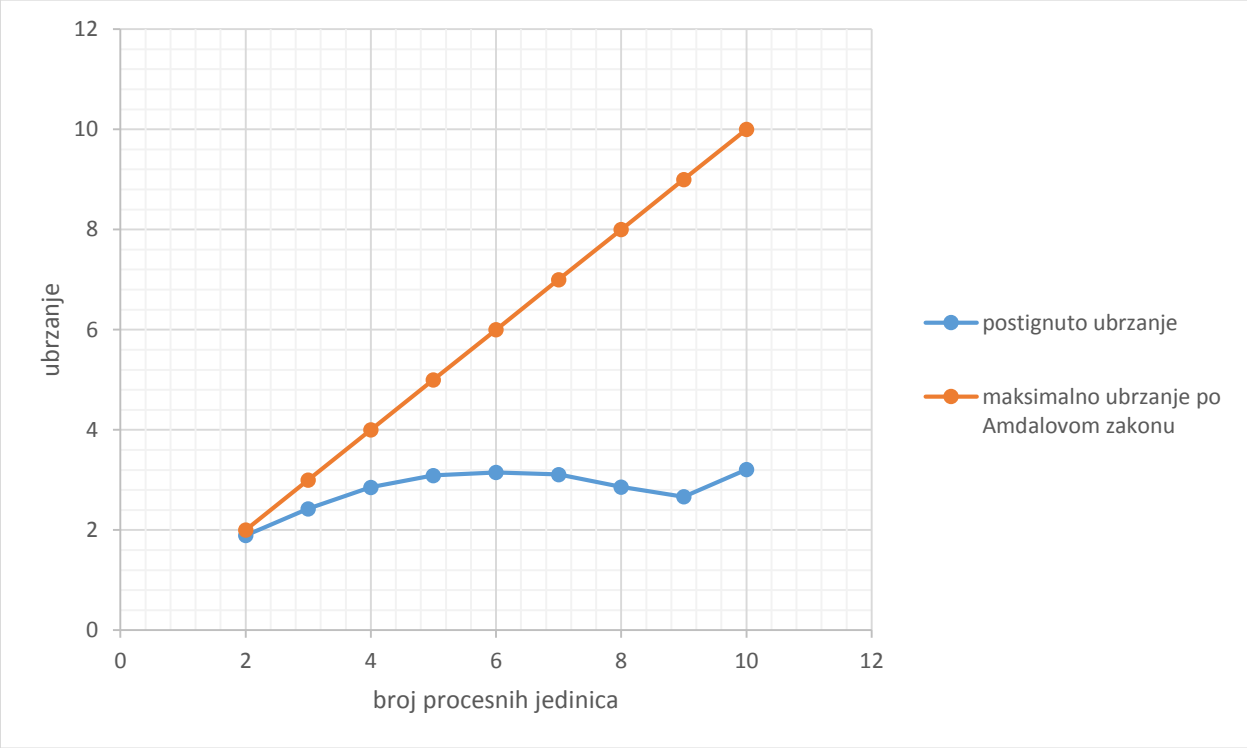
- procesor: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz (8 CPUs), ~2.6GHz
- RAM memorija: 12 GB
- Grafička kartica: NVIDIA GeForce GTX 950M @ 4 GB VRAM
- Operativni sistem: Windows 8.1 Pro 64-bit (6.3, Build 9600)

Jako skaliranje

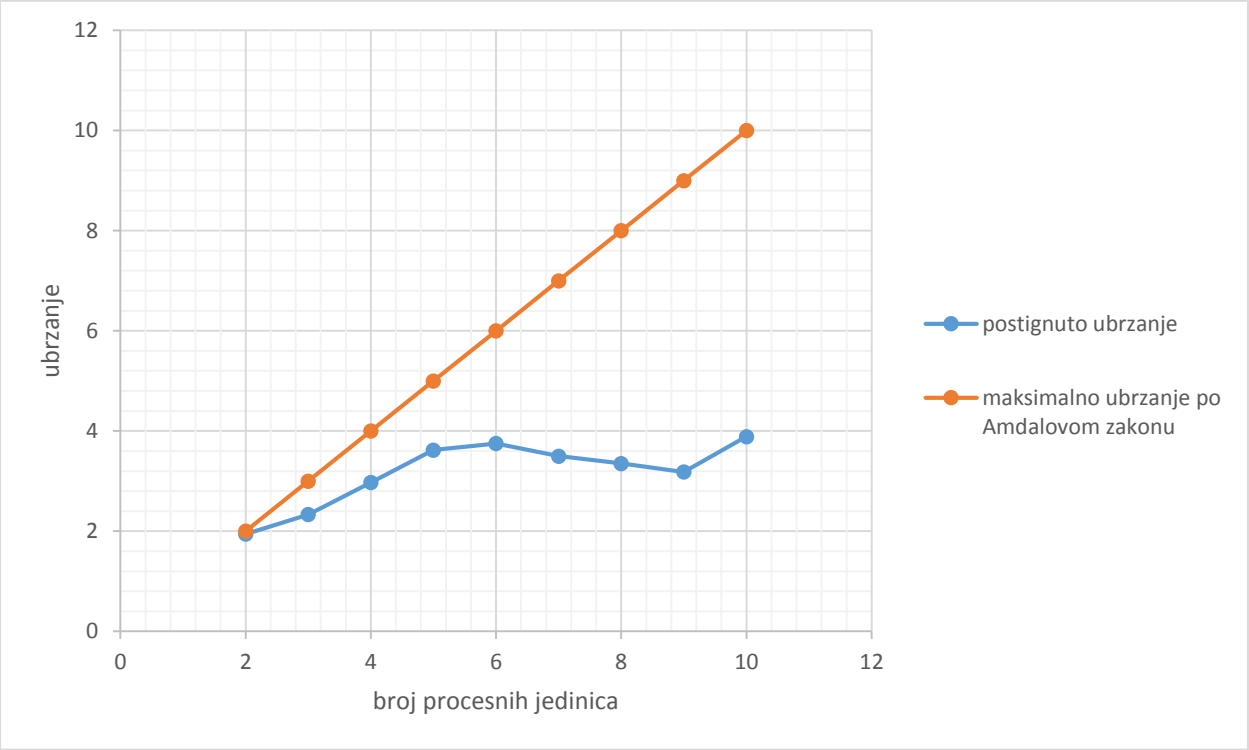
Kod jakog skaliranja ideja je da pratimo ostvareno ubrzanje paralelne implementacije kada su dimenzije ulaznih podataka iste, a povećavamo broj procesnih jedinica. Tada je teorijski maksimum ubrzanja određen Amdalovim zakonom: *speedup = 1 / (s + p / N)*.

U implementaciji jakog skaliranja najprije se učitava predefinisana matrica reda 10, koja se nalazi u direktorijumu test_data unutar repozitorijuma. Nad ovom matricom vrši se serijski proračun determinante i mjeri se njegovo trajanje koje služi kao osnova za računanje ubrzanja ostvarenog paralelizacijom. Potom se za istu matricu računa determinanta paralelnim proračunima pri čemu se mijenja broj korišćenih procesnih jedinica: N = 2, 3, ..., 10. Svaki put se mjeri trajanje paralelnog proračuna, te izračunava ostvareno ubrzanje u odnosu na serijski proračun kao i maksimalno ubrzanje po Amdalovom zakonu za dati broj procesnih jedinica i izmjerene vrijednosti s i p. Dobijeni rezultati se ispisuju na konzolu i čuvaju u CSV fajlovima u results direktorijumu unutar repozitorijuma.

Grafik 1 prikazuje rezultate jakog skaliranja u Python implementaciji. Grafik 2 prikazuje rezultate jakog skaliranja u Go implementaciji. Na x osi nalazi se broj procesnih jedinica korisšćenih u paralelnom proračunu, dok je na y osi prikazano ubrzanje u odnosu na serijski proračun. Plava linija predstavlja postignuto ubrzanje, izmjereno kao odnos vremena izvršavanja serijskog i paralelnog proračuna. Narandžasta linija predstavlja idealno teorijsko ubrzanje po Amdalovom zakonu, za dati broj procesnih jedinica. Vidimo da povećanje broja procesnih jedinica preko 5 ne dovodi do značajnijeg ubrzanja, tj. ubrzanje doživljava plato. Dakle, za matricu reda 10 dovoljno je angažovati 5 procesnih jedinica. Takođe možemo primjetiti da Go implementacija postiže nešto veće ubrzanje.



Grafik 1 – jako skaliranje Python paralelne implementacije u skladu sa Amdalovim zakonom za red matrice 10



Grafik 2 - jako skaliranje Go paralelne implementacije u skladu sa Amdalovim zakonom za red matrice 10

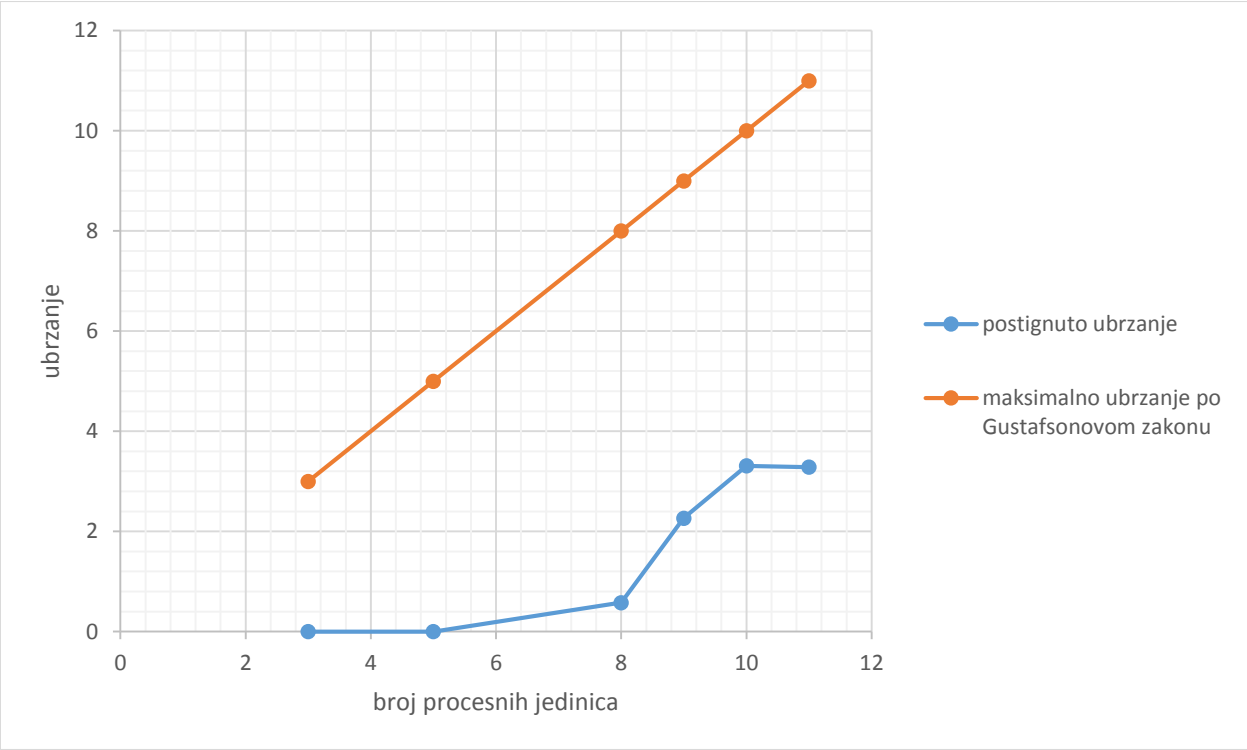
Slabo skaliranje

Kod slabog skaliranja ideja je da sa povećanjem broja procesnih jedinica proporcionalno povećavamo dimenzije ulaznih podataka. Tada je teorijski maksimum ubrzanja određen Gustafsonovim zakonom: *speedup = s + p × N*.

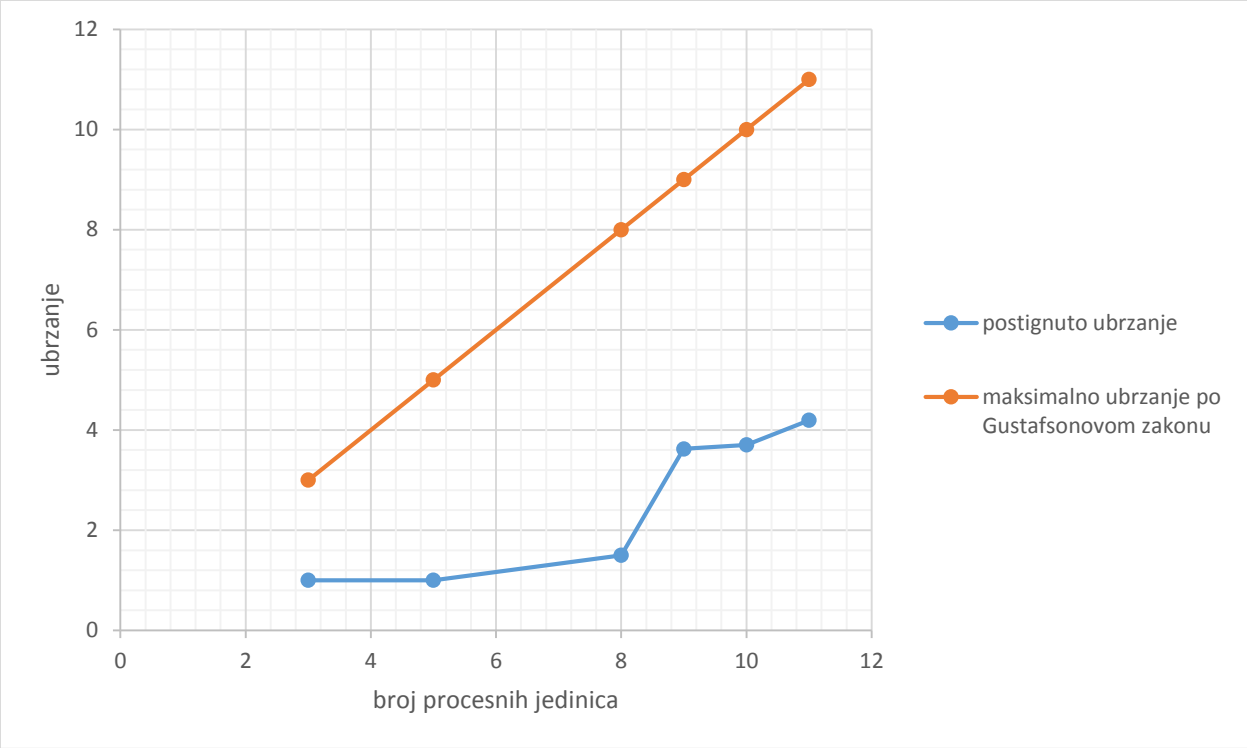
U ovom slučaju skaliranje posla je urađeno tako da je red ulazne matrice (n) jednak broju procesnih jedinica koje učestvuju u paralelnom proračunu. Na taj način svaka procesna jedinica dobija istu količinu posla: računa jedan minor reda n – 1. Za potrebe eksperimenta koriste se predefinisane matrice, redova 3, 5, 8, 9, 10 i 11, date u test_data direktorijumu, unutar repozitorijuma. Svaka od matrica se najprije učitava iz fajla. Zatim se vrši serijski proračun determinante, uz mjerenje vremena izvršavanja. Potom se vrši paralelizovan proračun determinante uz mjerenje vremena izvršavanja. Postignuto ubrzanje računa se kao odnos vremena izvršavanja serijskog proračuna i vremena izvršavanja paralelizovanog proračuna. Takođe se računa maksimalno teorijsko ubrzanje po Gustafsonovom zakonu, za dati broj procesnih. Dobijeni rezultati se ispisuju na konzolu i čuvaju u CSV fajlovima u results direktorijumu unutar repozitorijuma.

Grafik 3 prikazuje rezultate slabog skaliranja u Python implementaciji. Grafik 4 prikazuje rezultate slabog skaliranja u Go implementaciji. Na x osi nalazi se broj procesnih jedinica korišćenih u paralelnom proračunu koji ujedno odgovara i redu ulazne matrice, dok je na y osi prikazano ubrzanje u odnosu na serijski proračun. Plava linija predstavlja postignuto ubrzanje. Narandžasta linija predstavlja idealno

teorijsko ubrzanje po Gustafsonovom zakonu, za dati broj procesnih jedinica. Vidimo da povećanje broja procesnih jedinica preko 10 ne dovodi do značajnijeg ubrzanja, tj. ubrzanje doživljava plato. Takođe možemo primjetiti da Go implementacija postiže nešto veće ubrzanje.



Grafik 3 - slabo skaliranje Python paralelne implementacije u skladu sa Gustafsonovim zakonom



Grafik 4 - slabo skaliranje Go paralelne implementacije u skladu sa Gustafsonovim zakonom