Napredne tehnike programiranja

02.07.2024.

Nemanja Vujadinović

Pregled

Izveštaj prikazuje rezultate slabog i jakog skaliranja u okviru *Python* i *Rust* programskih jezika. Zadatak je da se nad ulaznom matricom primene operacije konvolucije, aktivacione funkcije i maksimalnog grupisanja.

Tehnički detalji

- Model procesora: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
- Organizacija *cache* memorije:
    a. L1 *cache*: 256kB
    b. L2 *cache*: 1.0mB
    c. L3 *cache*: 6.0mB
- Broj *NUMA node*-ova:
    a. *No. of cores*: 4
    b. *No. of logical processors*: 8
- Tip i količina *RAM* memorije: *DDR3, 16.0gB*
- Operativni sistem: *Windows 11*
- Biblioteke:
    a. ndarray = "0.15" (Rust)
    b. rand = "0.8" (Rust)
    c. csv = "1.1.6" (Rust)
    d. matrix_display = "0.3" (Rust)
    e. image = "0.23" (Rust)
    f. plotters = "0.3.3" (Rust)
    g. rayon = "1.8" (Rust)
    h. num-integer = "0.1" (Rust)
    i. numpy=1.25.0 (Python)
    j. csv (Python)
    k. multiprocessing (Python)
- Ostale informacije: Najbolji rezultati testova, dobijeni su kada se računar nalazio na punjaču i kada je *Power mode* u podešavanjima bio postavljen na *Best performance.* U toku testova, ugašeni su svi drugi programi i sam računar nije bio korišćen, jer je i samo otvaranje drugih fajlova umelo da pogorša rezultate za 20-30%.

Procenat sekvencijalnog i paralelnog koda

Kako bi se izračunala teorijska ubrzanja, potrebno je naći delove koda koji se mogu i ne mogu paralelizovati. Kod za učitavanje matrica iz fajlova je deo koji nije moguće paralelizovati, dok je delove za računanje konvolucije, aktivacione funkcije i maksimalnog grupisanja bilo moguće paralelizovati. Konkretne vrednosti biće navedene u tabelama u narednim poglavljima.

Bitno je napomenuti da je u slučaju ovog projekta, procenat sekvencijalnog koda umeo ići i do 50%, usled velikih matrica koje su učitavane. Rešenje ovog problema moglo bi biti čuvanje matrica u *hdf5* fajlovima. Ipak, ovaj format fajlova nije korišćen zbog problema sa *Rust dependency*-ima.

Osnovne informacije

Za *Python* paralelno rešenje, korišćena je biblioteka *multiprocessing*. Za *Rust* paralelno rešenje, korišćena je biblioteka *rayon.*

Legenda za oznake u tabelama:

- *dimension*: veličina ulazne matrice nad kojom se vrši konvolucija, aktivacija i grupisanje
- *No. of processes*: broj procesa pri paralelnom izvršavanju
- *t_not_par:* vreme izvršavanja koda koji ne može biti paralelizovan
- *t_s*: vreme izvršavanja sekvencijalnog rešenja
- *t_p*: vreme izvršavanja paralelnog rešenja
- *real speedup*: ubrzanje dobijeno rešenjem
- *theory speedup*: teorijsko ubrzanje

Kao filter matrica, u svakom slučaju se koristila matrica veličine 2x2, kako bi broj operacija bio što veći.

Grafici su napravljeni uz pomoć *Plotters* biblioteke u *Rust* jeziku.

Outlier-i u konačnim podacima nisu bili prisutni. Postojali su slučajevi kada je vreme izvršavanja bilo značajno veće od dotadašnjeg trenda. Ovo je najčešće bilo uzrokovano korišćenjem drugih programa u toku izvršavanja rešenja, što je usporilo konačno vreme. U takvim slučajevima, zapisana vremena su izbrisana, a rešenje pokrenuto ponovo.

Slabo skaliranje - Python

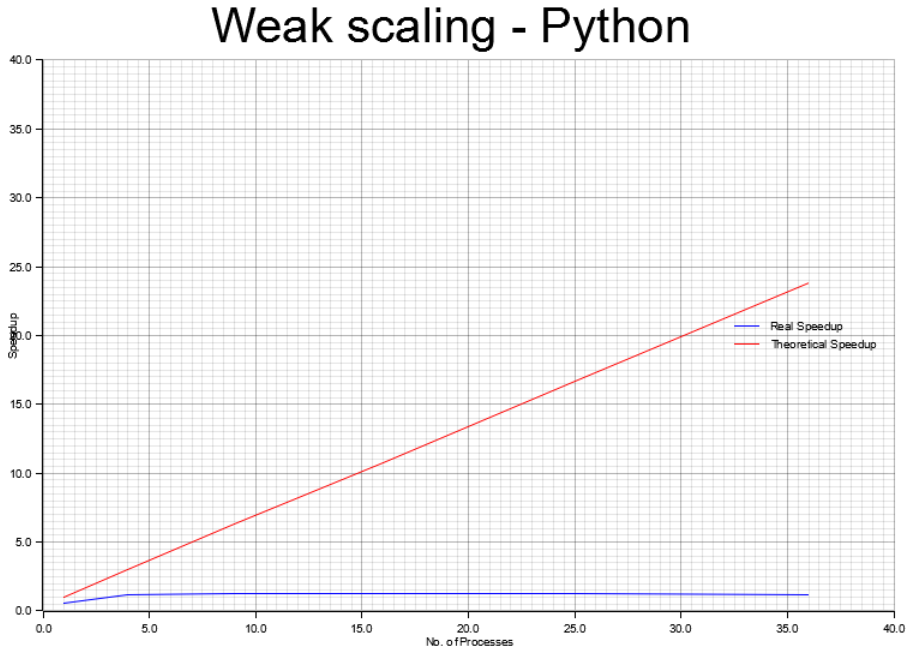
Broj procesa u slabom skaliranju može da ima vrednosti od 1, 4, 9, 16, 25 i 36 - ovaj broj mora biti kvadrat prirodnog broja kako bi se matrica podelila na jednake podmatrice.

Za slučaj kada je korišćen samo 1 proces, ulazna matrica je podešena na veličinu 512 x 512. Želimo da povećavamo broj procesa, ali tako da pri svakom testu, jedan proces vrši istu količinu posla kao i svaki drugi u ostalim testovima. Za jedan proces važi: f(x) = 1 x 512 x 512 = (1 x 512) x (1 x 512). Analogno, za 4 procesa biće: f(x) = 4 x 512 x 512 = (2 x 512) x (2 x 512) = 1024 x 1024. Zato, za 4 procesa koristimo matricu 1024 x 1024. Veličina matrice koju obrađuje jedan od četiri procesa je 512 x 512, što je ista veličina kao i u slučaju kada postoji samo jedan proces. Za ostali broj procesa, postupak je identičan, a konkretne veličine ulaznih matrica prikazane su u tabeli 1 (formula: (512 x sqrt(no. of processes) x (512 x sqrt(no. of processes)) . Takođe, veličina filter matrica je uvek ista, te će broj operacija jednog procesa zaista uvek biti isti.

Po Gustafsonovom zakonu, teorijsko ubrzanje računamo po formuli:

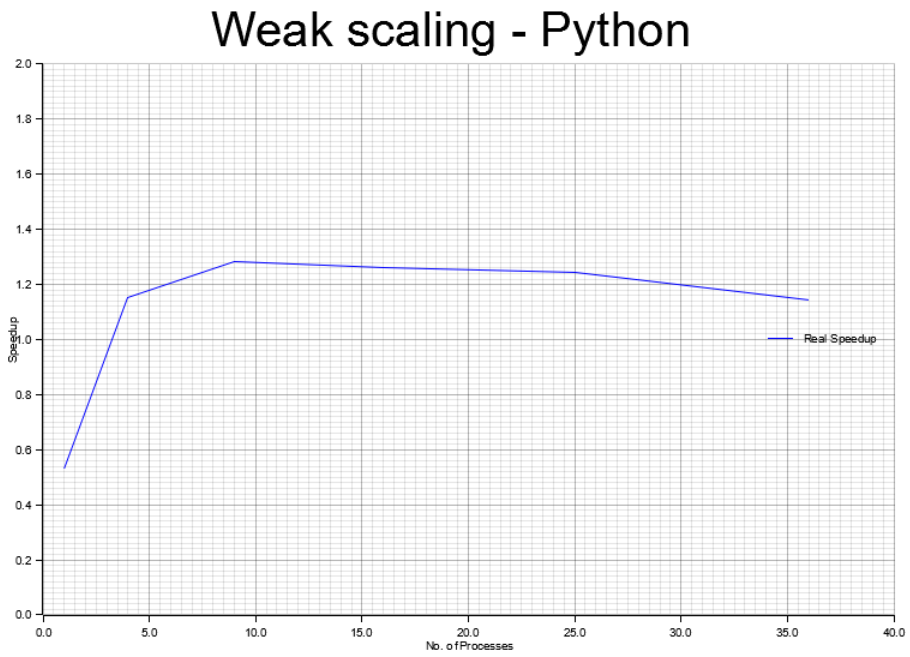
$$\text{speedup} = s + p \times N$$

Gledajući tabelu, s nalazimo kao količnik kolone t_not_par i t_s, a p kao $1\text{-}s$. Teorijsko ubrzanje, za svaki test, prikazano je u poslednjoj koloni tabele 1.



Slika 1: Naše i teorijsko ubrzanje pri slabom skaliranju u Python-u

Sa grafika na slici 1 vidimo da na samom početku, naše i teorijsko ubrzanje doživljavaju oštar skok, ali ne u istoj meri. Teorijsko ubrzanje raste linearno, dosta brže nego naše ubrzanje i ide do vrednosti 24. S druge strane, naše ubrzanje na početku ima linearan trend, ali taj trend nestaje u nastavku. Takođe, naše ubrzanje ne prelazi vrednost 2. Prikazaćemo još jedan grafik da dobijemo bolje informacije o našem ubrzanju.



Slika 2: Naše ubrzanje pri slabom skaliranju u Python-u

Sa grafika na slici 2, vidimo da ubrzanje raste dokle god je broj procesa manji ili jednak od 9, a posle toga krene u blagi pad (videti tabelu 1 za konkretne vrednosti). Ubrzanje za 9, 16 i 25 procesa je veće nego kod 4 procesa, ali je za 36 procesa to ubrzanje manje.

Po grafiku teorijskog ubrzanja, dupliranje broja procesa trebalo bi da rezultira duplo većim ubrzanjem. Kada poredimo ubrzanje pri 4 i 9 procesa (dok ubrzanje raste), vidimo da taj skok iznosi 1.11, što je dosta manje.

U tabeli 1, prikazane su srednje vrednosti izvršavanja za date ulazne parametre. Za svaku kombinaciju parametara, rešenja su pokretana bar 30 puta.

dimension	no. of processes	t_not_par [s]	t_s [s]	t_p [s]	real speedup	theory speedup
512x512	1	0.12614	0.35826	0.67942	0.52730	1
1024x1024	4	0.49902	1.45499	1.2613	1.15356	2.97108

1536x1536	9	1.14104	3.35376	2.61631	1.28186	6.27818
2048x2048	16	2.08191	5.91683	4.69213	1.26101	10.72206
2560x2560	25	3.19202	9.21561	7.41552	1.2427	16.6870
3072x3072	36	4.5052	12.94581	11.3234	1.14327	23.8197

Tabela 1: Srednja vremena izvršavanja za slabo skaliranje u Python-u

U tabeli 2, prikazane su vrednosti standardne devijacije za vremena izvršavanja.

dimension	no. of processes	t_not_par	t_s	t_p
512x512	1	0.01200	0.0174	0.03891
1024x1024	4	0.0210	0.05101	0.04045
1536x1536	9	0.06582	0.15549	0.12847
2048x2048	16	0.06939	0.13537	0.24098
2560x2560	25	0.11205	0.31157	0.20473
3072x3072	36	0.20432	0.27726	0.71545

Tabela 2: Standardne devijacije vremena izvršavanja za slabo skaliranje u Python-u

Iz tabele primećujemo da je najveća vrednost standardne devijacije kod paralelnog izvršavanja onda kada je i najveća veličina ulazne matrice. Kada dodatno posmatramo i grafike, možemo reći da rezultat pri 36 procesa varira u vremenu izvršavanja, te da takva pojava ima direktan uticaj na krajnji rezultat.

Jako skaliranje - Python

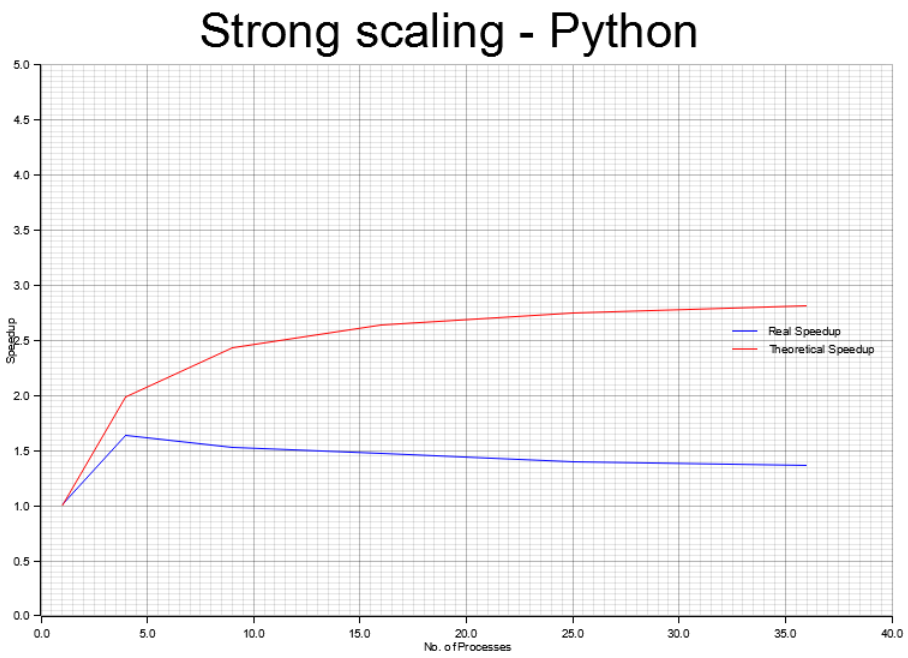
Broj procesa u jakom skaliranju može da ima vrednosti od 1, 4, 9, 16, 25 i 36.

U svim slučajevima, ulazna matrica ima veličinu 3600 x 3600.

Po Amdalovom zakonu, teorijsko ubrzanje računamo po formuli:

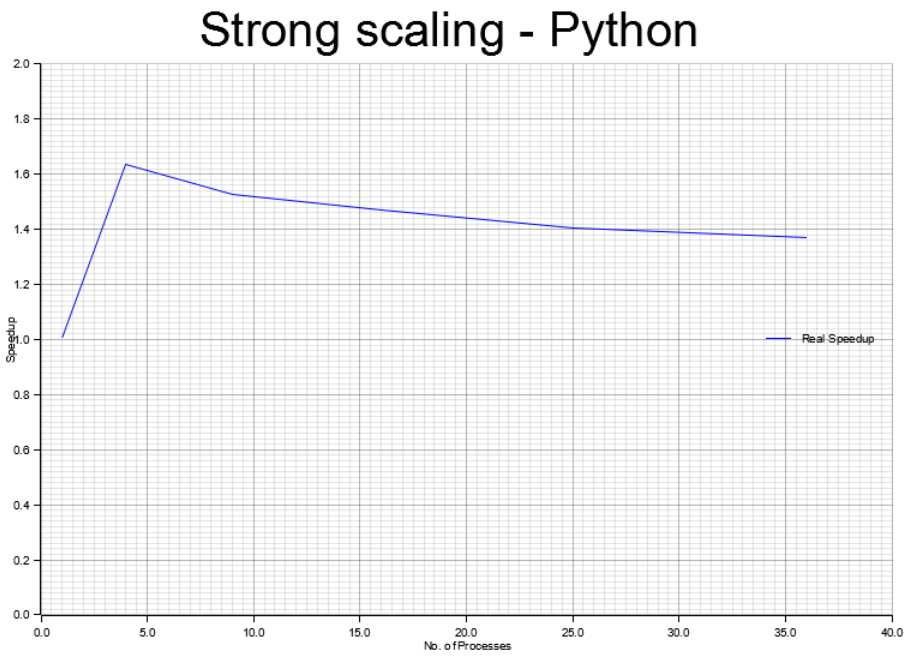
$$\text{speedup} = 1 \,/\, (s + p \,/\, N)$$

Gledajući tabelu, s nalazimo kao količnik kolone t_not_par i t_s, a p kao $1\text{-}s$. Teorijsko ubrzanje, za svaki test, je prikazano u poslednjoj koloni tabele 3.



Slika 3: Naše i teorijsko ubrzanje pri jakom skaliranju u Python-u

Sa grafika na slici 3 vidimo da na samom početku, naše i teorijsko ubrzanje doživljavaju oštar skok, koje je čak u približnoj meri. Teorijsko ubrzanje izgleda poput krive i ide ka asimptoti sa vrednošću y=3. Takođe, teorijsko ubrzanje raste brzo do 16. procesa, a onda usporava. Ovo se dešava jer je deo koda koji ne može da se paralelizuje (učitavanje matrica) proporcionalno velik, te kriva nema mogućnost većeg rasta na početku (ref. https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling/). S druge strane, naše ubrzanje na početku ima visok skok , ali taj trend nestaje u nastavku i ubrzanje opada. Prikazaćemo još jedan grafik da dobijemo bolje informacije o našem ubrzanju.



Slika 4: Naše ubrzanje pri jakom skaliranju u Python-u

Sa grafika na slici 4, vidimo da je najveća vrednost ubrzanja pri 4 procesa, posle čega ono konstantno opada. Naše ubrzanje očigledno ne prati trend rasta i krive koje teorijsko ubrzanje ima.

U tabeli 3, prikazane su srednje vrednosti izvršavanja za date ulazne parametre. Za svaku kombinaciju parametara, rešenja su pokretana bar 30 puta.

dimension	no. of processes	t_not_par [s]	t_s [s]	t_p [s]	real speedup	theory speedup
3600x3600	1	5.9969	17.78869	17.80792	0.99892	1
3600x3600	4	5.9969	17.78869	10.87135	1.63628	1.98870
3600x3600	9	5.9969	17.78869	11.66048	1.52555	2.43443
3600x3600	16	5.9969	17.78869	12.08460	1.47201	2.64165
3600x3600	25	5.9969	17.78869	12.66572	1.40447	2.75000
3600x3600	36	5.9969	17.78869	12.99575	1.36880	2.81267

Tabela 3: Srednja vremena izvršavanja za jako skaliranje u Python-u

U tabeli 4, prikazane su vrednosti standardne devijacije za vremena izvršavanja.

dimension	no. of processes	t_not_par	t_s	t_p
3600x3600	1	0.09804	0.39831	0.42024
3600x3600	4	0.09804	0.39831	0.51983
3600x3600	9	0.09804	0.39831	0.76111
3600x3600	16	0.09804	0.39831	0.42269
3600x3600	25	0.09804	0.39831	0.54390
3600x3600	36	0.09804	0.39831	0.32524

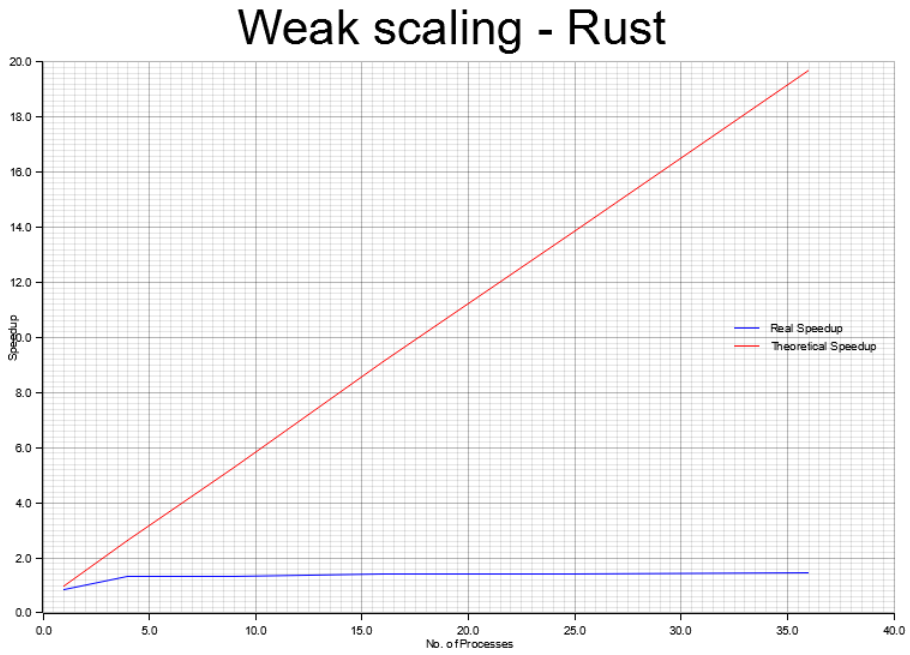
Tabela 4: Standardne devijacije vremena izvršavanja za jako skaliranje u Python-u

Iz tabele 4, primećujemo da je najveća vrednost standardne devijacije kod paralelnog izvršavanja onda kada je korišćeno 9 procesa. Kada dodatno posmatramo i grafike, možemo reći da je ovo direktan uzrok najvećeg pada vremena izvršavanja između dva procesa - najveći pad se dešava upravo između 4 i 9 procesa.

Slabo skaliranje - Rust

Kao i kod *Python* slabog skaliranja, broj procesa može da ima vrednosti od 1, 4, 9, 16, 25 i 36, a veličine matrica prikazane su u tabeli 5.

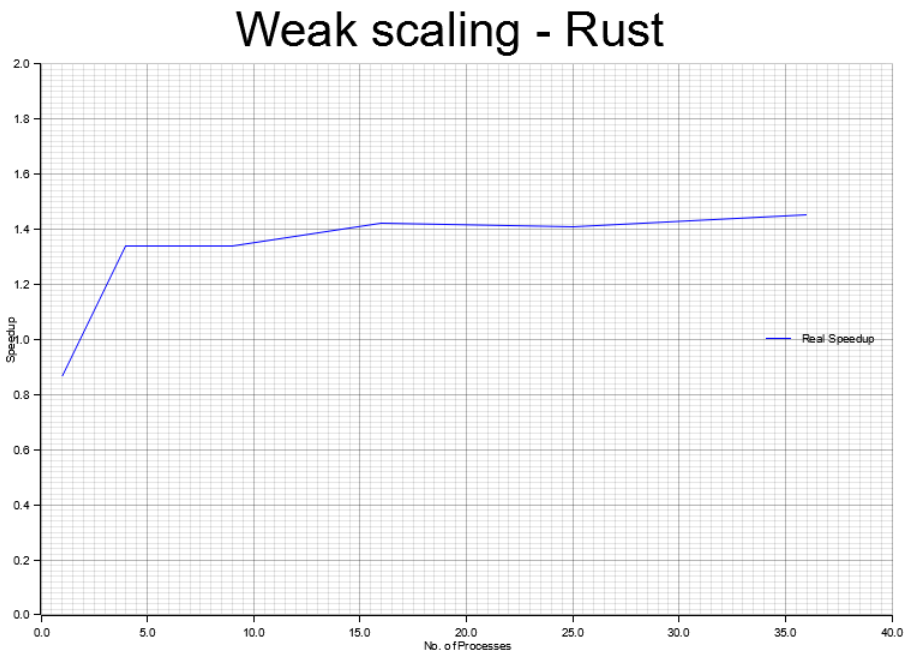
Teorijsko ubrzanje računamo na isti način kao i kod *Python* slabog skaliranja i ono je prikazano u poslednjoj koloni tabele 5.



Slika 5: Naše i teorijsko ubrzanje pri slabom skaliranju u Rust-u

Sa grafika na slici 5 vidimo da na samom početku, naše i teorijsko ubrzanje doživljavaju oštar skok, ali ne u istoj meri. Teorijsko ubrzanje raste linearno, brže nego naše ubrzanje i ide skoro do vrednosti 20. S druge strane, naše ubrzanje takođe ima linearan trend, ali trend značajno usporava posle 4 procesa. Takođe, naše ubrzanje ne prelazi vrednost 2.

Prikazaćemo još jedan grafik da dobijemo bolje informacije o našem ubrzanju.



Slika 6: Naše ubrzanje pri slabom skaliranju u Rust-u

Sa grafika na slici 6, vidimo da naše ubrzanje raste skoro sve vreme - pad se dešava između vremena pri 4 i 9 procesa (razlika u vremenima iznosi 0.001), kao i između vremena pri 16 i 25 procesa (0.011). Ipak, ovakve razlike su dosta male i na prvi pogled bi se moglo reći da ubrzanje raste sve vreme. Najveće ubrzanje javlja se pri 36 procesa, što je dobar znak, jer bi u eventualnom većem broju procesa, ono nastavilo da raste.

Odnos teorijskog ubrzanja pri 4 i 36 procesa iznosi 7.52, dok je odnos našeg ubrzanja u istom slučaju jednak 1.08. Značajno manje, ali rastući trend je dobra stvar.

Svakako, u poređenju sa grafikom pri *Python* slabom skaliranju, rezultati su značajno bolji. Naše ubrzanje manje odstupa od teorijskog, kako po pitanju samih vrednosti, tako i po pitanju rastućeg i linearnog trenda.

U tabeli 5, prikazane su srednje vrednosti izvršavanja za date ulazne parametre. Za svaku kombinaciju parametara, rešenja su pokretana bar 30 puta.

dimension	no. of processes	t_not_par [s]	t_s [s]	t_p [s]	real speedup	theory speedup
512x512	1	0.0233	0.05006	0.05796	0.86377	1
1024x1024	4	0.09017	0.19620	0.14620	1.34198	2.62126

1536x1536	9	0.20165	0.43355	0.32334	1.34083	5.2790
2048x2048	16	0.34462	0.75286	0.52937	1.42215	9.13378
2560x2560	25	0.56282	1.21255	1.41147	1.41147	13.85997
3072x3072	36	0.79251	1.70468	1.45305	1.45305	19.72835

Tabela 5: Srednja vremena izvršavanja za slabo skaliranje u Rust-u

U tabeli 6, prikazane su vrednosti standardne devijacije za vremena izvršavanja.

dimension	no. of processes	t_not_par	t_s	t_p
512x512	1	0.0014	0.002129	0.006024
1024x1024	4	0.00317	0.008293	0.009954
1536x1536	9	0.00584	0.013322	0.00771
2048x2048	16	0.00521	0.011218	0.01165
2560x2560	25	0.02016	0.059373	0.01241
3072x3072	36	0.00812	0.018445	0.014306

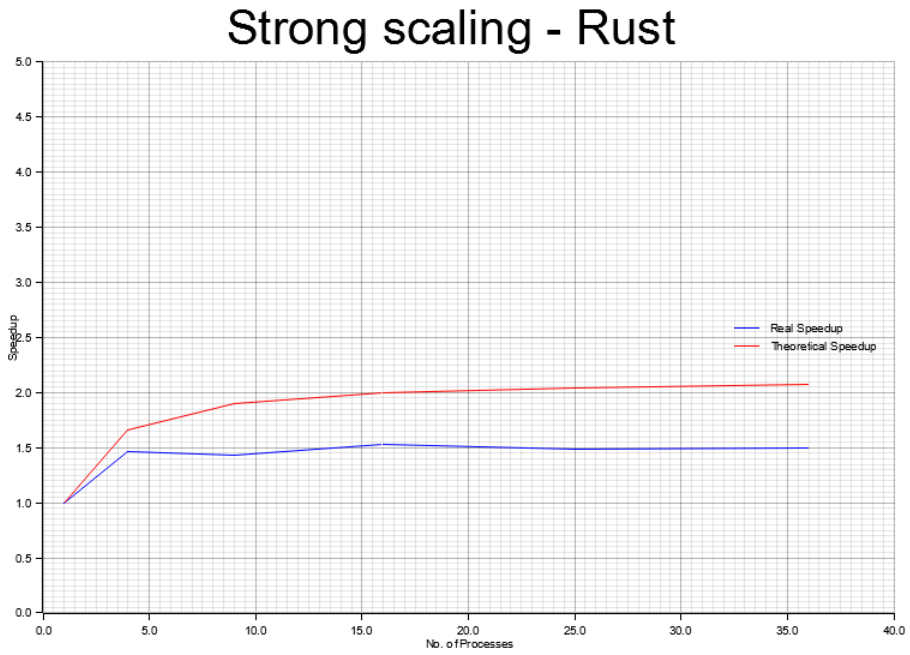
Tabela 6: Standardne devijacije vremena izvršavanja za slabo skaliranje u Rust-u

Iz tabele primećujemo da vrednosti standardnih devijacija nemaju konkretan rastući trend što je dobra stvar. Ove vrednosti su zaista niske, pogotovo u poređenju sa vrednostima u *Python* slabom skaliranju. Ovo takođe objašnjava i zašto naše ubrzanje pri *Rust* slabom skaliranju ima rastući trend, dok *Python* ubrzanje to nema.

Jako skaliranje - Rust

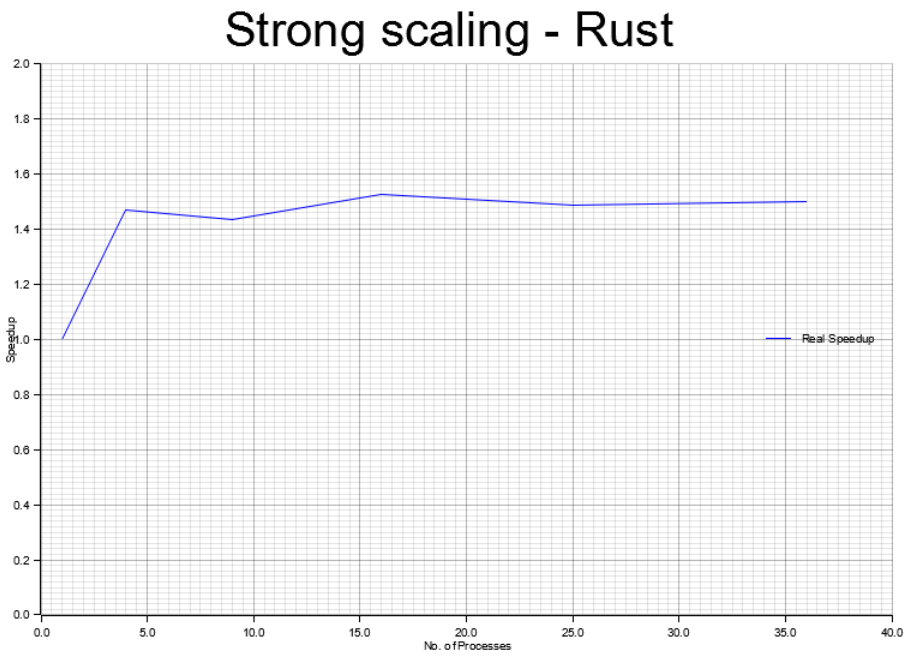
Kao i kod *Python* jakog skaliranja, broj procesa može da ima vrednosti od 1, 4, 9, 16, 25 i 36, a veličina ulazne matrice je 3600 x 3600.

Teorijsko ubrzanje računamo na isti način kao i kod *Python* jakog skaliranja i ono je prikazano u poslednjoj koloni tabele 7.



Slika 7: Naše i teorijsko ubrzanje pri jakom skaliranju u Rust-u

Sa grafika na slici 7 primećujemo da oba ubrzanja na samom početku doživljavaju oštar skok, dok su vrednosti oba prilično slične. Teorijsko ubrzanje izgleda poput krive i ide ka asimptoti sa vrednošću y=2.2. U poređenju sa *Python* jakim skaliranjem, maksimalna vrednost teorijskog ubrzanja je manja. *Rust* sekvencijalni je značajno brži od *Python* koda, te operacije nad matricama kraće traju. U takvom slučaju, učitavanje matrica zauzima veći procenat celokupnog vremena, pa je i samo teorijsko ubrzanje manje. Kao i u *Python* jakom skaliranju, teorijsko ubrzanje raste brzo do 16. procesa, a onda usporava. S druge strane, naše ubrzanje uz male oscilacije, prati trend teorijskog ubrzanja. Prikazaćemo još jedan grafik da dobijemo bolje informacije o našem ubrzanju.



Slika 8: Naše ubrzanje pri jakom skaliranju u Rust-u

Sa grafika na slici 8, vidimo da je najveća vrednost ubrzanja pri 16 procesa, posle čega ono blago opadne, pa blago poraste. Trend na početku je identičan - ubrzanje skoči, pa blago opadne, te opet skoči. Svakako, ove oscilacije nisu velike i možemo reći da naše ubrzanje prati trend teorijskog. Takođe, odnos našeg i teorijskog ubrzanja je između 70% i 90%, što je zaista dobar rezultat.

U poređenju sa grafikom pri *Python* jakom skaliranju, rezultati su značajno bolji. Naše ubrzanje manje odstupa od teorijskog, kako po pitanju samih vrednosti, tako i po pitanju trenda kretanja.

U tabeli 7, prikazane su srednje vrednosti izvršavanja za date ulazne parametre. Za svaku kombinaciju parametara, rešenja su pokretana bar 30 puta.

dimension	no. of processes	t_not_par [s]	t_s [s]	t_p [s]	real speedup	theory speedup
3600x3600	1.12851	2.41058	1	2.4341	0.9899	1
3600x3600	1.12851	2.41058	4	1.64065	1.46928	1.66358
3600x3600	1.12851	2.41058	9	1.681517	1.43357	1.8966
3600x3600	1.12851	2.41058	16	1.58062	1.525088	1.99445

3600x3600	1.12851	2.41058	25	1.61962	1.48836	2.04321
3600x3600	1.12851	2.41058	36	1.60424	1.50263	2.07071

Tabela 7: Srednja vremena izvršavanja za jako skaliranje u Rust-u

U tabeli 8, prikazane su vrednosti standardne devijacije za vremena izvršavanja.

dimension	no. of processes	t_not_par	t_s	t_p
3600x3600	1	0.04007	0.064941	0.05104
3600x3600	4	0.04007	0.064941	0.030124
3600x3600	9	0.04007	0.064941	0.015768
3600x3600	16	0.04007	0.064941	0.017023
3600x3600	25	0.04007	0.064941	0.01877
3600x3600	36	0.04007	0.064941	0.02516

Tabela 8: Standardne devijacije vremena izvršavanja za jako skaliranje u Rust-u

Iz tabele primećujemo da su vrednosti standardnih devijacija niske, naročito u poređenju sa vrednostima u *Python* jakom skaliranju. Ovo opet objašnjava i zašto naše ubrzanje pri *Rust* jakom skaliranju ima bolji trend i vrednosti nego ubrzanje pri *Python* skaliranju.

Diskusija

Na osnovu prethodno urađenih testova i skaliranja, možemo utvrdi da:

- *Rust* jezik značajno brže izvršava sekvencijalno rešenje nego *Python* jezik
- *Rust* jezik značajno brže izvršava paralelno rešenje nego *Python* jezik, te da bolje koristi paralelizam i veći broj procesa
- Naša ubrzanja pri *Rust* jeziku više prate vrednosti i trend teorijskih ubrzanja, nego što je slučaj u *Python* jeziku

Među svim skaliranjima, najbolje rezultate smo dobili pri jakom skaliranju u *Rust* jeziku.

U tabeli 9, prikazana su najveća ubrzanja za svaki tip skaliranja i jezik u kom je pokrenuto skaliranje.

Jezik	Tip	Broj procesa	Ubrzanje
Python	Weak scaling	9	1.28186
Python	Strong scaling	4	1.63628
Rust	Weak scaling	36	1.45305
Rust	Strong scaling	16	1.525088

Tabela 9: Najbolje dobijana ubrzanja