Druga domaća zadaća

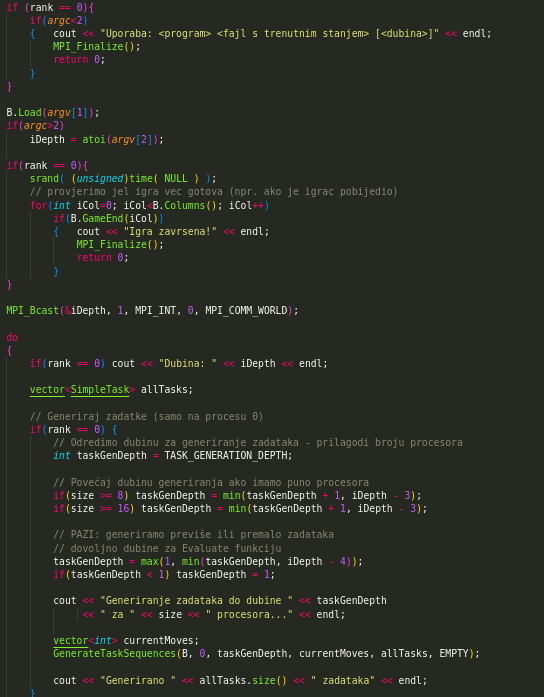
Igra Connect4

## Implementacija

Prije svega bitno je napomenuti da sam kao podlogu koristio program dostupan u repozitoriju predmeta (connect4) te sam paralelizirao dio koji evaluira trenutne korake na način da sam dijelio stupce procesima.

Prva prepreka koju sam morao rješiti je inicijalizacija posla u glavnom (rank 0) procesu.

To sam ostvario na sljedeći način:

Glavni proces prvo provjerava argumente naredbenog retka, učitava trenutno stanje igre iz datoteke i inicijalizira generator slučajnih brojeva. Ovdje je također vidljivo da se iDepth *broadcasta* svim procesima u svrhu sinkronizacije.

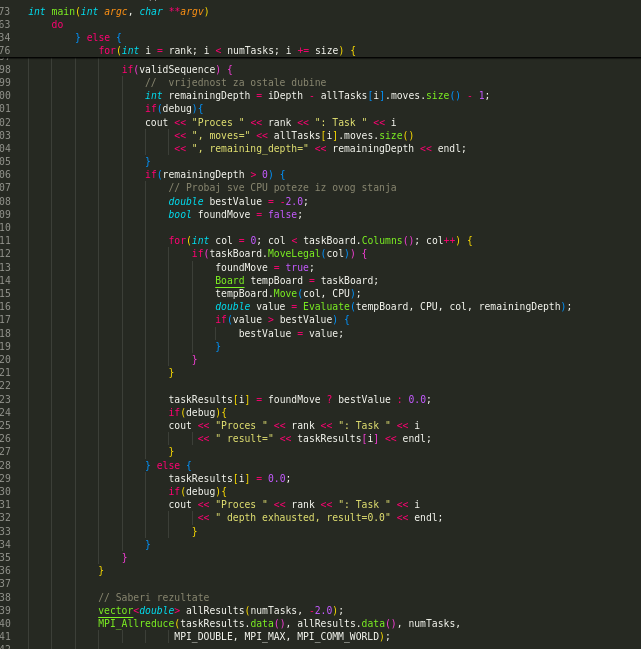
Sljedeća prepreka je bila podjela posla ostalim procesima.

Na slici je vidljivo kako je to razrješeno:



Početak je do-while petlja koju smo imali kada smo koristili sekvencijalni algoritam sa istim uvjetima. Ovdje je jasno vidljivo kako radimo prodjelu poslova procesima za recimo 6x7 polje i 4 procesa proces 0 će dobiti na evaluaciju retke 0 i 4, proces 1 će dobiti retke 1 i 5, proces 2 retke 2 i 6, proces 3 redak 3 tj. svi će dobiti u ovom slučaju ili 2 retka ili 1 za evaluaciju što je puno bolje nego da jedan proces radi sve.

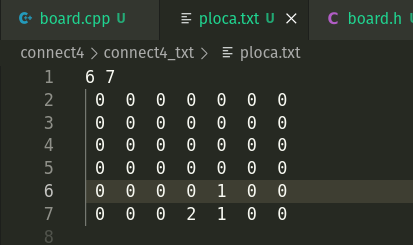
Kada je sve izračunato morao sam pronaći način da sva ta rješenja kombiniram i dođem do rješenja.

Koristi se MPI\_Allreduce operacija. Prvo se prikupljaju informacije o broju rezultata od svakog procesa, a zatim se svi rezultati prikupljaju u jedan vektor. Svaki rezultat se kodira kao tri integera (stupac, vrijednost kao double pretvorena u int, i legalni flag).

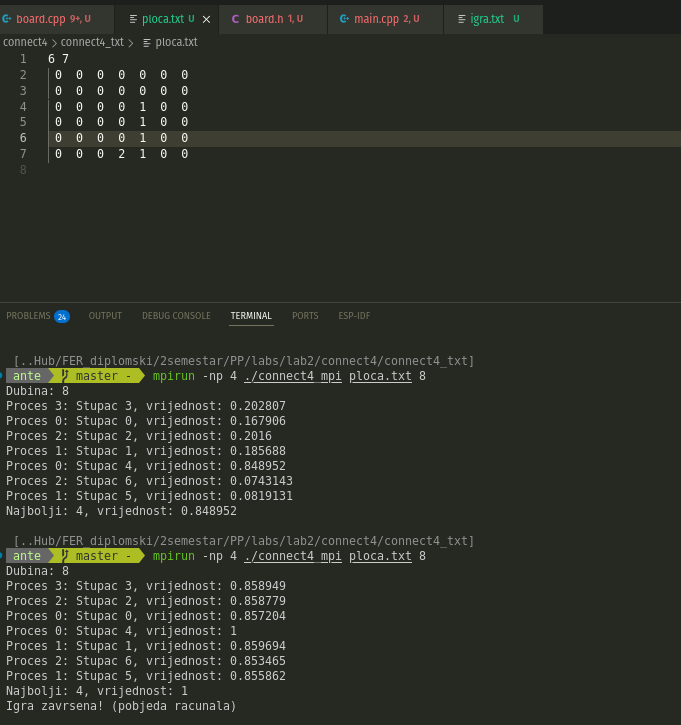
Za kraj još bih volio samo prikazati kako se „pobjednik” stupaca računao.

Još je bitno napomenuti da za 1 proces imamo conditional na početku koji radi obični sekvencijalni run (kod koji smo dobili na početku) te da ukoliko imamo više procesa od stupaca, procesima koji ne bi dobili nijedan zadatak dajemo dummy zadatke kako bi tok programa ostao isti.

Za primjer pobjede u 2 koraka sam modificirao ploca.txt :



Kao što vidimo potrebno je računalu 2 koraka do pobjede. Sada ćemo iskoristiti program da prikažemo ta zadnja 2 koraka:

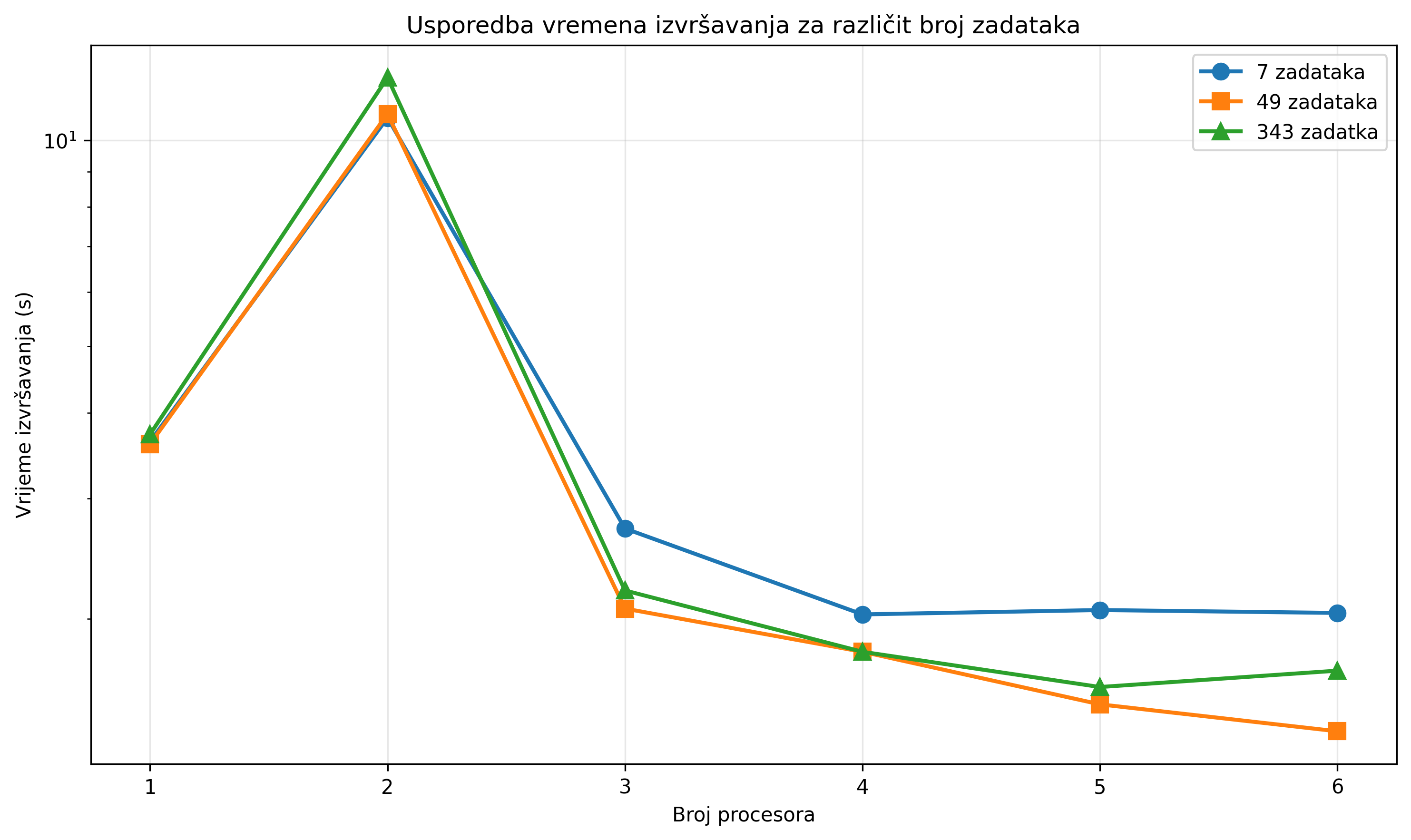
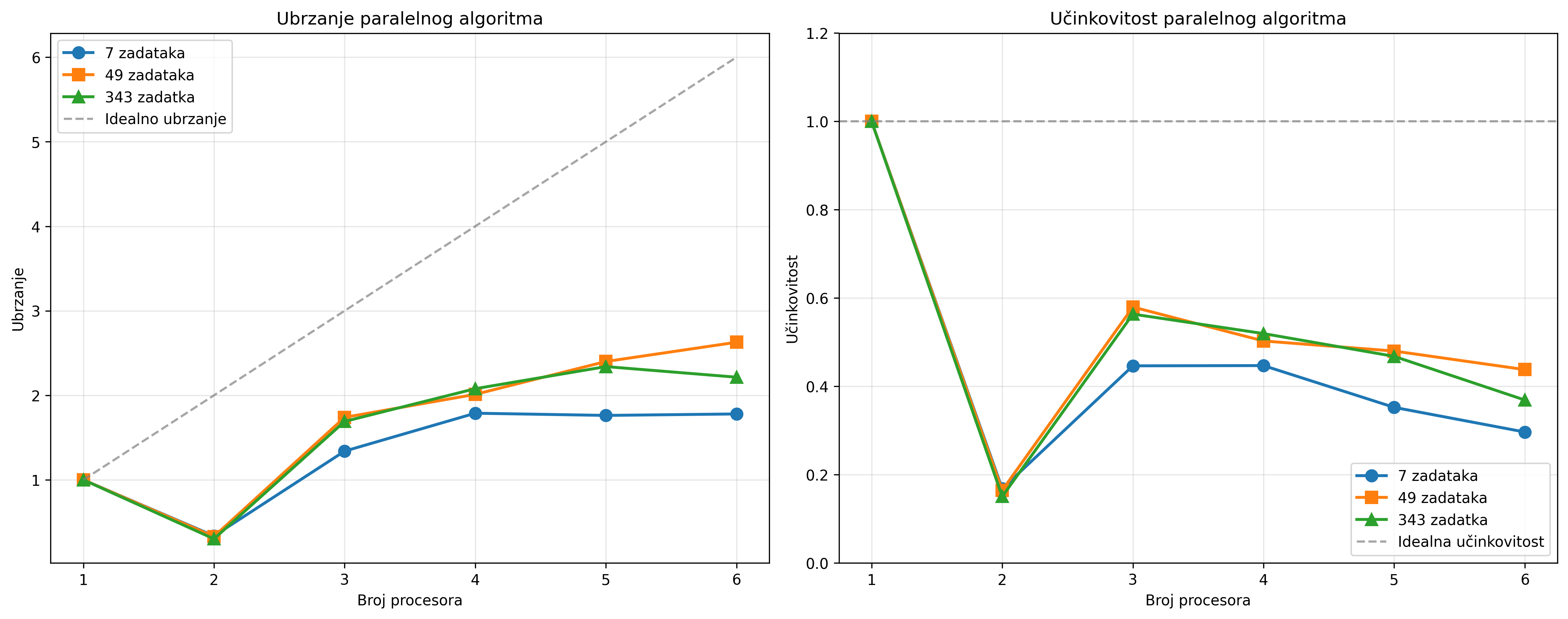
Na kraju smo uspješno pobijedili koristeći 4 procesora i dubinu 8.

Smatram da je bitno nadodati kako generiram zadatke kao što je vidljivo iz priložene slike.



## Kvantitativna analiza

*Upute: U ovom dijelu potrebno je priložiti* ***tablice s rezultatima mjerenja te******grafove ubrzanja i učinkovitosti*** *za tri različita scenarija: kada paralelni algoritam ima 7, 49 i 343 zadatka (uz aglomeraciju na dubini 1, 2 i 3). Mjerenja treba provesti tako da je najmanje mjereno trajanje (za 8 procesora) reda veličine barem* ***nekoliko sekundi*** *(definirajte potrebnu dubinu pretraživanja). Uz grafove, dodajte kratki komentar koji opisuje kako broj zadataka utječe na ubrzanje i učinkovitost (uzevši u obzir utjecaj zrnatosti zadataka, komunikacijskog overhead-a, te udjela programa koji se ne može paralelizirati).*



Prikazi grafova vidljivi gore jasno pokazuju promjene i odnos granularnosti i komunikacijskog overhead-a. Iz mojih podataka je vidljivo da se prilikom pokretanja sa 2 procesa javlja veliki zastoj – to je anomalija koju sam nisam dokučio zašto se javlja no vidimo da se za ostatak program ponaša „normalno”.

Maksimalno ubrzanje:

7 zadataka: 1.79x (s 4 procesora)

49 zadataka: 2.63x (s 6 procesora)

343 zadatka: 2.34x (s 5 procesora)

Najbolja učinkovitost (osim 1 procesora):

7 zadataka: 0.45 (s 4 procesora)

49 zadataka: 0.58 (s 3 procesora)

343 zadatka: 0.56 (s 3 procesora)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Broj procesora | 7 zadataka - Vrijeme (s) | 7 zadataka - Ubrzanje | 7 zadataka - Učinkovitost | 49 zadataka - Vrijeme (s) | 49 zadataka - Ubrzanje | 49 zadataka - Učinkovitost | 343 zadatka - Vrijeme (s) | 343 zadatka - Ubrzanje | 343 zadatka - Učinkovitost |
| 1 | 3.63 | 1 | 1 | 3.6 | 1 | 1 | 3.72 | 1 | 1 |
| 2 | 10.79 | 0.34 | 0.17 | 10.93 | 0.33 | 0.16 | 12.35 | 0.3 | 0.15 |
| 3 | 2.71 | 1.34 | 0.45 | 2.07 | 1.74 | 0.58 | 2.2 | 1.69 | 0.56 |
| 4 | 2.03 | 1.79 | 0.45 | 1.79 | 2.01 | 0.5 | 1.79 | 2.08 | 0.52 |
| 5 | 2.06 | 1.76 | 0.35 | 1.5 | 2.4 | 0.48 | 1.59 | 2.34 | 0.47 |
| 6 | 2.04 | 1.78 | 0.3 | 1.37 | 2.63 | 0.44 | 1.68 | 2.21 | 0.37 |

- **7 zadataka (gruba zrnatost)**: Najgore skaliranje zbog nedovoljne paralelizacije

Maksimalno ubrzanje ~1.8x, što ukazuje na ograničenu mogućnost paralelizacije

- **49 zadataka (srednja zrnatost)**: Najbolje skaliranje i učinkovitost

Maksimalno ubrzanje ~2.6x s dobrom učinkovitošću na višim brojevima procesora

- **343 zadatka (fina zrnatost)**: Umjereno skaliranje zbog komunikacijskog overhead-a

Maksimalno ubrzanje ~2.3x, ali s padom učinkovitosti na 6 procesora