Univerzitet u Beogradu - Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



Domaći zadatak – OPENMP

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni saradnici: | Student: |
| prof. dr Marko Mišić | Vuk Lužanin 29/2022 |

Beograd, april 2025.

# Sadržaj

[Sadržaj 2](#_Toc198388880)

[1. Napomene 4](#_Toc198388881)

[2. Problem 1 - Poasonova jednačina 5](#_Toc198388882)

[2.1. Tekst problema 5](#_Toc198388883)

[2.1.1. Diskusija 5](#_Toc198388884)

[2.1.2. Način paralelizacije [FOR COLLAPSE] 5](#_Toc198388885)

[2.2. Rezultati 6](#_Toc198388886)

[2.2.1. Logovi izvršavanja 6](#_Toc198388887)

[2.2.1.1. Logovi 2D problema 6](#_Toc198388888)

[2.2.1.2. Grafici ubrzanja 2D problema 8](#_Toc198388889)

[2.2.1.3. Logovi 3D problema 10](#_Toc198388890)

[2.2.1.4. Grafici ubrzanja 3D problema 12](#_Toc198388891)

[2.2.1.5. Diskusija dobijenih rezultata 14](#_Toc198388892)

[2.2.2. Način paralelizacije [FOR + schedule(dynamic)] 14](#_Toc198388893)

[2.3. Rezultati 14](#_Toc198388894)

[2.3.1. Logovi izvršavanja 15](#_Toc198388895)

[2.3.1.1. Logovi 1D problema 15](#_Toc198388896)

[2.3.1.2. Grafici ubrzanja 1D problema 17](#_Toc198388897)

[2.3.1.3. Logovi 2D problema 19](#_Toc198388898)

[2.3.1.4. Grafici ubrzanja 2D problema 21](#_Toc198388899)

[2.3.1.5. Logovi 3D problema 23](#_Toc198388900)

[2.3.1.6. Grafici ubrzanja 3D problema 25](#_Toc198388901)

[2.3.1.7. Diskusija dobijenih rezultata 27](#_Toc198388902)

[3. Problem 2 - Poasonova jednačina 28](#_Toc198388903)

[3.1. Tekst problema 28](#_Toc198388904)

[3.2. Delovi koje treba paralelizovati 28](#_Toc198388905)

[3.2.1. Diskusija 28](#_Toc198388906)

[3.2.2. Način paralelizacije TASK + ATOMIC direktiva 28](#_Toc198388907)

[3.3. Rezultati 29](#_Toc198388908)

[3.3.1. Logovi izvršavanja 1D problema 29](#_Toc198388909)

[3.3.2. Grafici ubrzanja 1D problema 31](#_Toc198388910)

[3.3.3. Logovi izvršavanja 2D problema 33](#_Toc198388911)

[3.3.4. Grafici ubrzanja 2D problema 35](#_Toc198388912)

[3.3.5. Logovi izvršavanja 3D problema 37](#_Toc198388913)

[3.3.6. Grafici ubrzanja 3D problema 39](#_Toc198388914)

[3.3.7. Diskusija dobijenih rezultata 41](#_Toc198388915)

[3.3.8. Način paralelizacije TASK + locks distributed across multiple threads 41](#_Toc198388916)

[3.4. Rezultati 42](#_Toc198388917)

[3.4.1. Logovi izvršavanja 1D problema 42](#_Toc198388918)

[3.4.2. Grafici ubrzanja 1D problema 44](#_Toc198388919)

[3.4.3. Logovi izvršavanja 2D problema 46](#_Toc198388920)

[3.4.4. Grafici ubrzanja 2D problema 48](#_Toc198388921)

[3.4.5. Logovi izvršavanja 3D problema 50](#_Toc198388922)

[3.4.6. Grafici ubrzanja 3D problema 52](#_Toc198388923)

[3.4.7. Diskusija dobijenih rezultata 54](#_Toc198388924)

# Napomene

Projekat se nalazi na Githubu, na linku: [**Feynman-Kac-Parallelization-Research**](https://github.com/Vuk-Luzanin/Feynman-Kac-Parallelization-Research)**.** Kako bi ga pokrenuli, potrebno ga je klonirati i zatim pratiti uputstvo napisano u **README.md** fajlu.

Svi testovi su pokretani na fakultetskom **rtidev5** računaru, a na njemu se projekat nalazi u direktorijumu:

**/home/lv220029d/Istrazivanje/Feynman-Kac-Parallelization-Research/**

Odgovarajuće implementacije se mogu pronaći u folderu **src/OpenMP/.** Svaki fajl sadrži po 4 različita rešenja koja su u ovom dokumentu razmatrana. U prvoj sekciji dokumenta je reč o prva dva rešenja, dok je u drugoj sekciji dokumenta reč o druga dva.

Data su objašnjenja za 3D verziju, bez umanjenja opštosti (sve dimenzije problema sadrže isto/slično rešenje, samo što je ono prilagođeno datoj verziji), dok su grafici ubrzanja priloženi za sve tri dimenzije problema

# Problem 1 - Poasonova jednačina

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 1.

## Tekst problema

Paralelizovati program koji vrši izračunavanje 1D, 2D i 3D [Poasonove jednačine](https://en.wikipedia.org/wiki/Poisson%27s_equation) korišćenjem [Feynman-Kac](https://en.wikipedia.org/wiki/Feynman%E2%80%93Kac_formula) algoritma. Algoritam stohastički računa rešenje parcijalne diferencijalne jednačine krenuvši N puta iz različitih tačaka domena. Tačke se kreću po nasumičnim putanjama i prilikom izlaska iz granica domena kretanje se zaustavlja računajući dužinu puta do izlaska. Proces se ponavlja za svih N tačaka i konačno aproksimira rešenje jednačine. Program se nalazi u datoteci **./src/OpenMP/** u arhivi koja je priložena uz ovaj dokument. Delovi koje treba paralelizovati

### Diskusija

Implementacije su odvojene u zasebne funkcije, obeležene kao **feynman\_i(),** a koje implementiraju sam *Feynman-Kac* algoritam**.** Promenljiva **i** u nazivu funkcije uzima vrednosti [1, 2], za rešenja pomoću **for** worksharing direktive, pa je u nastavku objašnjenje svakog od rešenja. Specifičnost ovog algoritma je nasumično generisanje putanja tačaka, za šta se koristi generator pseudoslučajnih brojeva po uniformnoj raspodeli u funkciji **r8\_uniform\_01()**. Razmatrana je mogućnost kretanja čestice ne samo diskretno po rešetki za korak od ±stepsz u svakoj dimenziji, već i pomeranja za manje korake unutar intervala između -stepsz i stepsz, kao i korišćenje ugrađene **rand()** funkcije za generisanje pseudo/nasumičnih brojeva. Ova implementacija za sada nije testirana zbog zahteva za dodatnim vremenom izvođenja, ali može biti aktivirana korišćenjem kompilacionog flega -DSMALL\_STEP. Postoji *seed* za generisanje ovih pseudo-nasumičnih brojeva, ali on ne pravi problem za paralelizaciju, jer svaka nit može da ima svoj sopstveni *seed* i brojevi koji se generišu će nastaviti da budu uniformni. Ukoliko je ovaj *seed* deljen, zaključili smo da se ne garantuje uniformnost raspodele.

### Način paralelizacije [FOR COLLAPSE]

**Napomena: u 1D rešenju se korišćenje for *worksharing* direktive sa collapse opcijom svodi na for *worksharing* direktivu bez opcija, tako da indeks funkcije počinje od 2.**

Paralelizacija je izvršena jednom **for** *worksharing* direktivom sa **collapse(3)** opcijom nad tri prve petlje. Ovom prilikom je računanje brojeva **x** i **y** premešteno unutar iste petlje kao računanje broja **z**, zbog zahteva ove direktive, ne izazivajući naročito usporenje u izvršavanju.

Kako različite niti počinju sa istim *seed*-om, zaključeno je da će generisati iste brojeve po uniformnoj raspodeli. Kada je pokušano da se svakoj niti dodeli različit *seed*, uvećavanjem podrazumevanog *seed*-a za identifikator trenutne niti, dobijeno je da je *root-mean-square error* malo bliži onom od sekvencijalnog algoritma, pa je tako i ostavljeno.

Sinhronizacija između niti postignuta je sabirajućom redukcijom po promenljivama **err** i **n\_inside**.

## Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije ovog problema.

### Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

### Logovi 2D problema

**TEST: func=0, N=1000, num\_threads=1**

**1000 0.022755 0.995499**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=0, N=1000, num\_threads=2**

**1000 0.022633 0.498286**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=0, N=1000, num\_threads=4**

**1000 0.023235 0.391813**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=0, N=1000, num\_threads=8**

**1000 0.023249 0.221893**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=0, N=1000, num\_threads=16**

**1000 0.022652 0.136212**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=0, N=5000, num\_threads=1**

**5000 0.021163 4.957427**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=0, N=5000, num\_threads=2**

**5000 0.021191 2.480981**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=0, N=5000, num\_threads=4**

**5000 0.021471 1.943766**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=0, N=5000, num\_threads=8**

**5000 0.021752 1.162587**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=0, N=5000, num\_threads=16**

**5000 0.021565 0.785023**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=0, N=10000, num\_threads=1**

**10000 0.021240 9.898465**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=0, N=10000, num\_threads=2**

**10000 0.021161 4.960533**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=0, N=10000, num\_threads=4**

**10000 0.021484 3.884980**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=0, N=10000, num\_threads=8**

**10000 0.021674 2.472048**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=0, N=10000, num\_threads=16**

**10000 0.021792 1.619796**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=0, N=20000, num\_threads=1**

**20000 0.021439 19.821658**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=0, N=20000, num\_threads=2**

**20000 0.021273 9.926953**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=0, N=20000, num\_threads=4**

**20000 0.021485 7.937330**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=0, N=20000, num\_threads=8**

**20000 0.021552 4.968104**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=0, N=20000, num\_threads=16**

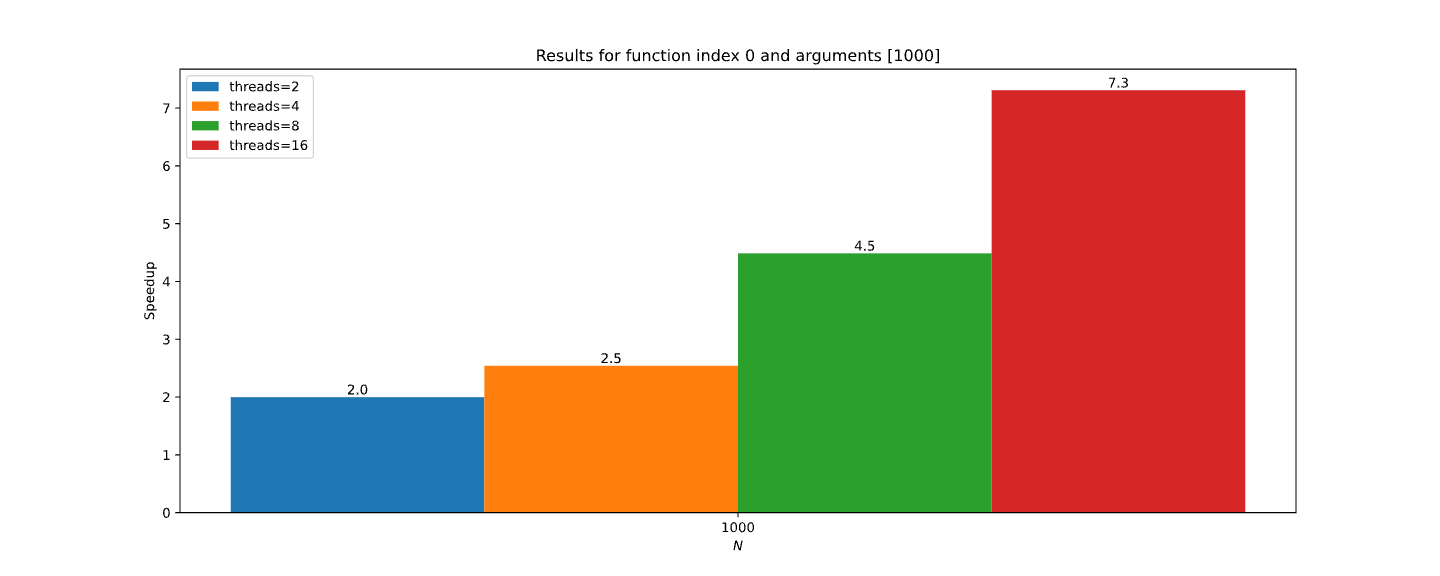
**20000 0.021485 3.224030**

**TEST END**

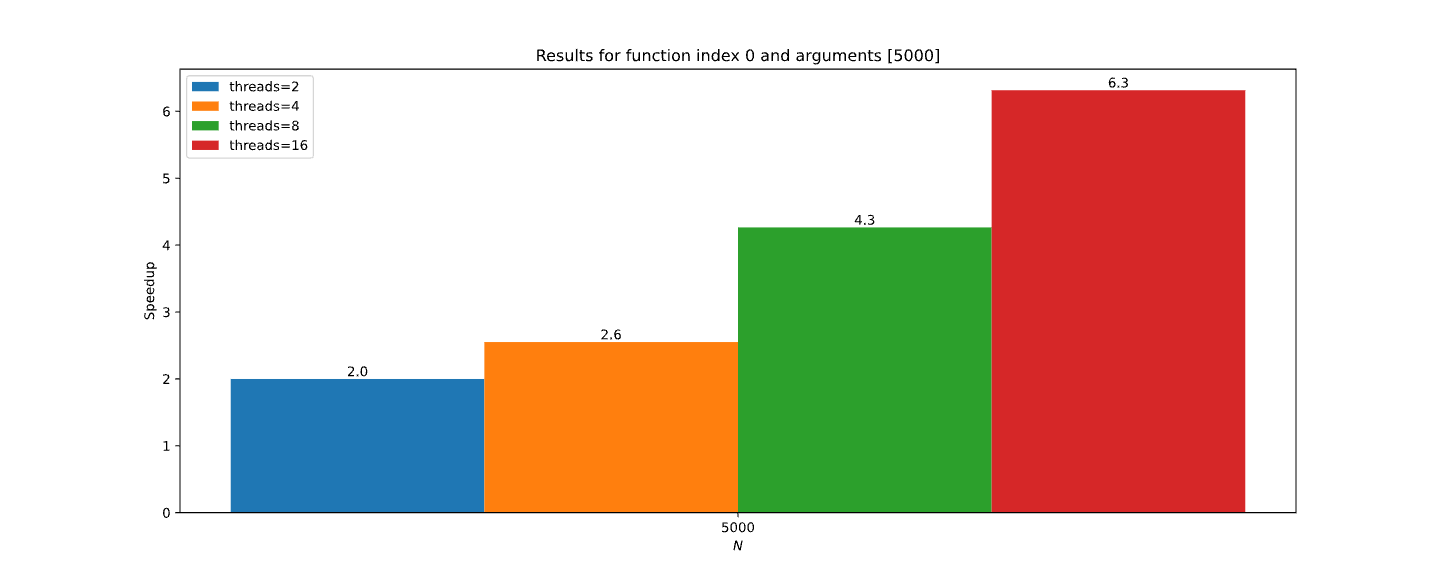
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 2D problema

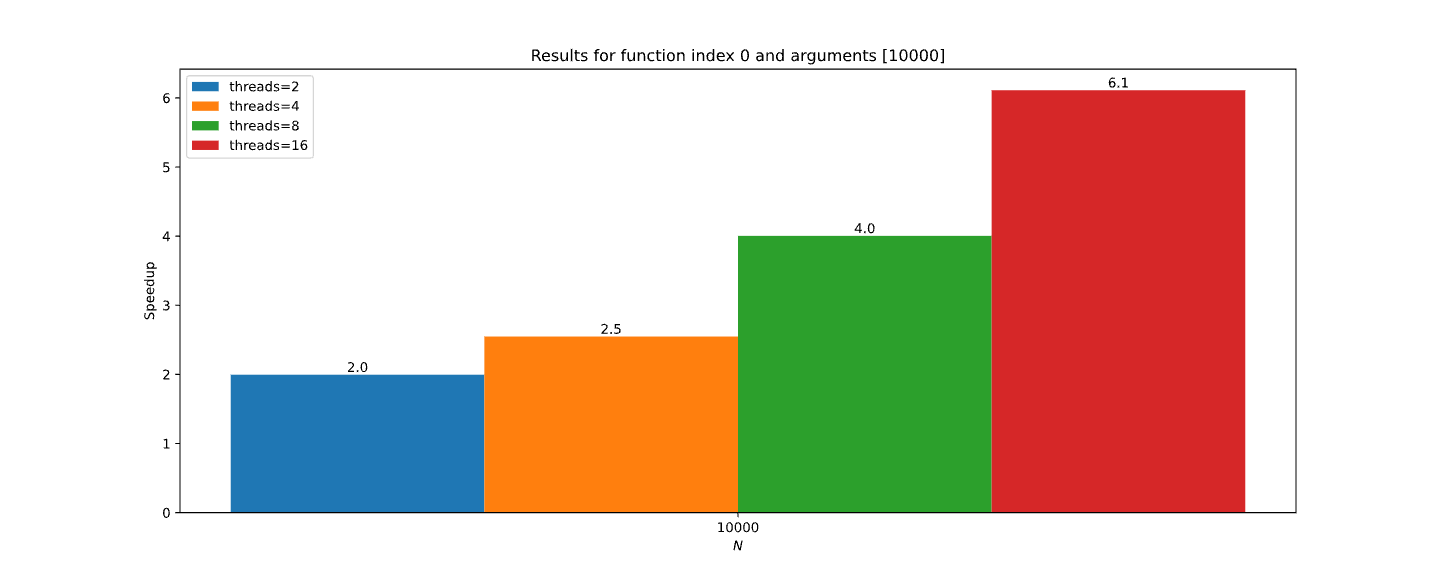
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.

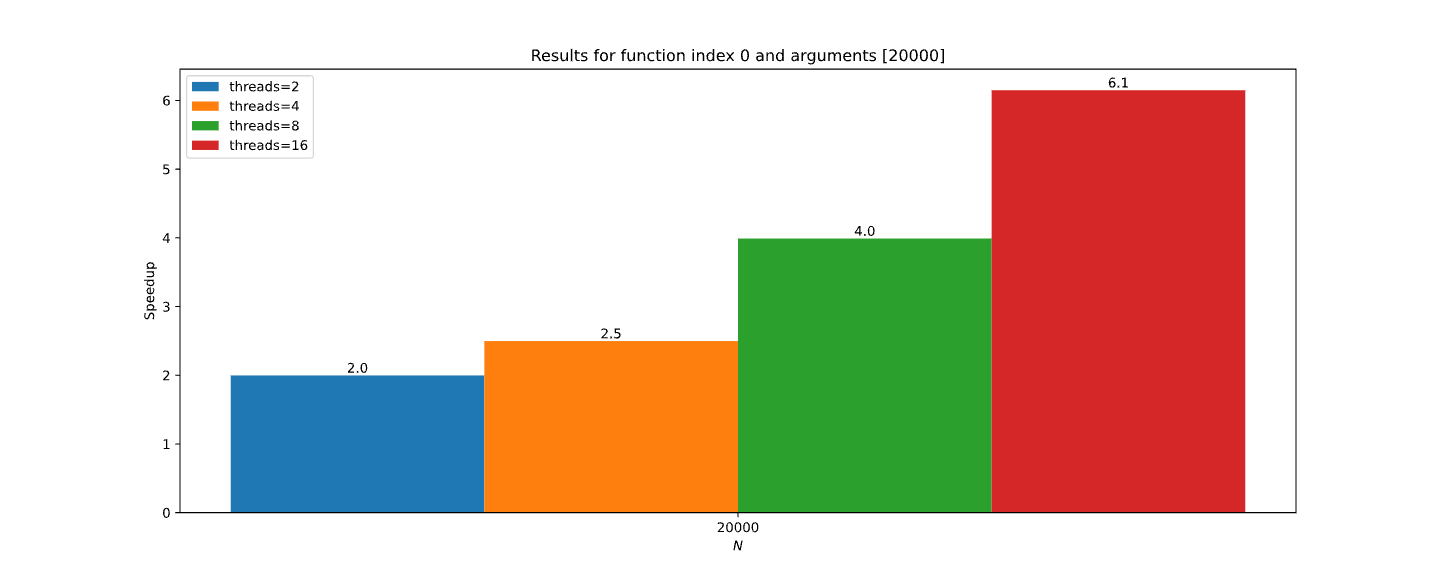


**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi 3D problema

**TEST: func=0, N=1000, num\_threads=1**

**1000 0.021717 3.046836**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=0, N=1000, num\_threads=2**

**1000 0.021614 1.528430**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=0, N=1000, num\_threads=4**

**1000 0.021846 1.263588**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=0, N=1000, num\_threads=8**

**1000 0.021634 0.807631**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=0, N=1000, num\_threads=16**

**1000 0.022469 0.523907**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=0, N=5000, num\_threads=1**

**5000 0.021273 15.264332**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=0, N=5000, num\_threads=2**

**5000 0.021125 7.661872**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=0, N=5000, num\_threads=4**

**5000 0.021317 6.612114**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=0, N=5000, num\_threads=8**

**5000 0.020931 4.179080**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=0, N=5000, num\_threads=16**

**5000 0.020941 2.494137**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=0, N=10000, num\_threads=1**

**10000 0.021100 30.591238**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=0, N=10000, num\_threads=2**

**10000 0.020984 15.456571**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=0, N=10000, num\_threads=4**

**10000 0.021043 13.145688**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=0, N=10000, num\_threads=8**

**10000 0.021000 8.240244**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=0, N=10000, num\_threads=16**

**10000 0.020995 5.323410**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=0, N=20000, num\_threads=1**

**20000 0.021027 61.031435**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=0, N=20000, num\_threads=2**

**20000 0.020978 30.858416**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=0, N=20000, num\_threads=4**

**20000 0.021046 26.159093**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=0, N=20000, num\_threads=8**

**20000 0.020947 16.413737**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=0, N=20000, num\_threads=16**

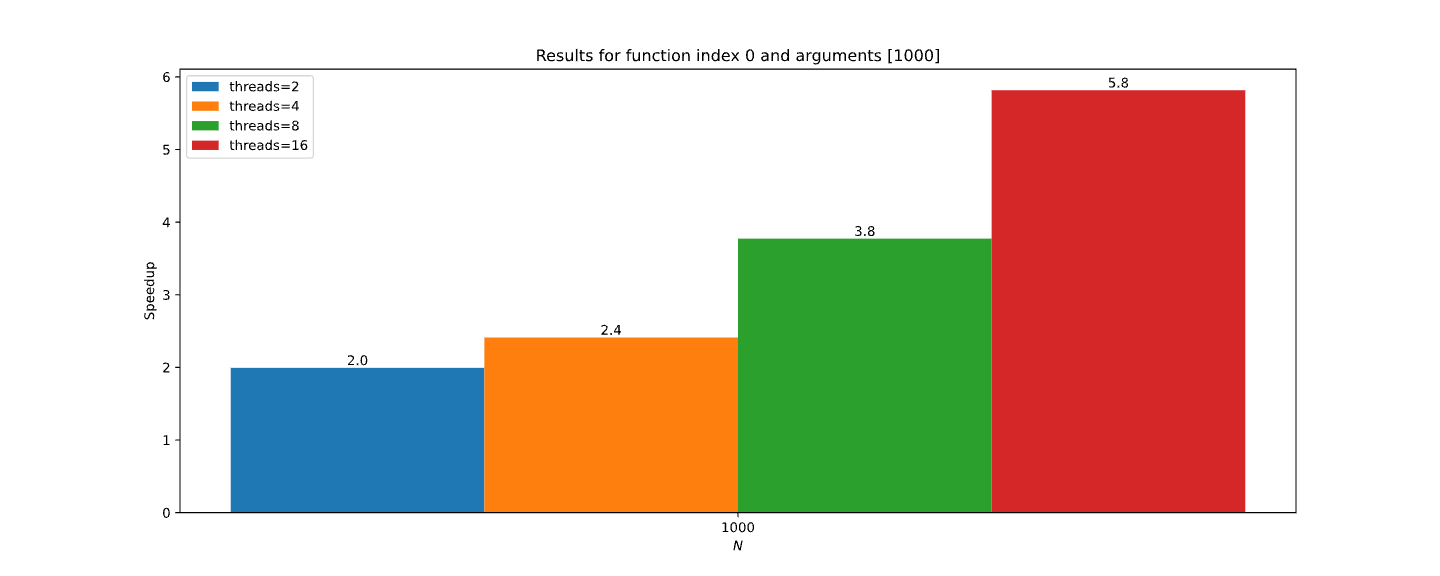
**20000 0.020964 10.313700**

**TEST END**

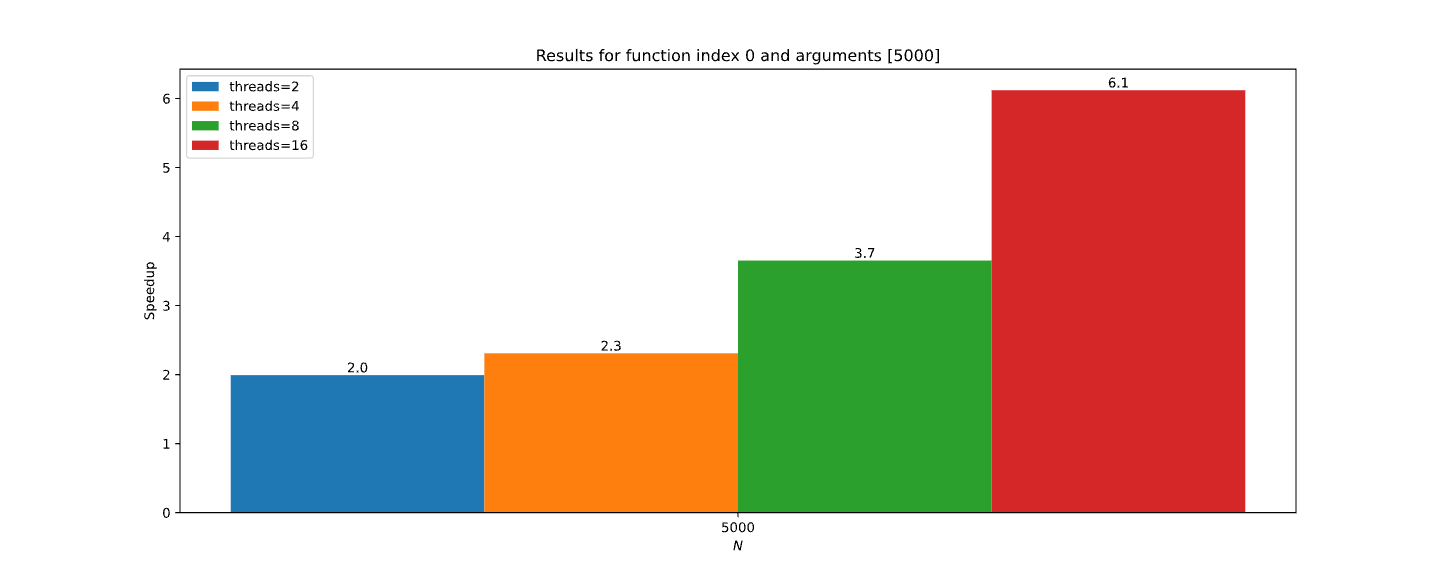
**Izvršavanje za argumente 20000 i šestnaest niti**

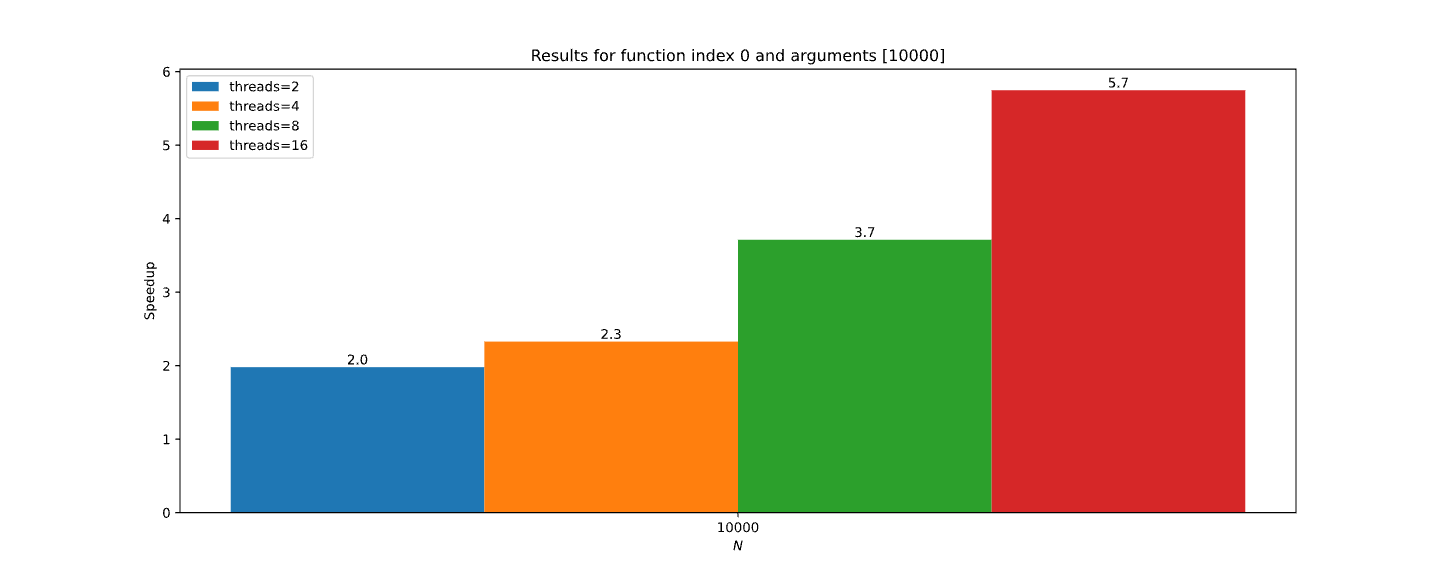
### Grafici ubrzanja 3D problema

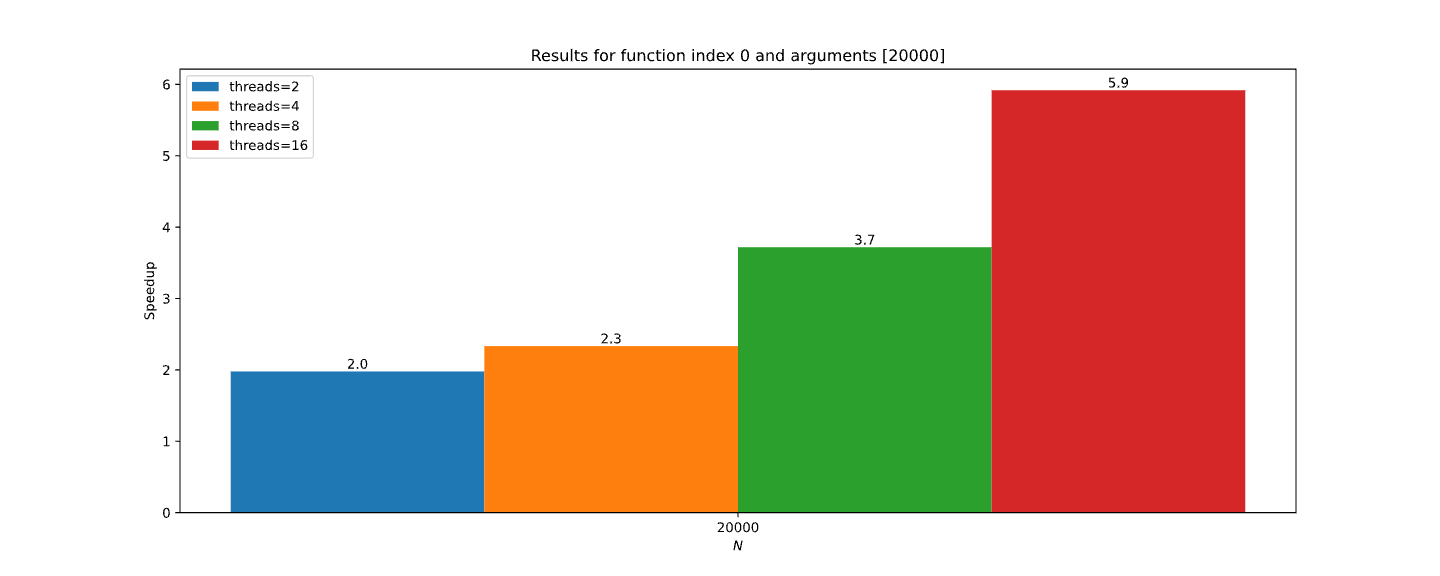
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 10000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** (kao što je navedeno u tekstu zadatka). Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati.

### Način paralelizacije [FOR + schedule(dynamic)]

Paralelizacija je izvršena jednom **for** *worksharing* direktivom sa **schedule(dynamic)** opcijom nad najspoljnijom petljom. Kako je vreme izvršavanja unutrašnje petlje neravnomerno između niti, empirijski je zaključeno da će **dynamic** raspored dati najbolje rezultate, budući da će niti koje imaju manje posla, uzeti veći broj iteracija za obradu.

Ostatak implementacije je gotovo isti kao kod prethodnog rešenja.

Sinhronizacija između niti takođe je postignuta sabirajućom redukcijom po promenljivama **err** i **n\_inside**.

## Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije ovog problema.

### Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

### Logovi 1D problema

**TEST: func=0, N=1000, num\_threads=1**

**1000 0.009362 1.320087**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=0, N=1000, num\_threads=2**

**1000 0.007205 0.688187**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=0, N=1000, num\_threads=4**

**1000 0.008971 0.360296**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=0, N=1000, num\_threads=8**

**1000 0.006166 0.212293**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=0, N=1000, num\_threads=16**

**1000 0.006447 0.257275**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=0, N=5000, num\_threads=1**

**5000 0.003243 6.737046**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=0, N=5000, num\_threads=2**

**5000 0.004111 3.427073**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=0, N=5000, num\_threads=4**

**5000 0.003185 1.764354**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=0, N=5000, num\_threads=8**

**5000 0.004699 1.129232**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=0, N=5000, num\_threads=16**

**5000 0.003618 1.132560**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=0, N=10000, num\_threads=1**

**10000 0.002901 13.415577**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=0, N=10000, num\_threads=2**

**10000 0.002852 6.930533**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=0, N=10000, num\_threads=4**

**10000 0.002422 3.576365**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=0, N=10000, num\_threads=8**

**10000 0.002504 2.555768**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=0, N=10000, num\_threads=16**

**10000 0.002442 2.564570**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=0, N=20000, num\_threads=1**

**20000 0.002446 26.884451**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=0, N=20000, num\_threads=2**

**20000 0.002761 14.132822**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=0, N=20000, num\_threads=4**

**20000 0.002395 7.465715**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=0, N=20000, num\_threads=8**

**20000 0.002404 5.026315**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=0, N=20000, num\_threads=16**

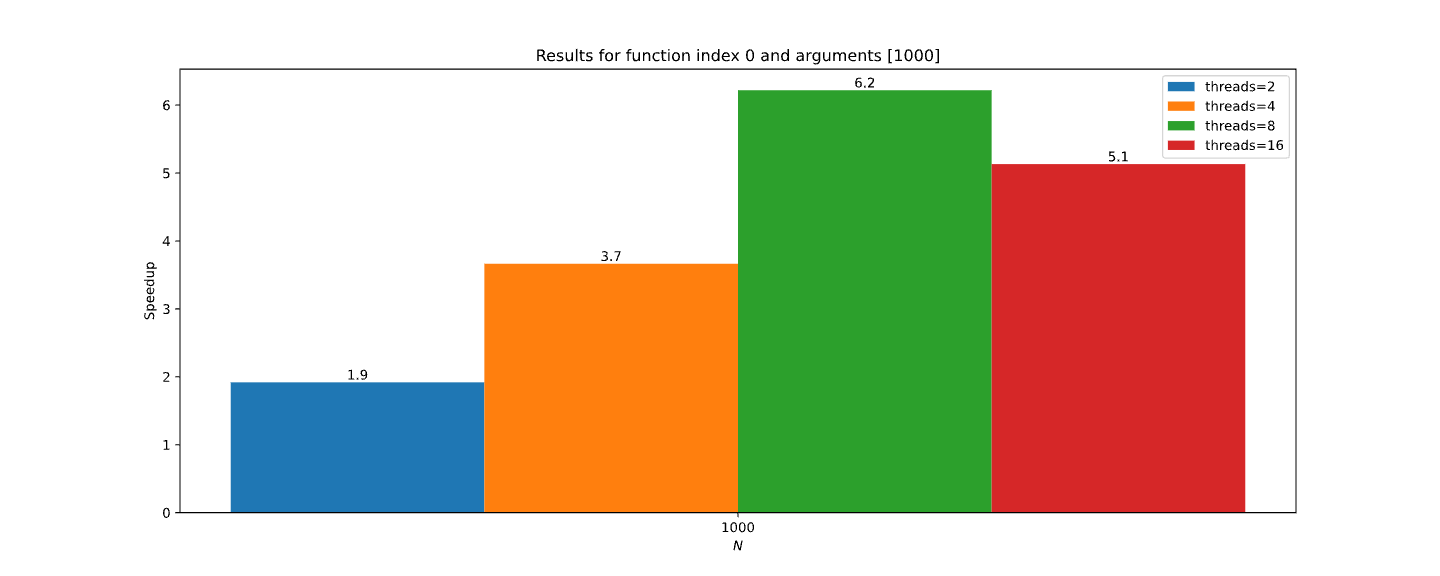
**20000 0.002423 5.190004**

**TEST END**

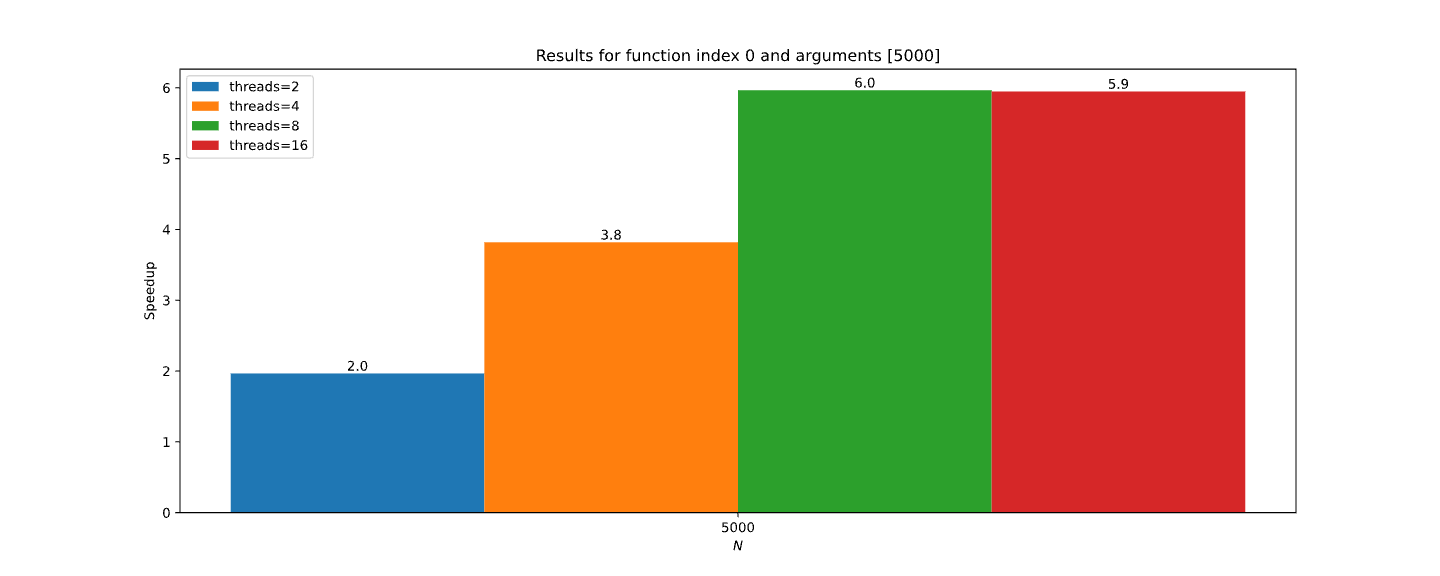
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 1D problema

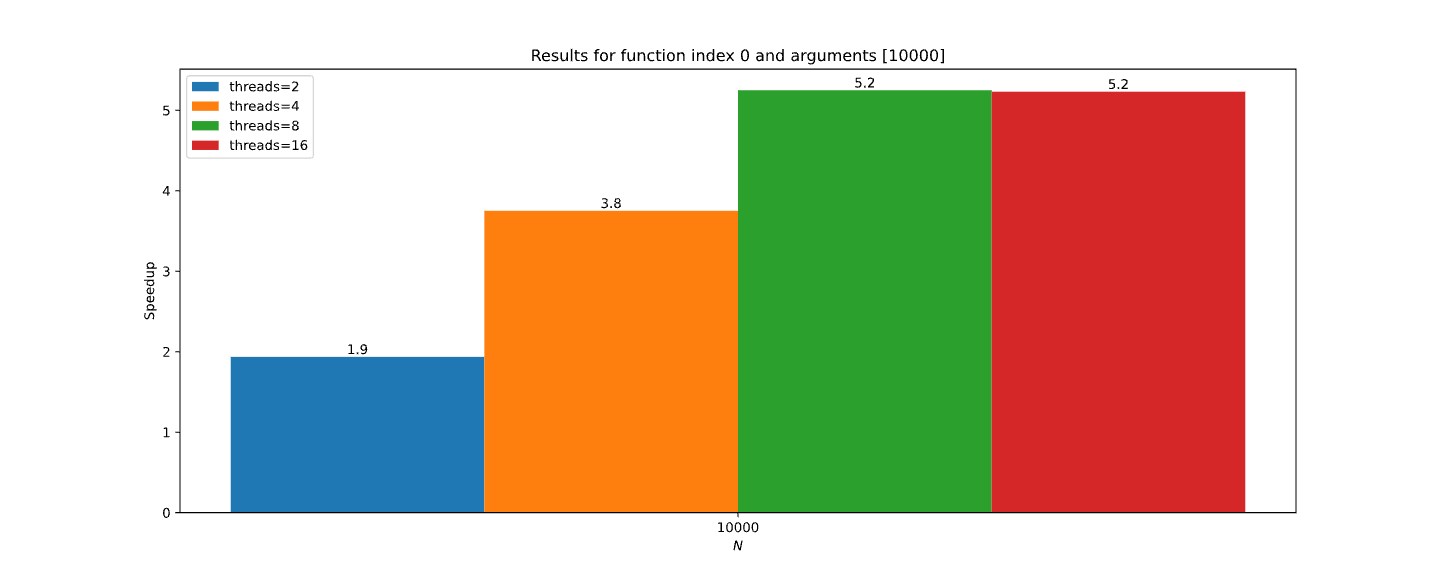
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.

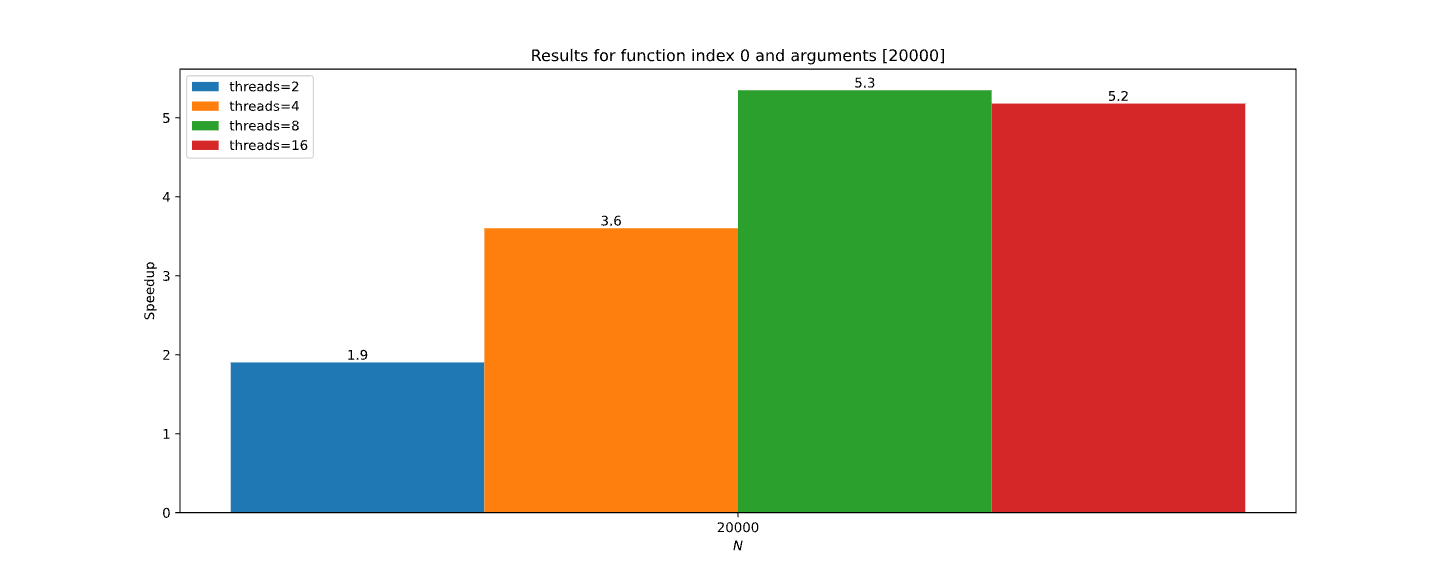


**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi 2D problema

**TEST: func=1, N=1000, num\_threads=1**

**1000 0.022755 0.991304**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=1, N=1000, num\_threads=2**

**1000 0.022589 0.497080**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=1, N=1000, num\_threads=4**

**1000 0.023187 0.256413**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=1, N=1000, num\_threads=8**

**1000 0.022579 0.147595**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=1, N=1000, num\_threads=16**

**1000 0.021948 0.173404**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=1, N=5000, num\_threads=1**

**5000 0.021163 4.937234**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=1, N=5000, num\_threads=2**

**5000 0.021217 2.472245**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=1, N=5000, num\_threads=4**

**5000 0.021906 1.281648**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=1, N=5000, num\_threads=8**

**5000 0.021947 0.896379**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=1, N=5000, num\_threads=16**

**5000 0.021616 0.811666**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=1, N=10000, num\_threads=1**

**10000 0.021240 9.862222**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=1, N=10000, num\_threads=2**

**10000 0.021708 4.957641**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=1, N=10000, num\_threads=4**

**10000 0.021694 2.677300**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=1, N=10000, num\_threads=8**

**10000 0.021632 1.784100**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=1, N=10000, num\_threads=16**

**10000 0.021433 1.632045**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=1, N=20000, num\_threads=1**

**20000 0.021439 19.750499**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=1, N=20000, num\_threads=2**

**20000 0.021277 9.906535**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=1, N=20000, num\_threads=4**

**20000 0.021647 5.547020**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=1, N=20000, num\_threads=8**

**20000 0.021402 3.572340**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=1, N=20000, num\_threads=16**

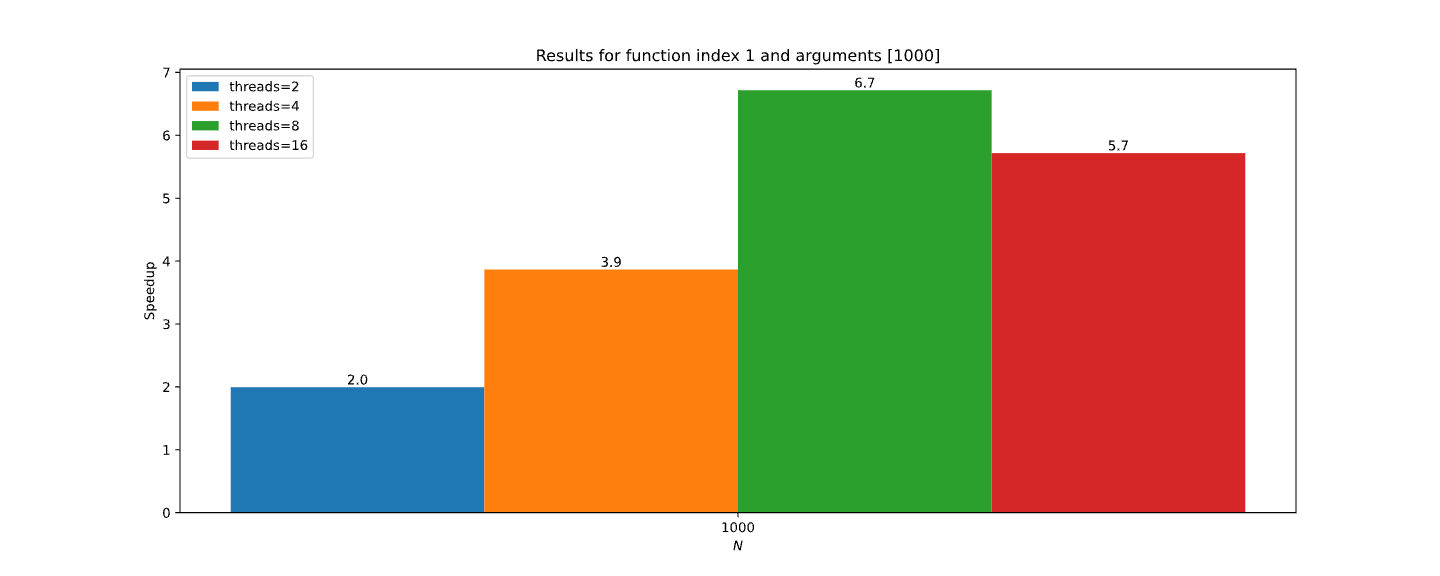
**20000 0.021273 3.199425**

**TEST END**

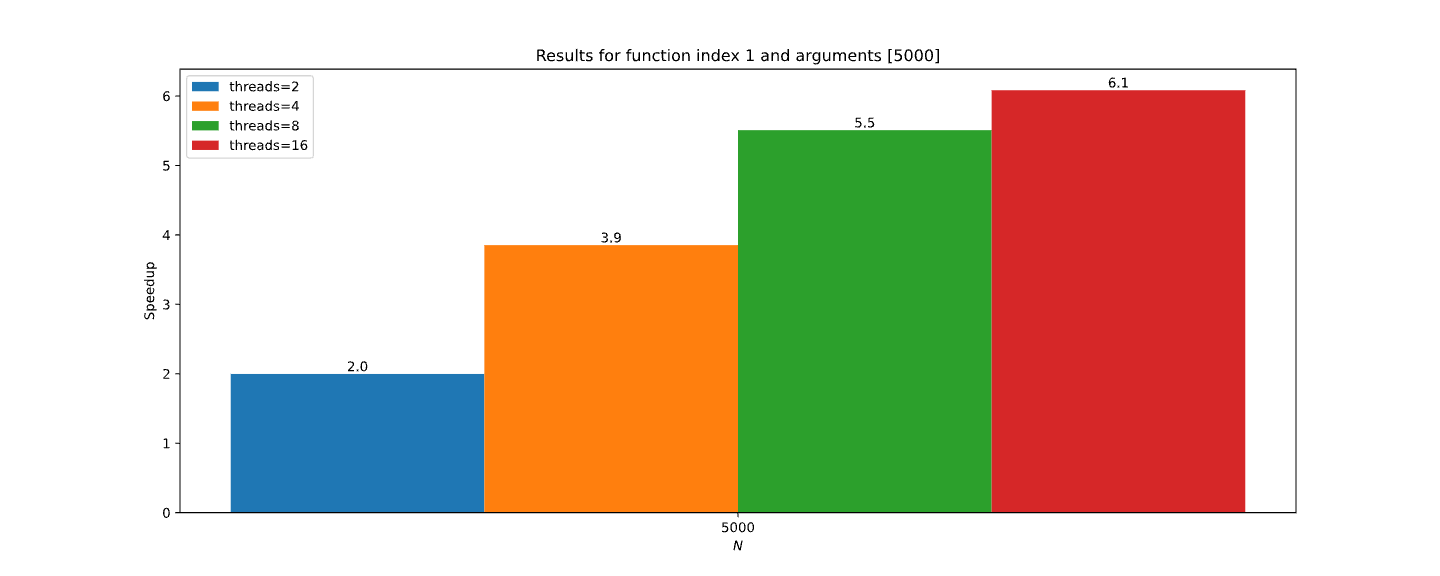
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 2D problema

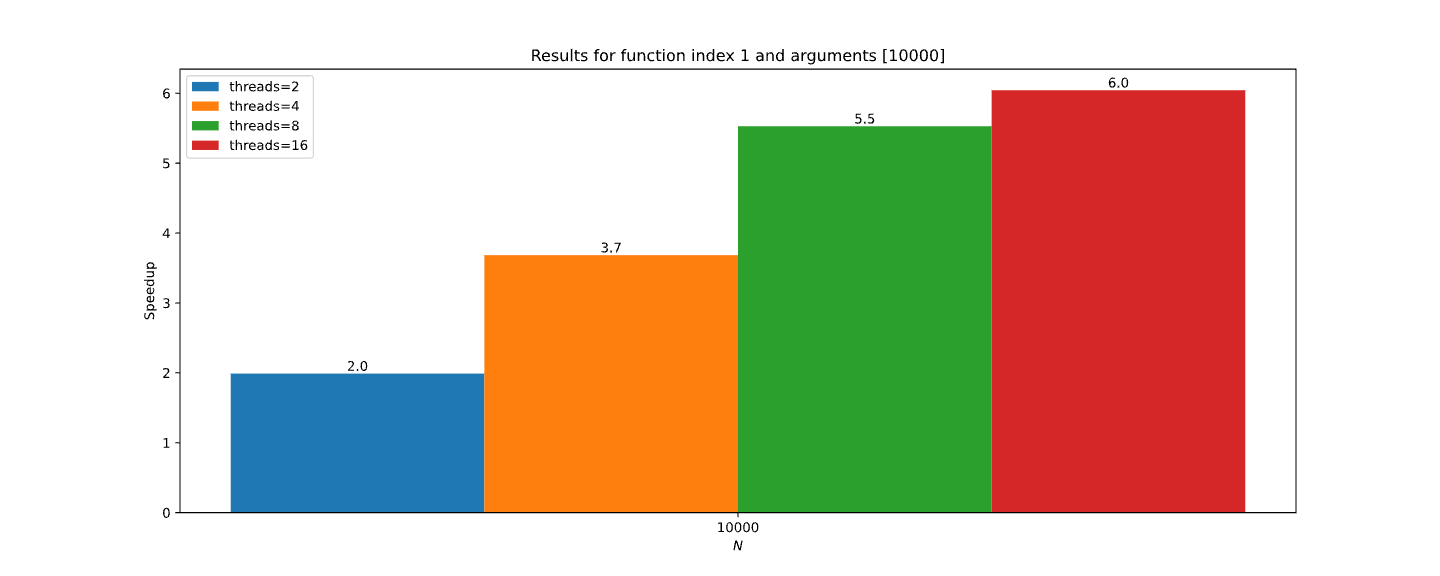
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.

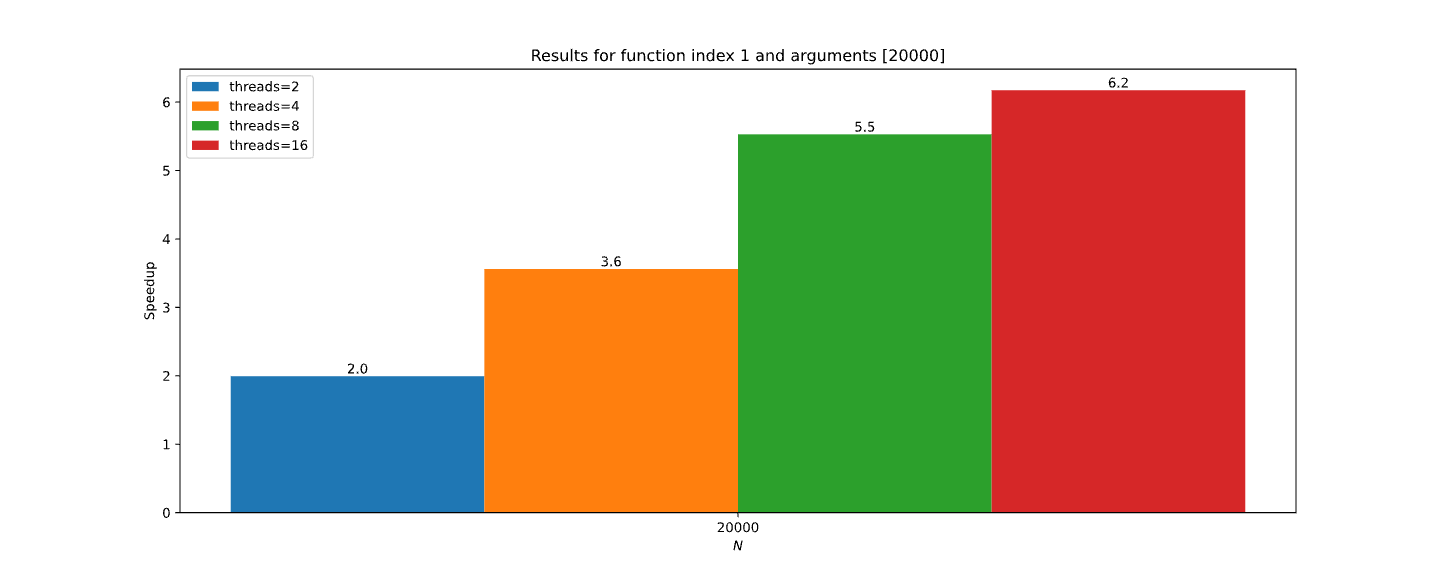


**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi 3D problema

**TEST: func=1, N=1000, num\_threads=1**

**1000 0.021717 3.062745**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=1, N=1000, num\_threads=2**

**1000 0.022281 1.535872**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=1, N=1000, num\_threads=4**

**1000 0.021581 0.803876**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=1, N=1000, num\_threads=8**

**1000 0.022497 0.555150**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=1, N=1000, num\_threads=16**

**1000 0.022259 0.548322**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=1, N=5000, num\_threads=1**

**5000 0.021273 15.359097**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=1, N=5000, num\_threads=2**

**5000 0.021476 7.700150**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=1, N=5000, num\_threads=4**

**5000 0.021044 4.336099**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=1, N=5000, num\_threads=8**

**5000 0.021017 2.710036**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=1, N=5000, num\_threads=16**

**5000 0.021090 2.677024**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=1, N=10000, num\_threads=1**

**10000 0.021100 30.697407**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=1, N=10000, num\_threads=2**

**10000 0.020977 15.478918**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=1, N=10000, num\_threads=4**

**10000 0.020969 8.728394**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=1, N=10000, num\_threads=8**

**10000 0.021013 5.442690**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=1, N=10000, num\_threads=16**

**10000 0.021182 5.214098**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=1, N=20000, num\_threads=1**

**20000 0.021027 61.423389**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=1, N=20000, num\_threads=2**

**20000 0.021059 31.049201**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=1, N=20000, num\_threads=4**

**20000 0.021030 17.497032**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=1, N=20000, num\_threads=8**

**20000 0.020920 10.889129**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=1, N=20000, num\_threads=16**

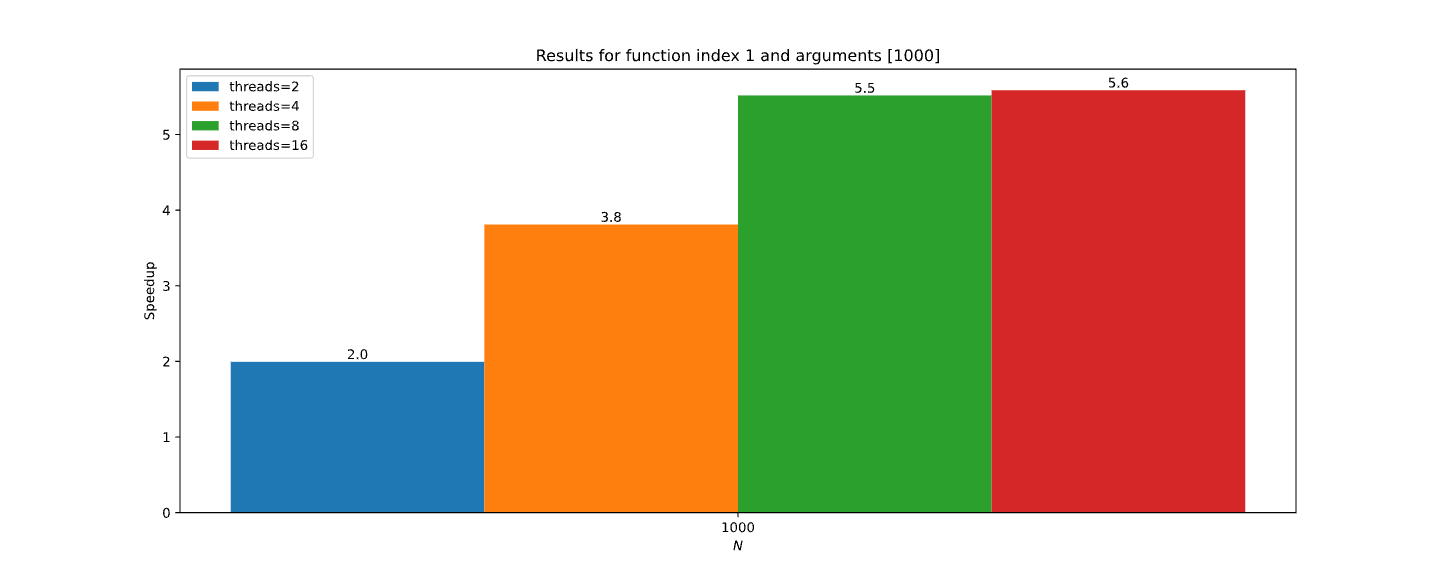
**20000 0.021028 10.379655**

**TEST END**

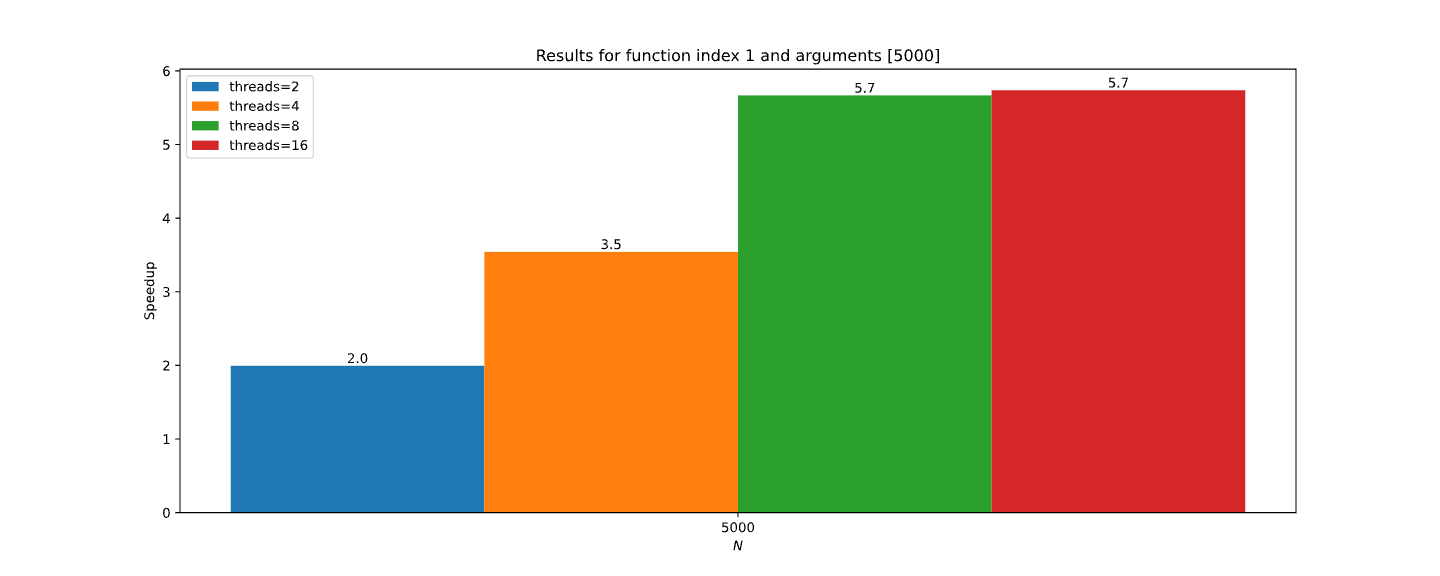
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 3D problema

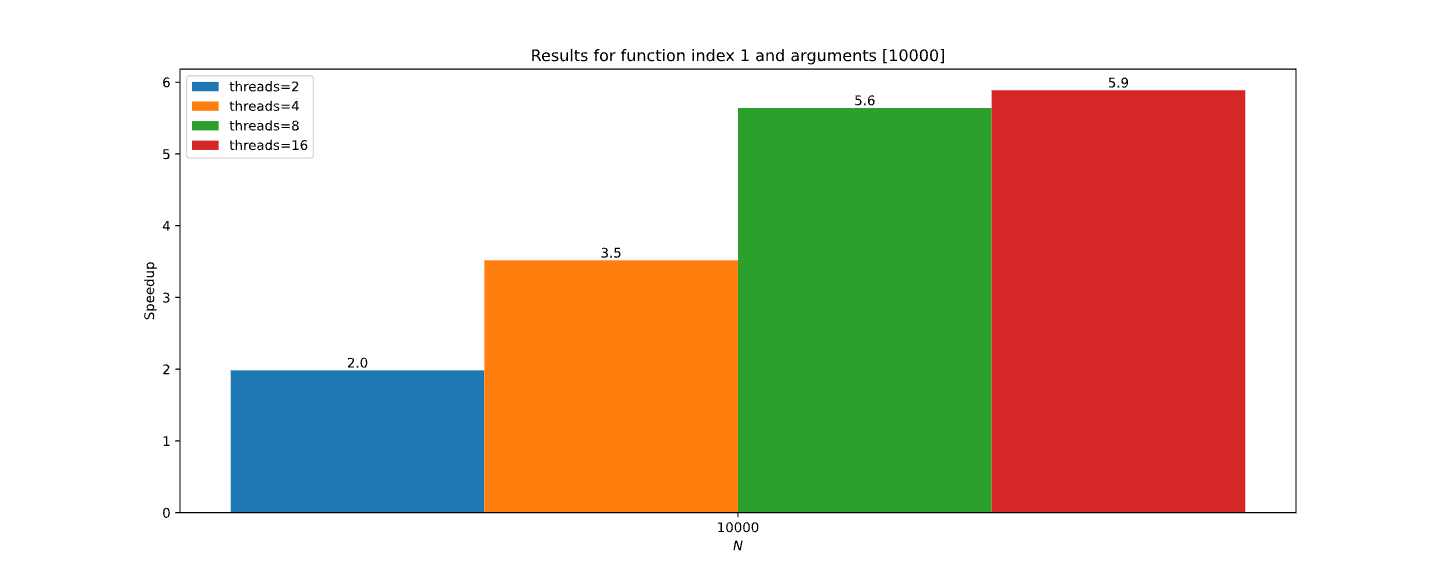
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.

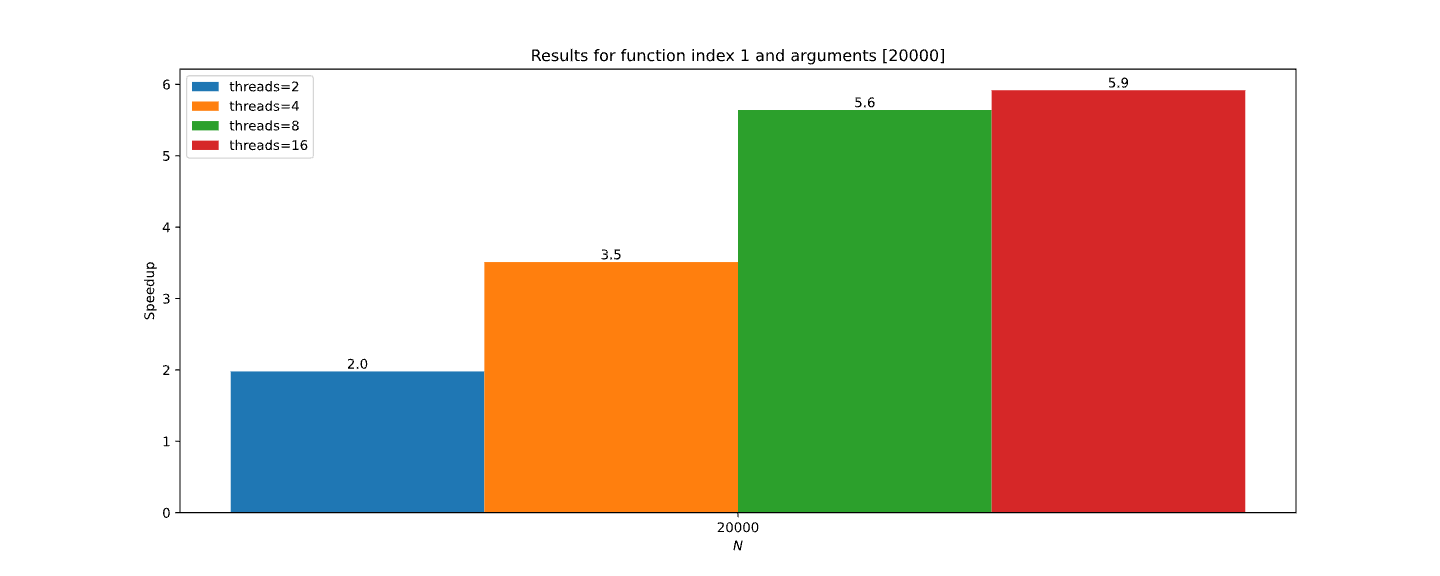


**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.** 

**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** (kao što je navedeno u tekstu zadatka). Za veći broj niti dobija se vidljivo veće ubrzanje, do broja niti 16, gde se vidi zasićenje. Zbog ovoga je prvi način paralelizacije pomoću collapse opcije bolji izbor.

# Problem 2 - Poasonova jednačina

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 4.

## Tekst problema

Rešiti prethodni problem korišćenjem koncepta poslova (*tasks*). Obratiti pažnju na eventualnu potrebu za sinhronizacijom i granularnost poslova.

## Delovi koje treba paralelizovati

### Diskusija

Implementacije su odvojene u zasebne funkcije, obeležene kao **feynman\_i(),** a koje implementiraju sam *Feynman-Kac* algoritam**.** Promenljiva **i** u nazivu funkcije uzima vrednosti [3, 4], za rešenja pomoću koncepta poslova, pa je u nastavku objašnjenje svakog od rešenja.

Diskusija je slična kao i za prethodni problem. Ipak, korišćenje poslova nam daje mogućnost da paralelizujemo sadržaj skroz unutrašnje **for** petlje, koji u sebi sadrži još jednu **while** petlju sa promenljivim brojem iteracija. Ovo ukazuje na to da poslovi mogu značajno pomoći u raspodeli posla ukoliko bi se paralelizovali na tom nivou. Paralelizacija na nivou pojedinačnih iteracija **while** petlje nije smislena, jer iteracije zavise jedna od druge.

Ipak, ovaj način paralelizacije donosi dodatne probleme sa sobom. Promenljiva **wt**, koja služi za izračunavanje sabirka sa ukupnom greškom, se sada računa unutar jednog posla, i računanje pomenutog sabirka bi moralo da se vrši nakon završetka poslova koje menjaju tu promenljivu. Prva ideja koja bi mogla da radi jeste **taskwait** direktiva nakon pomenute **for** petlje, ali to može da donese značajno usporenje.

### Način paralelizacije TASK + ATOMIC direktiva

Način na koji odabrano rešavanje ovog problema jeste mala preformulacija algoritma. Pošto je primećeno da su dimenzije tri spoljašnje petlje jako male (17, 12 i 7 iteracija respektivno), promenljiva koja čuva izračunate **w\_exact** i **wt** vrednosti u zavisnosti od iteracije petlje ne bi zauzela značajnu količinu memorije, ali se jasno vidi da problem ne skalira baš najbolje. Zbog ovoga, unutar samog posla se na **wt** promenljivu dodaje izračunata vrednost (zaštićena **atomic** direktivom) a nakon cele te procedure se vrši sabiranje izračunatih vrednosti kako bi se dobila krajnja greška. Na ovaj način više nije potrebna sinhronizacija po **err** niti **n\_inside**, jer im pristupa samo jedna nit. Takođe, održano je svojstvo da svaka nit koristi različit **seed**, definisanjem promenljive **seed** kao statičke, a zatim korišćenjem **threadprivate(seed)** opcije postižemo da svaka nit ima svoju privatnu kopiju ove promenljive, pa tako ne postoji **race-condition.**

## Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije ovog problema.

### Logovi izvršavanja 1D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=1, N=1000, num\_threads=1**

**1000 0.007980 1.356493**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=1, N=1000, num\_threads=2**

**1000 0.004651 0.669341**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=1, N=1000, num\_threads=4**

**1000 0.009101 0.347682**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=1, N=1000, num\_threads=8**

**1000 0.008146 0.184080**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=1, N=1000, num\_threads=16**

**1000 0.006432 0.159051**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=1, N=5000, num\_threads=1**

**5000 0.003724 6.686004**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=1, N=5000, num\_threads=2**

**5000 0.003106 3.361074**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=1, N=5000, num\_threads=4**

**5000 0.003821 1.707203**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=1, N=5000, num\_threads=8**

**5000 0.003914 1.103313**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=1, N=5000, num\_threads=16**

**5000 0.003544 0.973251**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=1, N=10000, num\_threads=1**

**10000 0.002754 13.444329**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=1, N=10000, num\_threads=2**

**10000 0.003149 6.729859**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=1, N=10000, num\_threads=4**

**10000 0.002879 3.570461**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=1, N=10000, num\_threads=8**

**10000 0.003148 2.247729**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=1, N=10000, num\_threads=16**

**10000 0.002969 1.951506**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=1, N=20000, num\_threads=1**

**20000 0.002440 26.961134**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=1, N=20000, num\_threads=2**

**20000 0.002483 13.511319**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=1, N=20000, num\_threads=4**

**20000 0.002685 7.369683**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=1, N=20000, num\_threads=8**

**20000 0.002452 4.501850**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=1, N=20000, num\_threads=16**

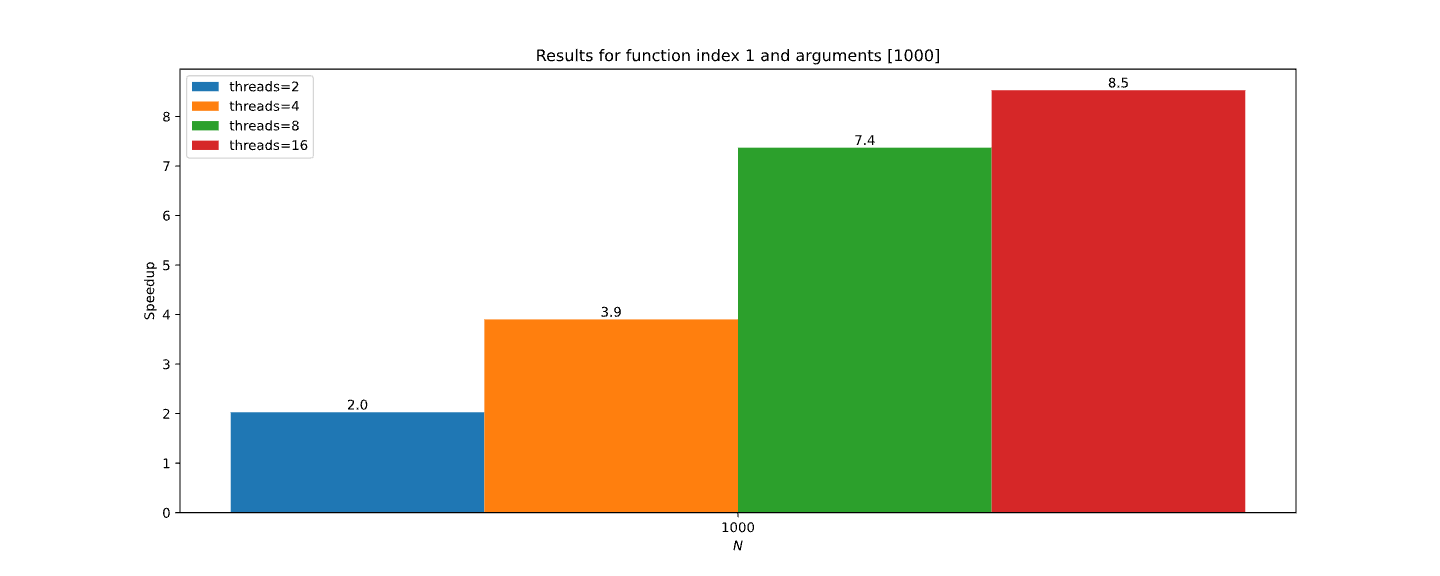
**20000 0.002382 3.926390**

**TEST END**

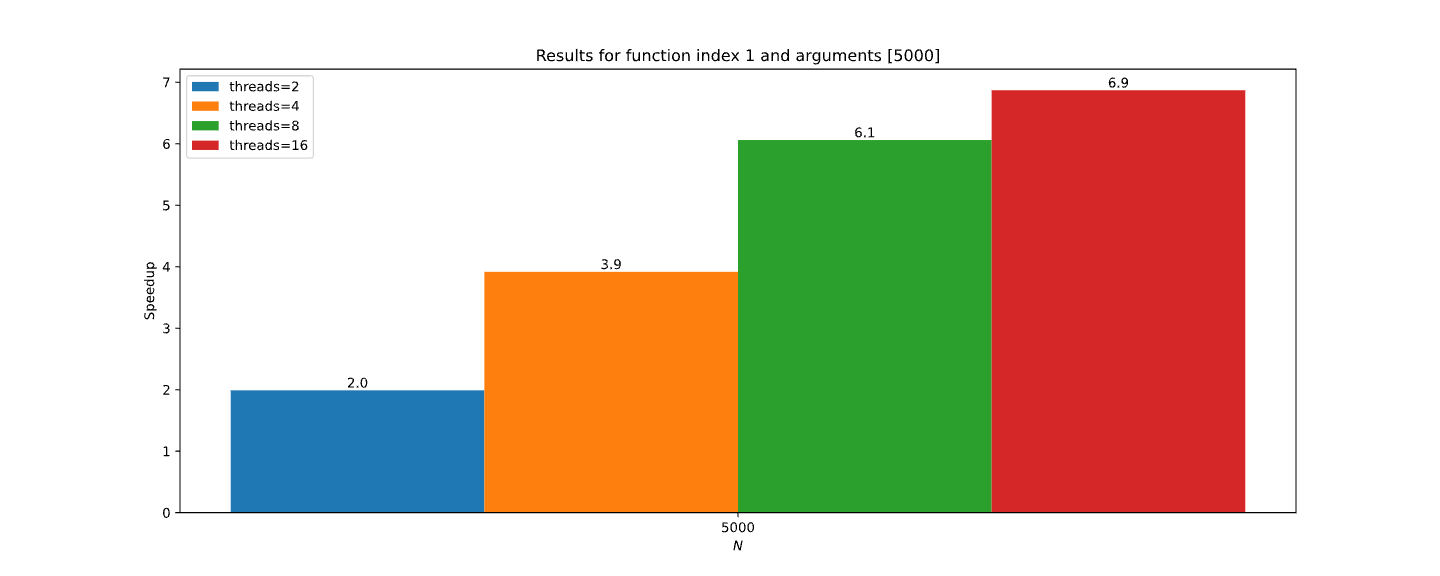
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 1D problema

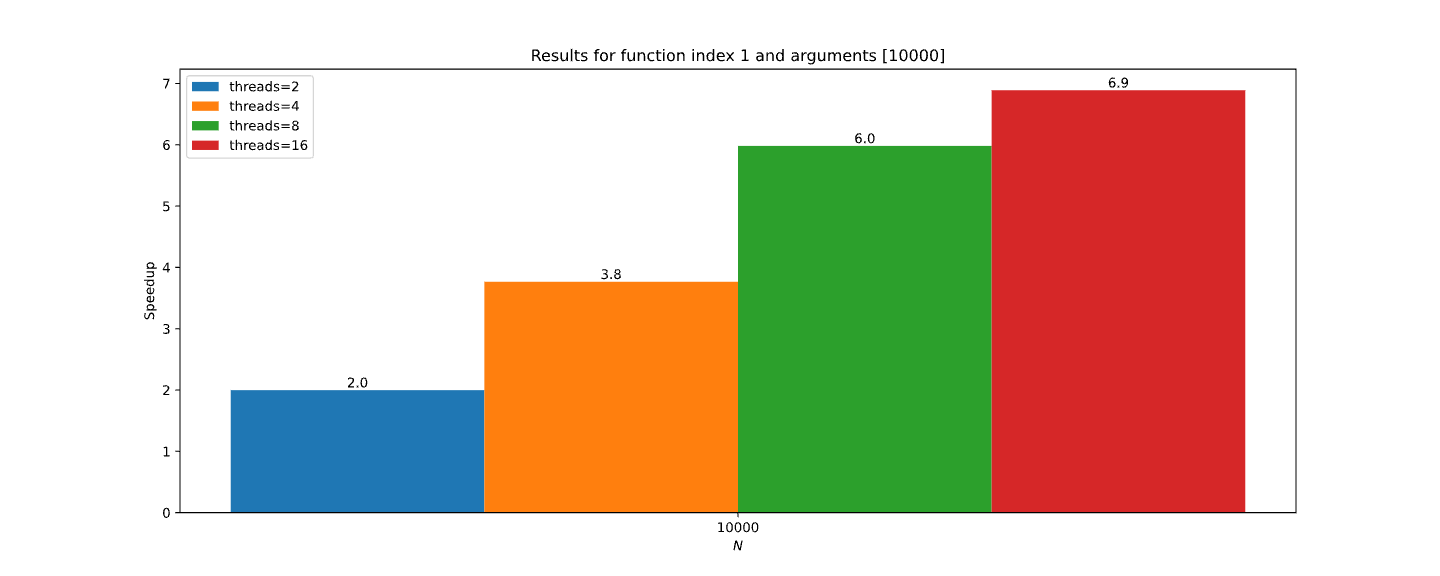
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



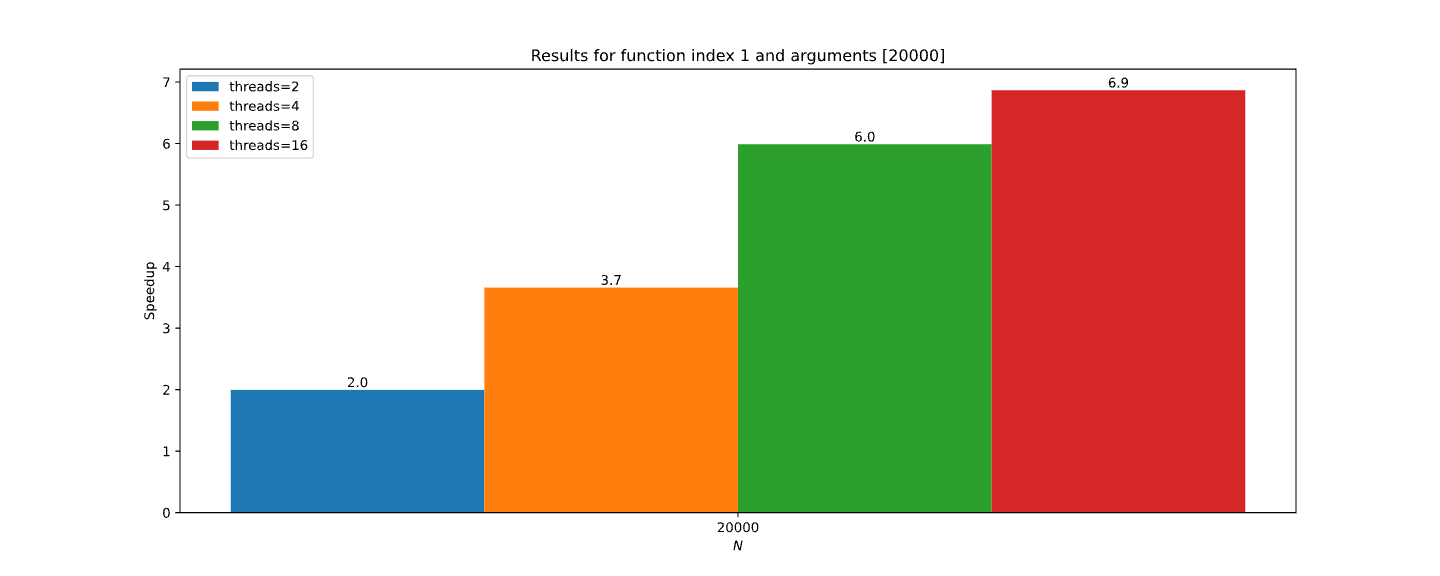
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi izvršavanja 2D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=2, N=1000, num\_threads=1**

**1000 0.022715 0.992876**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=2, N=1000, num\_threads=2**

**1000 0.023224 0.503911**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=2, N=1000, num\_threads=4**

**1000 0.022838 0.261095**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=2, N=1000, num\_threads=8**

**1000 0.022767 0.144217**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=2, N=1000, num\_threads=16**

**1000 0.024250 0.150924**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=2, N=5000, num\_threads=1**

**5000 0.021192 4.945979**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=2, N=5000, num\_threads=2**

**5000 0.021662 2.509540**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=2, N=5000, num\_threads=4**

**5000 0.021603 1.320600**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=2, N=5000, num\_threads=8**

**5000 0.021608 0.908570**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=2, N=5000, num\_threads=16**

**5000 0.021707 0.823187**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=2, N=10000, num\_threads=1**

**10000 0.021232 9.885788**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=2, N=10000, num\_threads=2**

**10000 0.021477 5.018174**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=2, N=10000, num\_threads=4**

**10000 0.021518 2.747521**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=2, N=10000, num\_threads=8**

**10000 0.021555 1.821954**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=2, N=10000, num\_threads=16**

**10000 0.021696 1.643764**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=2, N=20000, num\_threads=1**

**20000 0.021435 19.780673**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=2, N=20000, num\_threads=2**

**20000 0.021345 10.028531**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=2, N=20000, num\_threads=4**

**20000 0.021367 5.653628**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=2, N=20000, num\_threads=8**

**20000 0.021527 3.651123**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=2, N=20000, num\_threads=16**

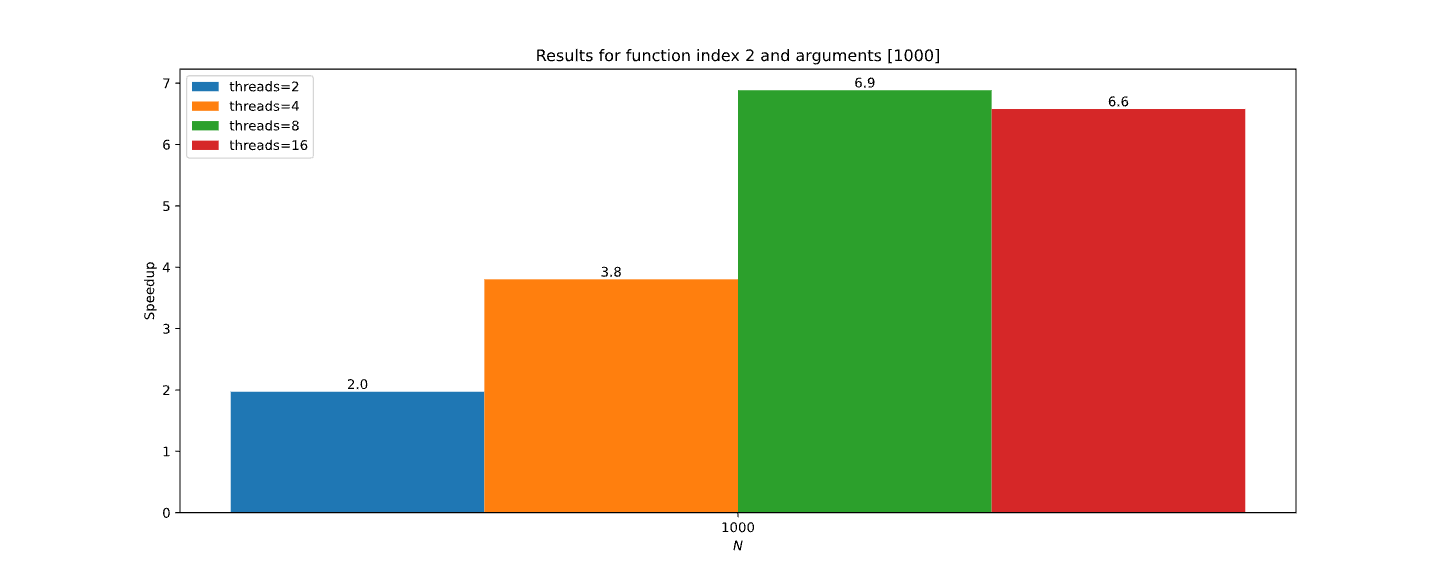
**20000 0.021741 3.227234**

**TEST END**

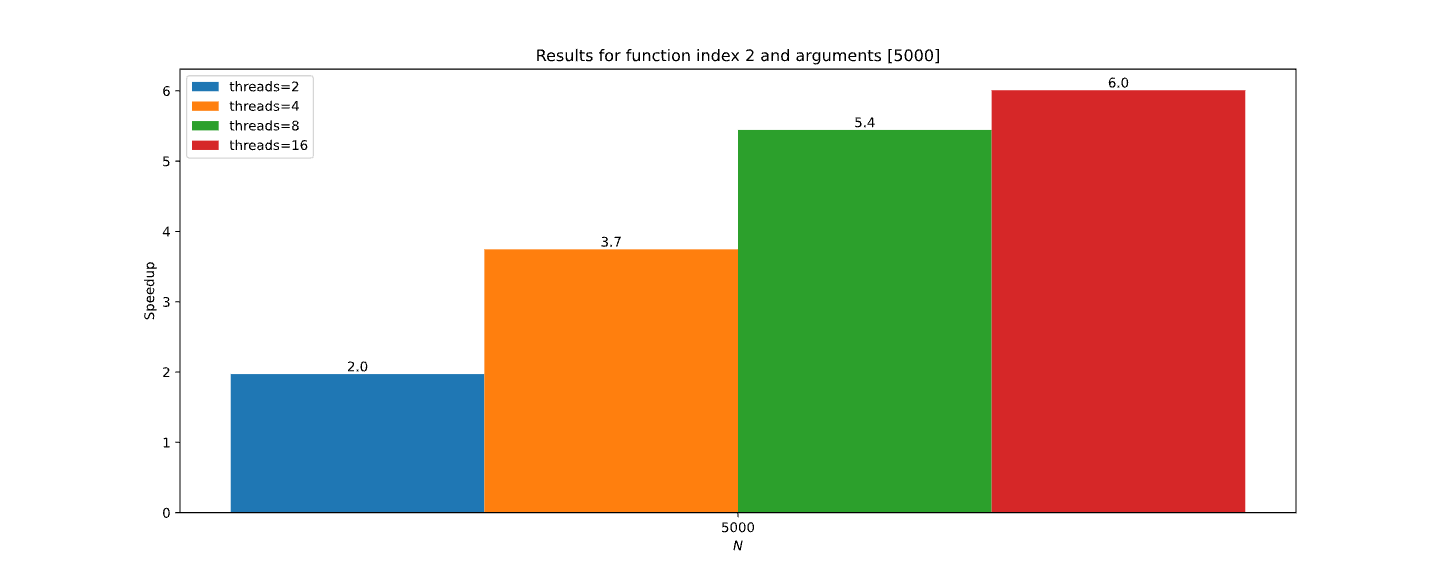
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 2D problema

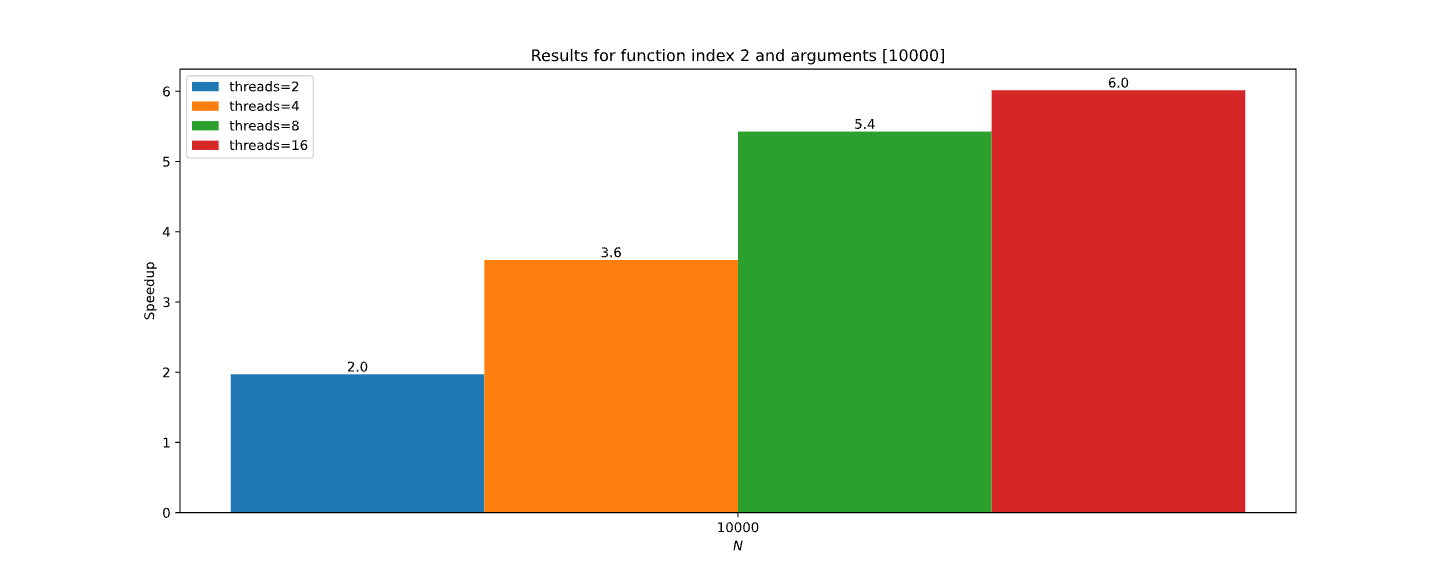
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



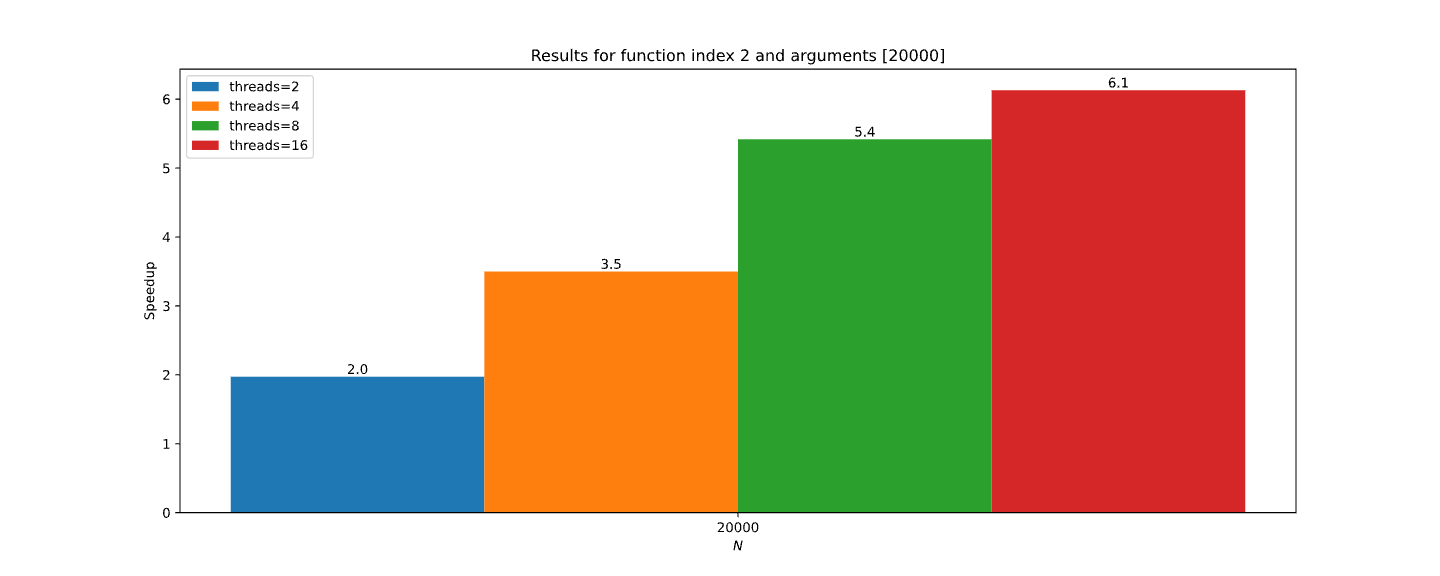
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi izvršavanja 3D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=2, N=1000, num\_threads=1**

**1000 0.021712 3.074757**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=2, N=1000, num\_threads=2**

**1000 0.022109 1.571493**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=2, N=1000, num\_threads=4**

**1000 0.021475 0.992118**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=2, N=1000, num\_threads=8**

**1000 0.021303 0.564069**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=2, N=1000, num\_threads=16**

**1000 0.021691 0.457407**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=2, N=5000, num\_threads=1**

**5000 0.021272 15.410696**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=2, N=5000, num\_threads=2**

**5000 0.021007 7.822821**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=2, N=5000, num\_threads=4**

**5000 0.020992 4.482696**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=2, N=5000, num\_threads=8**

**5000 0.021078 2.863595**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=2, N=5000, num\_threads=16**

**5000 0.021072 2.550105**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=2, N=10000, num\_threads=1**

**10000 0.021099 30.827771**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=2, N=10000, num\_threads=2**

**10000 0.020862 15.828347**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=2, N=10000, num\_threads=4**

**10000 0.021022 9.011122**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=2, N=10000, num\_threads=8**

**10000 0.021001 5.734614**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=2, N=10000, num\_threads=16**

**10000 0.021061 5.131505**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=2, N=20000, num\_threads=1**

**20000 0.021027 61.652812**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=2, N=20000, num\_threads=2**

**20000 0.020951 31.848441**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=2, N=20000, num\_threads=4**

**20000 0.020963 18.044311**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=2, N=20000, num\_threads=8**

**20000 0.020968 11.524479**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=2, N=20000, num\_threads=16**

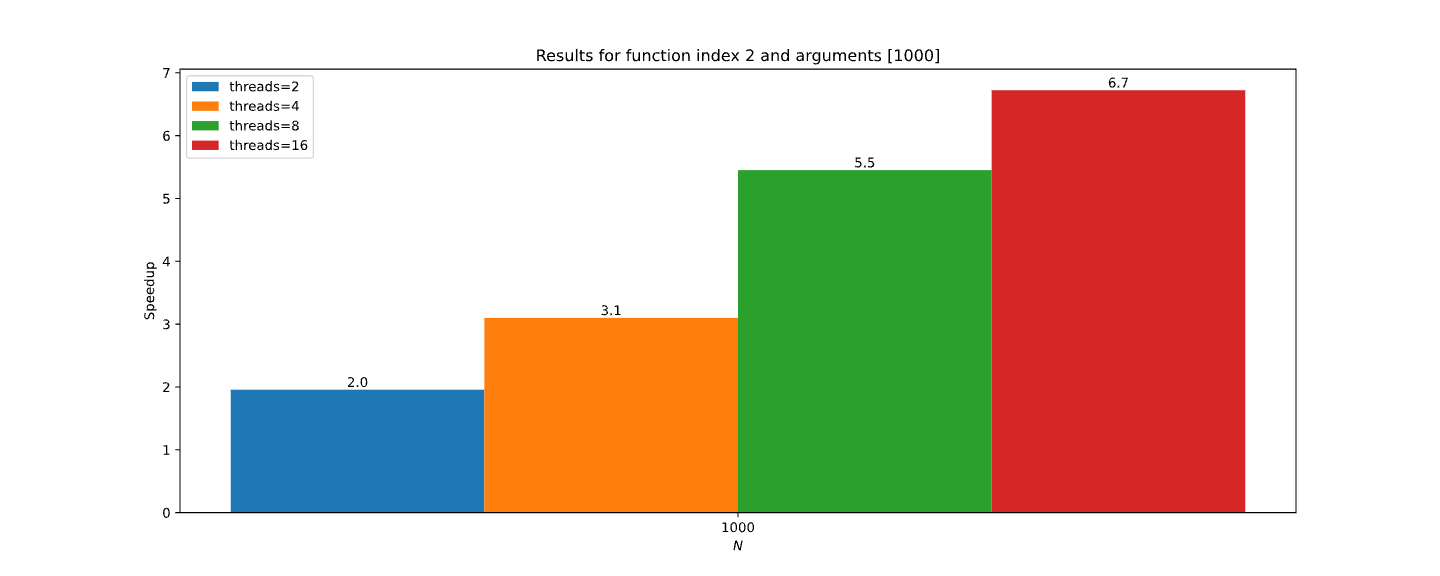
**20000 0.020975 10.076146**

**TEST END**

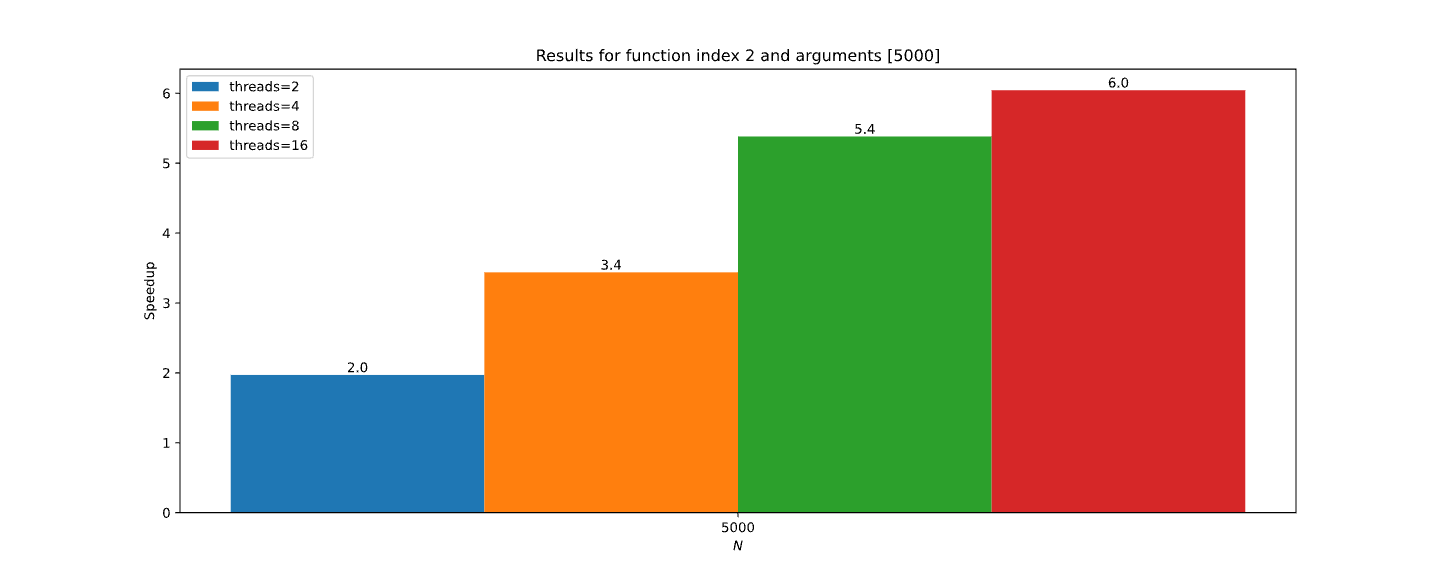
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 3D problema

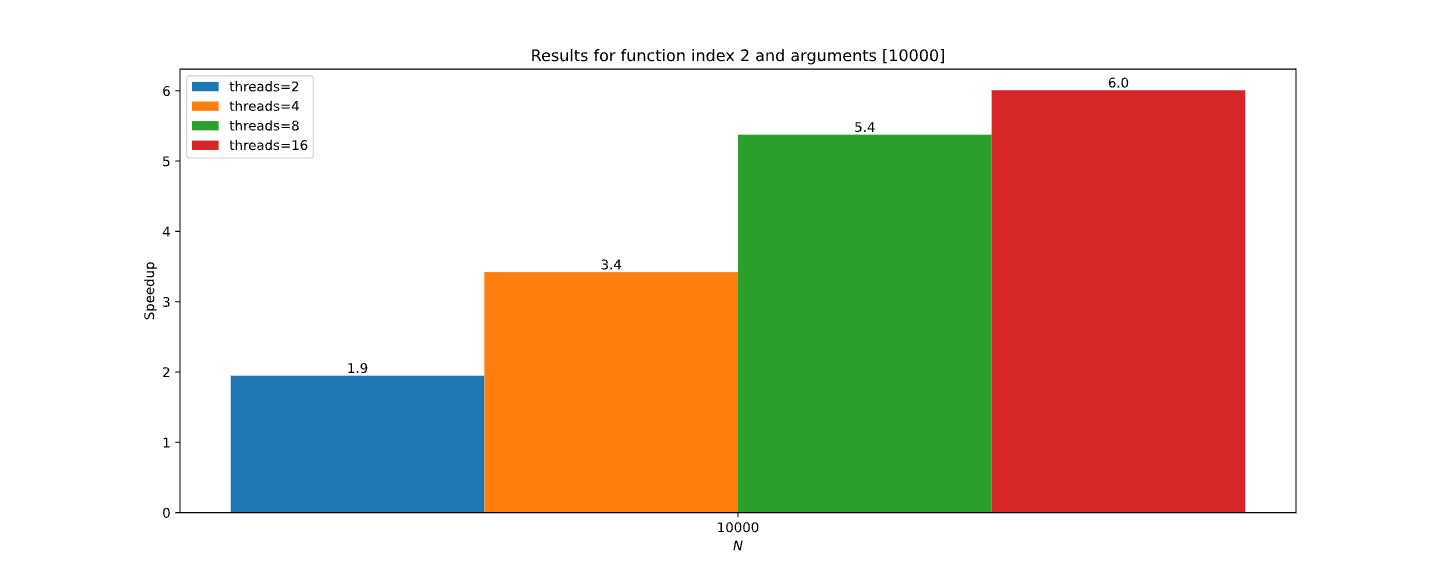
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



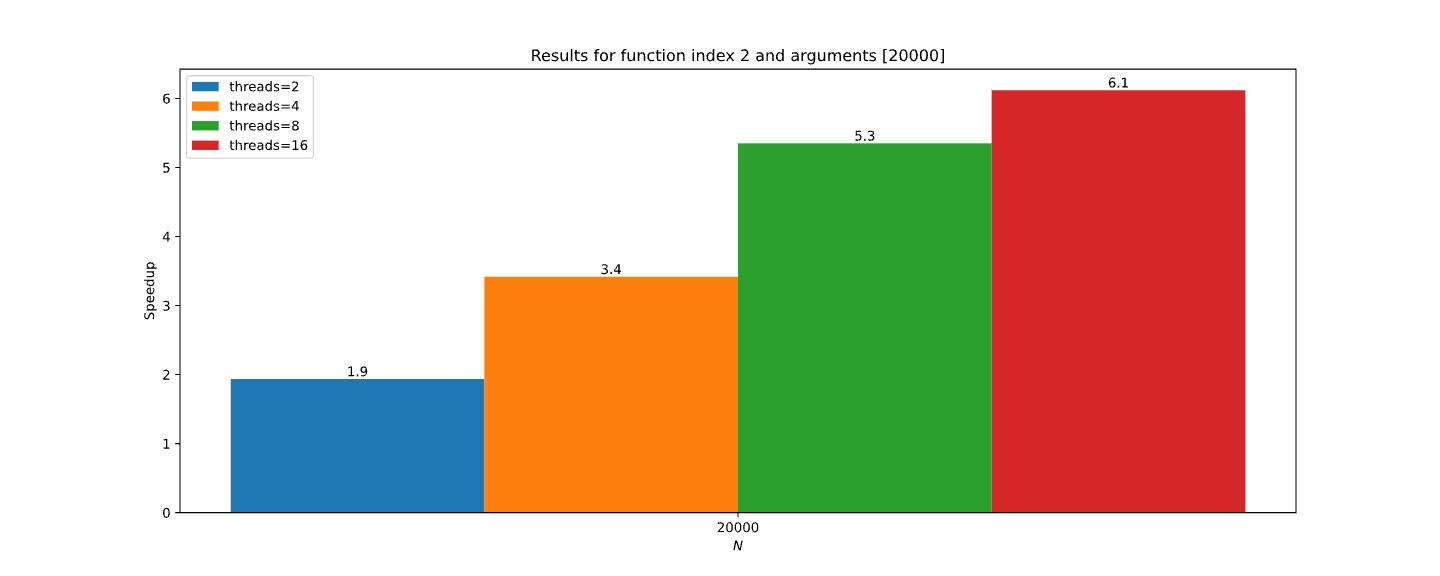
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Diskusija dobijenih rezultata

Grafici ubrzanja ovom metodom na nekim delovima pokazuju veće ubrzanje nego u prethodnom zadatku. Ovu paralelizaciju smatramo uspešnom, a tražena tačnost je takođe u traženom opsegu.

### Način paralelizacije TASK + locks distributed across multiple threads

Razlika u odnosu na prethodno rešenje se ogleda u tome što sada postoji fiksan broj **NUM\_LOCKS (trenutno 256 – mogu se ispobati i druge vrednosti)** brava na kojima se sinhronizuju niti. Kada nit dođe do kritične sekcije, uzeće indeks brave, koja joj pripada na osnovu pozicije u rešetki (brojača **i, j i k**). Kako se kretanje čestice koja polazi iz tačke koja je van elipsoida odmah završava, obična podela indeksa brave samo na osnovu brojača **i, j i k,** neće dati dobre rezultate, jer će jedan broj brava biti više korišćen, dok drugi broj neće biti korišćen. Ovo dovodi do neuravnoteženosti, pa samim tim do lošijih performansi. Zato se u rešenju, za dobijanje indeksa brave koristi neka vrsta heš funkcije, koja koristi velike proste brojeve kako bi bolje podelila indekse. Na ovoj funkciji se može dalje raditi, tako što će se pronaći funkcija koja svakoj od brava dodeli bribližno jednak broj tačaka unutar i van elipsoida. Njena trenutna implementacija i primer korišćenja je dana u nastavku:

**unsigned int get\_lock\_index(int i, int j, int k)**

**{**

**unsigned int hash = (unsigned int)(**

**i \* 73856093 ^**

**j \* 19349663 ^**

**k \* 83492791**

**);**

**return hash % NUM\_LOCKS;**

**}**

**. . .**

**int lock\_id = get\_lock\_index(i, j, k);**

**omp\_set\_lock(&locks[lock\_id]);**

**wt[i][j][k] += w;**

**omp\_unset\_lock(&locks[lock\_id]);**

**. . .**

Kao što je navedeno, rešenje se može unaprediti odabirom bolje heš funkcija koju koristi **get\_lock\_index().** U razmatranje je potrebno uzeti i prirodu problema, budući da interpoliranjem, za indekse **i, j i k,** brave se više koriste, što je tačka bliža koordinatnom početku, tj. kada su vrednosti ovih brojača bliže vrednostima **NI/2, NJ/2** i **NK/2** respektivno.

## Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije ovog problema.

### Logovi izvršavanja 1D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=2, N=1000, num\_threads=1**

**1000 0.007980 1.356881**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=2, N=1000, num\_threads=2**

**1000 0.004295 0.671027**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=2, N=1000, num\_threads=4**

**1000 0.008037 0.345547**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=2, N=1000, num\_threads=8**

**1000 0.006519 0.183218**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=2, N=1000, num\_threads=16**

**1000 0.006184 0.176642**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=2, N=5000, num\_threads=1**

**5000 0.003724 6.685248**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=2, N=5000, num\_threads=2**

**5000 0.003653 3.339278**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=2, N=5000, num\_threads=4**

**5000 0.003274 1.715172**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=2, N=5000, num\_threads=8**

**5000 0.003978 1.120695**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=2, N=5000, num\_threads=16**

**5000 0.003546 0.976868**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=2, N=10000, num\_threads=1**

**10000 0.002754 13.447884**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=2, N=10000, num\_threads=2**

**10000 0.002877 6.735230**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=2, N=10000, num\_threads=4**

**10000 0.002617 3.531056**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=2, N=10000, num\_threads=8**

**10000 0.002713 2.242829**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=2, N=10000, num\_threads=16**

**10000 0.003436 1.949794**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=2, N=20000, num\_threads=1**

**20000 0.002440 26.962216**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=2, N=20000, num\_threads=2**

**20000 0.002327 13.462419**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=2, N=20000, num\_threads=4**

**20000 0.002549 7.367382**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=2, N=20000, num\_threads=8**

**20000 0.002512 4.500842**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=2, N=20000, num\_threads=16**

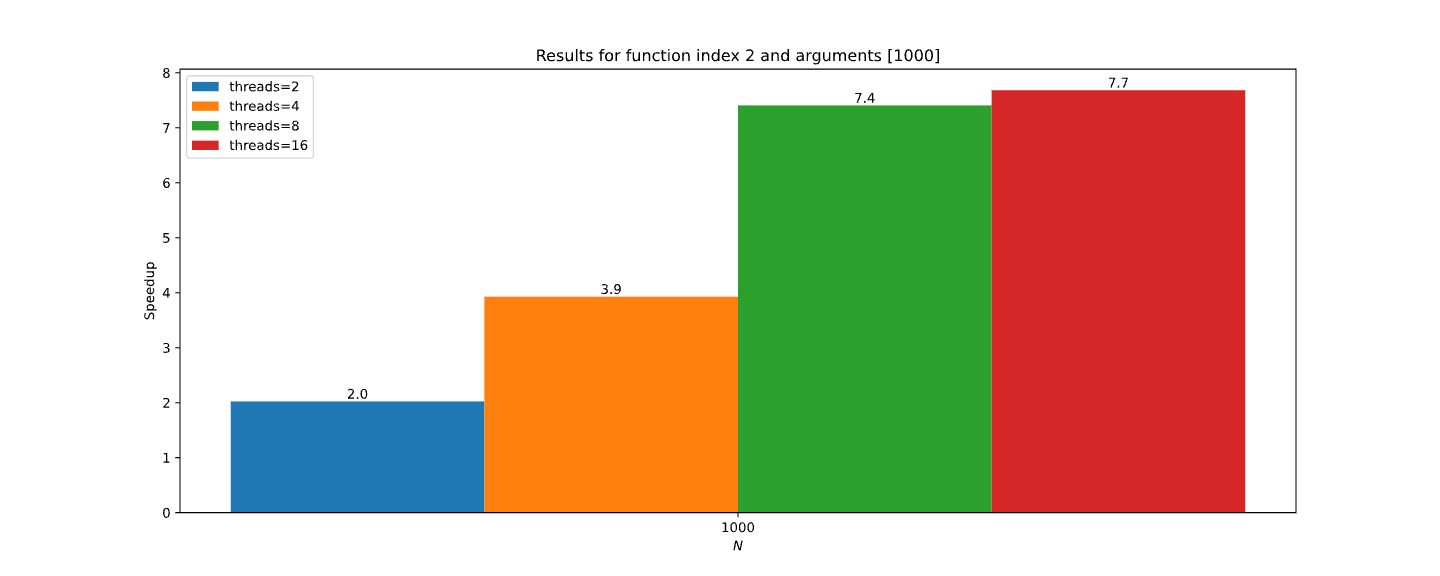
**20000 0.002628 3.939088**

**TEST END**

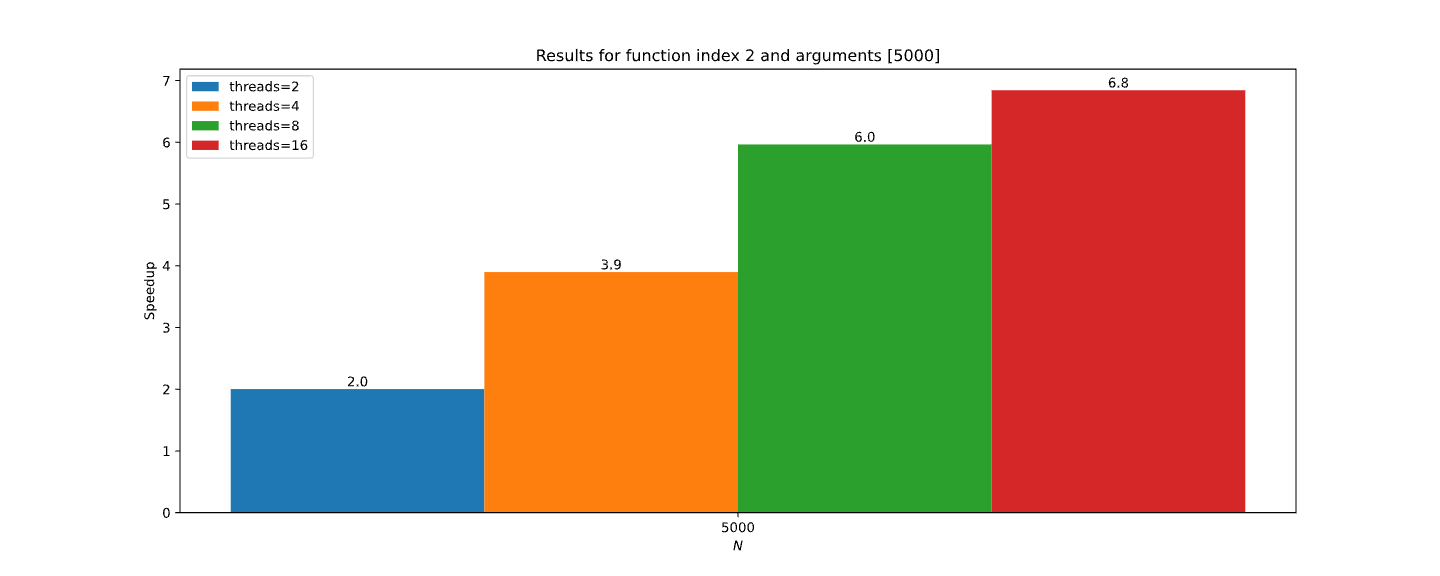
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 1D problema

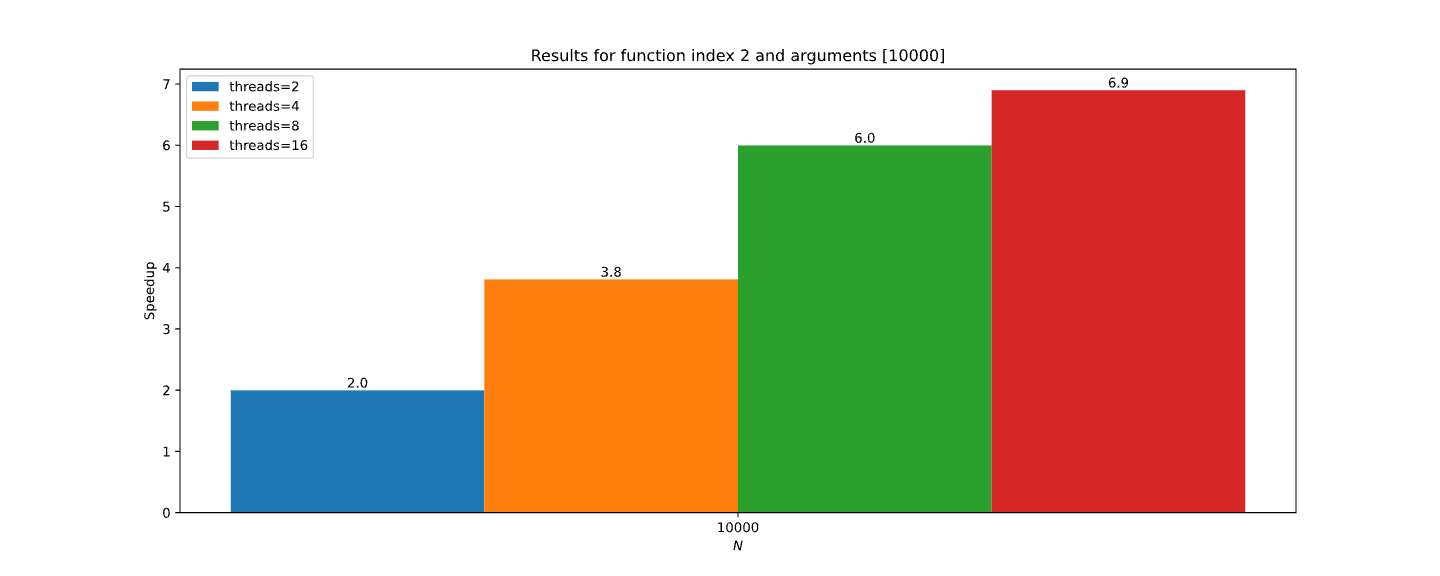
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



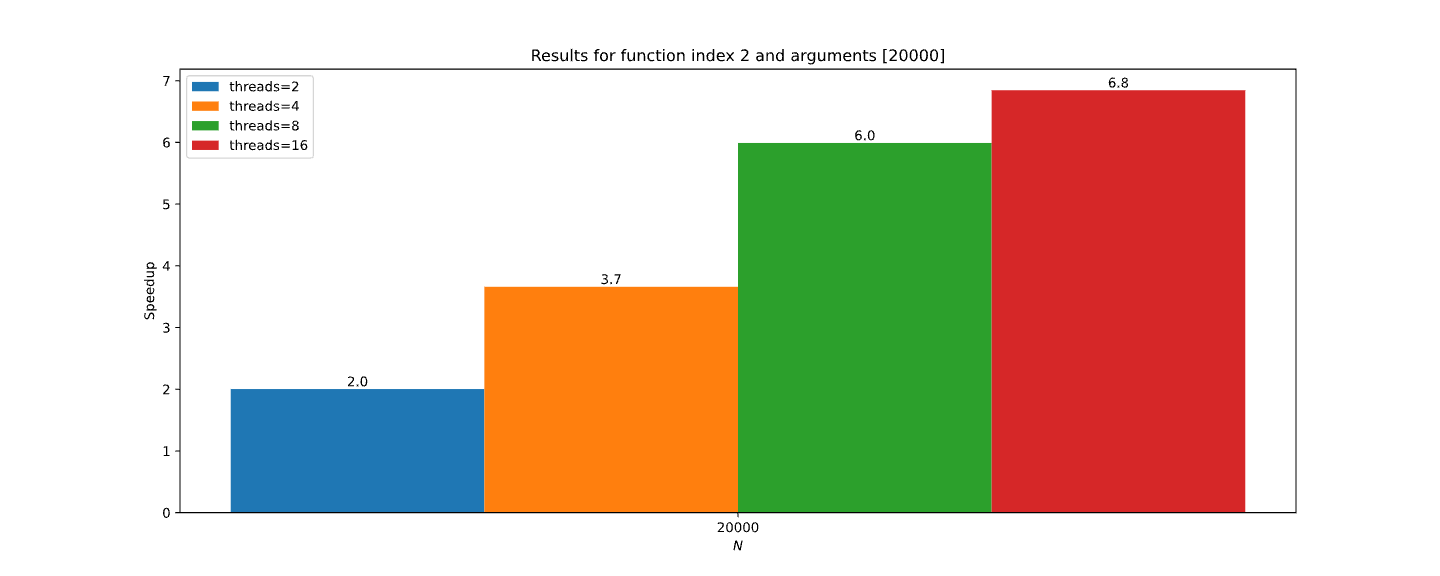
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi izvršavanja 2D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=3, N=1000, num\_threads=1**

**1000 0.022715 0.994604**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=3, N=1000, num\_threads=2**

**1000 0.023353 0.505000**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=3, N=1000, num\_threads=4**

**1000 0.022595 0.261688**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=3, N=1000, num\_threads=8**

**1000 0.023413 0.148423**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=3, N=1000, num\_threads=16**

**1000 0.023386 0.155257**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=3, N=5000, num\_threads=1**

**5000 0.021192 4.956835**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=3, N=5000, num\_threads=2**

**5000 0.021755 2.516966**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=3, N=5000, num\_threads=4**

**5000 0.021694 1.322462**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=3, N=5000, num\_threads=8**

**5000 0.021605 0.911495**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=3, N=5000, num\_threads=16**

**5000 0.021717 0.848224**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=3, N=10000, num\_threads=1**

**10000 0.021232 9.902769**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=3, N=10000, num\_threads=2**

**10000 0.021414 5.033443**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=3, N=10000, num\_threads=4**

**10000 0.021447 2.763848**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=3, N=10000, num\_threads=8**

**10000 0.021603 1.831766**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=3, N=10000, num\_threads=16**

**10000 0.021467 1.612478**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=3, N=20000, num\_threads=1**

**20000 0.021435 19.817893**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=3, N=20000, num\_threads=2**

**20000 0.021386 10.051411**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=3, N=20000, num\_threads=4**

**20000 0.021382 5.670350**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=3, N=20000, num\_threads=8**

**20000 0.021333 3.659758**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=3, N=20000, num\_threads=16**

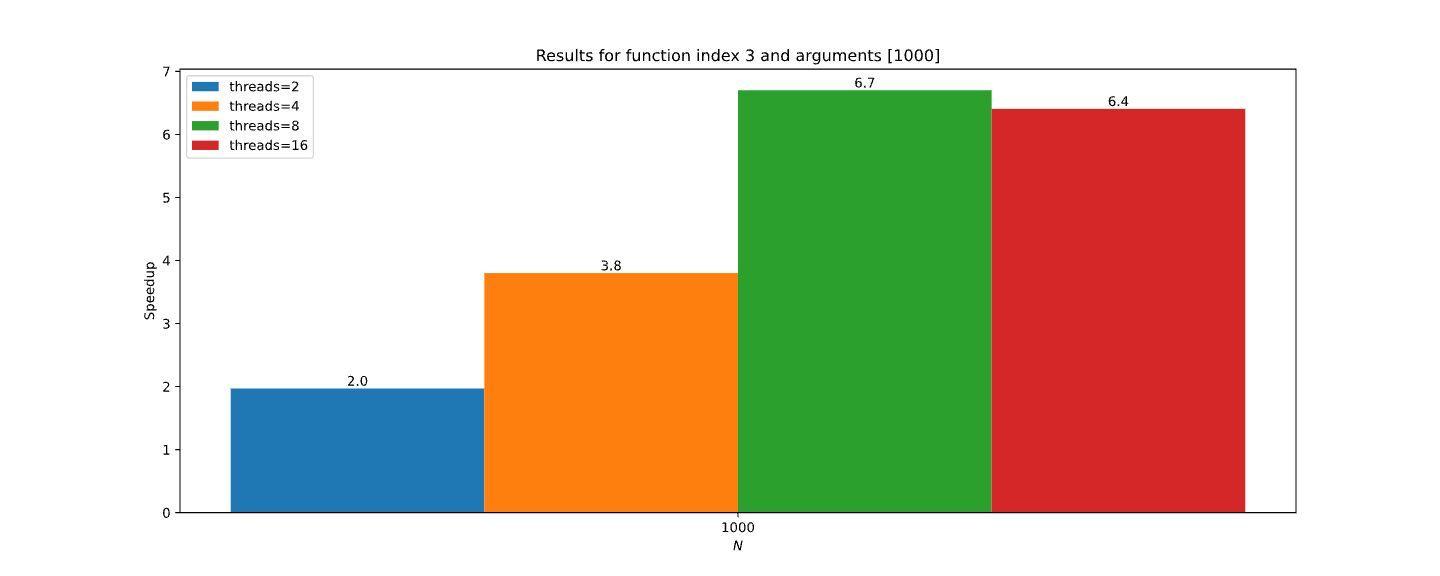
**20000 0.021538 3.204024**

**TEST END**

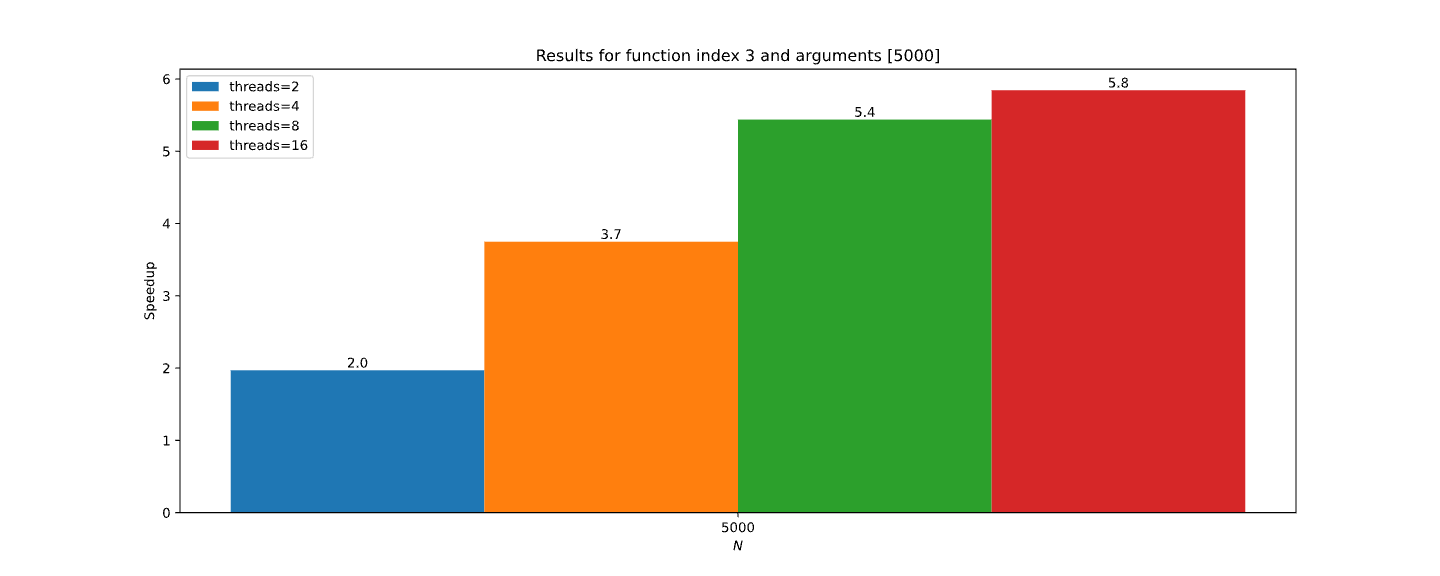
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 2D problema

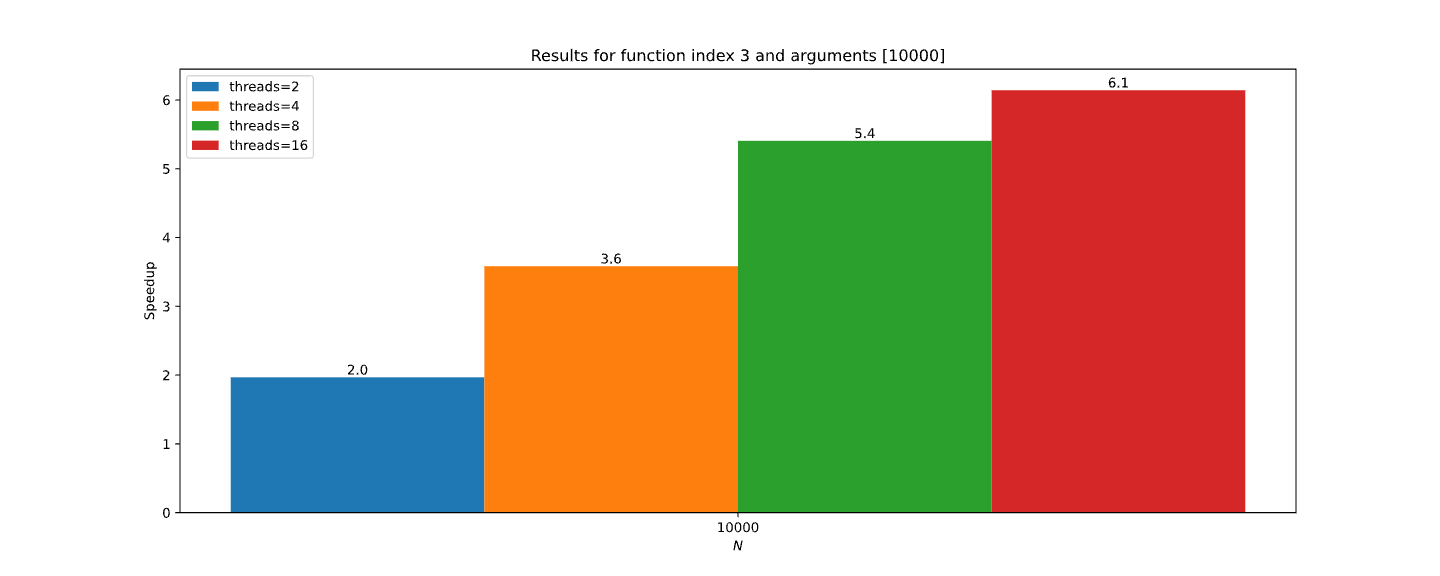
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



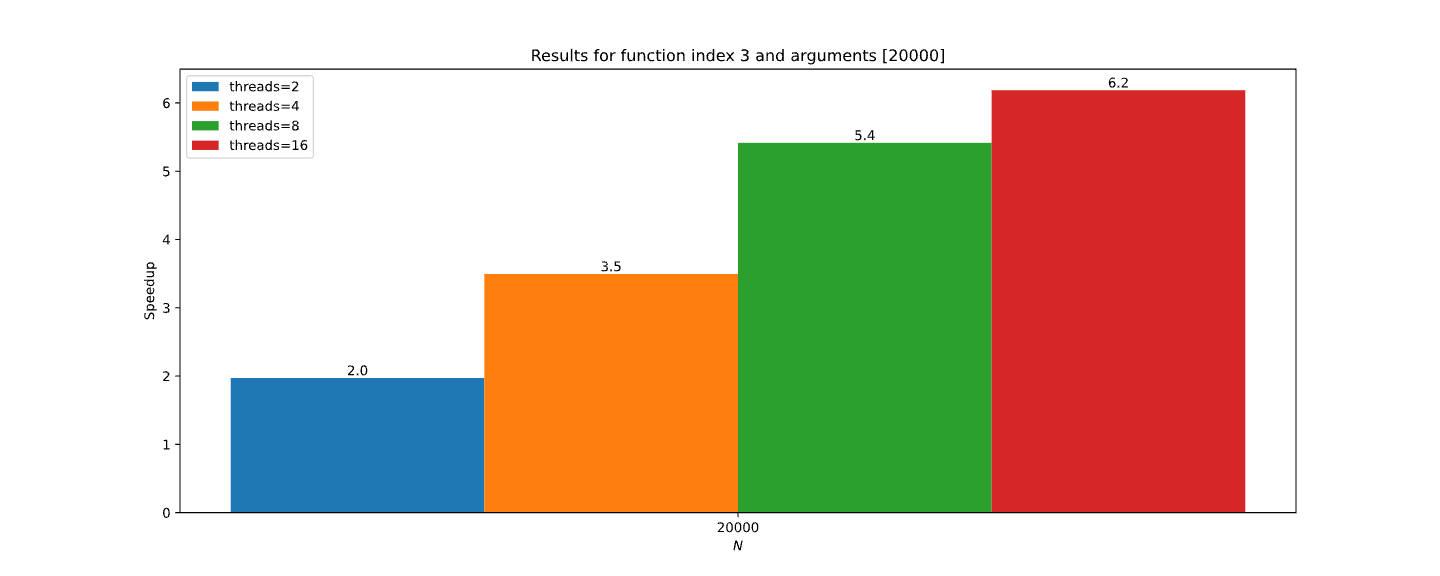
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Logovi izvršavanja 3D problema

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Merenje vremena je rađeno korišćenjem *wall clock time*, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=3, N=1000, num\_threads=1**

**1000 0.021712 3.097028**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

**TEST: func=3, N=1000, num\_threads=2**

**1000 0.021648 1.573662**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

**TEST: func=3, N=1000, num\_threads=4**

**1000 0.021311 0.844098**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

**TEST: func=3, N=1000, num\_threads=8**

**1000 0.022280 0.569881**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

**TEST: func=3, N=1000, num\_threads=16**

**1000 0.021871 0.451361**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

**TEST: func=3, N=5000, num\_threads=1**

**5000 0.021272 15.532099**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

**TEST: func=3, N=5000, num\_threads=2**

**5000 0.020995 7.893211**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

**TEST: func=3, N=5000, num\_threads=4**

**5000 0.021099 4.536205**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

**TEST: func=3, N=5000, num\_threads=8**

**5000 0.021165 2.874447**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

**TEST: func=3, N=5000, num\_threads=16**

**5000 0.020993 2.470741**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

**TEST: func=3, N=10000, num\_threads=1**

**10000 0.021099 31.074860**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

**TEST: func=3, N=10000, num\_threads=2**

**10000 0.020841 15.898273**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

**TEST: func=3, N=10000, num\_threads=4**

**10000 0.021096 9.088829**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

**TEST: func=3, N=10000, num\_threads=8**

**10000 0.021068 5.755733**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

**TEST: func=3, N=10000, num\_threads=16**

**10000 0.020911 5.109369**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

**TEST: func=3, N=20000, num\_threads=1**

**20000 0.021027 62.221741**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

**TEST: func=3, N=20000, num\_threads=2**

**20000 0.020933 32.099376**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

**TEST: func=3, N=20000, num\_threads=4**

**20000 0.020870 18.135718**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

**TEST: func=3, N=20000, num\_threads=8**

**20000 0.020829 11.606387**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

**TEST: func=3, N=20000, num\_threads=16**

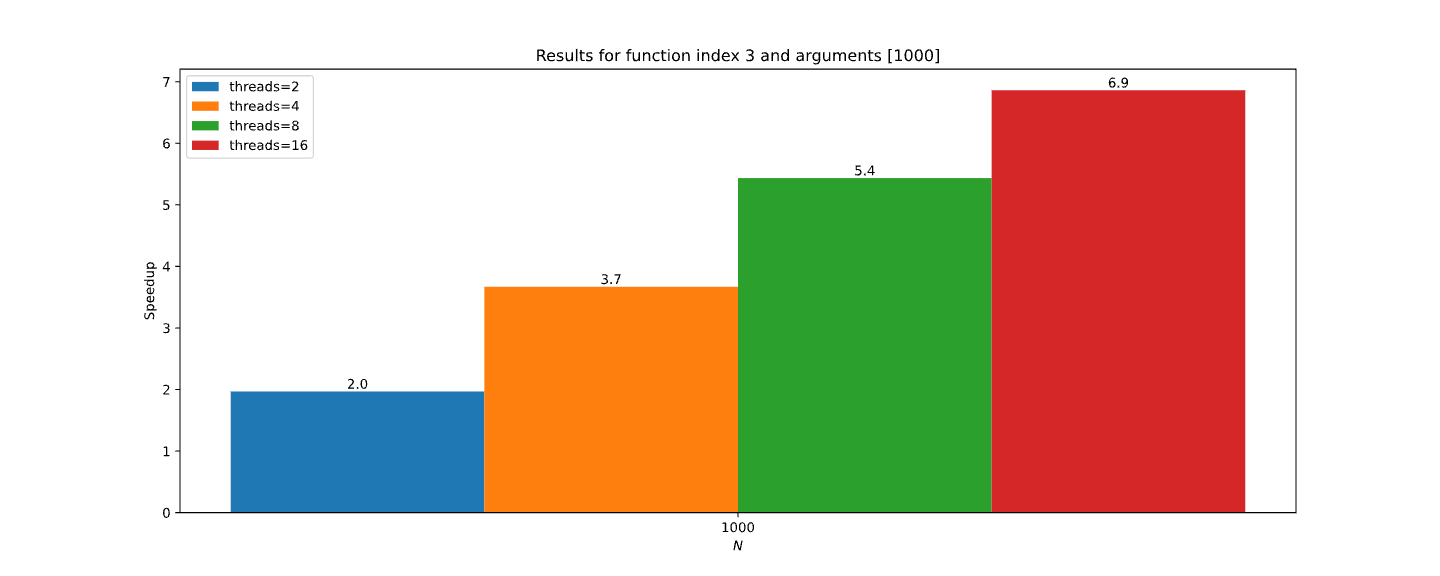
**20000 0.020981 10.016916**

**TEST END**

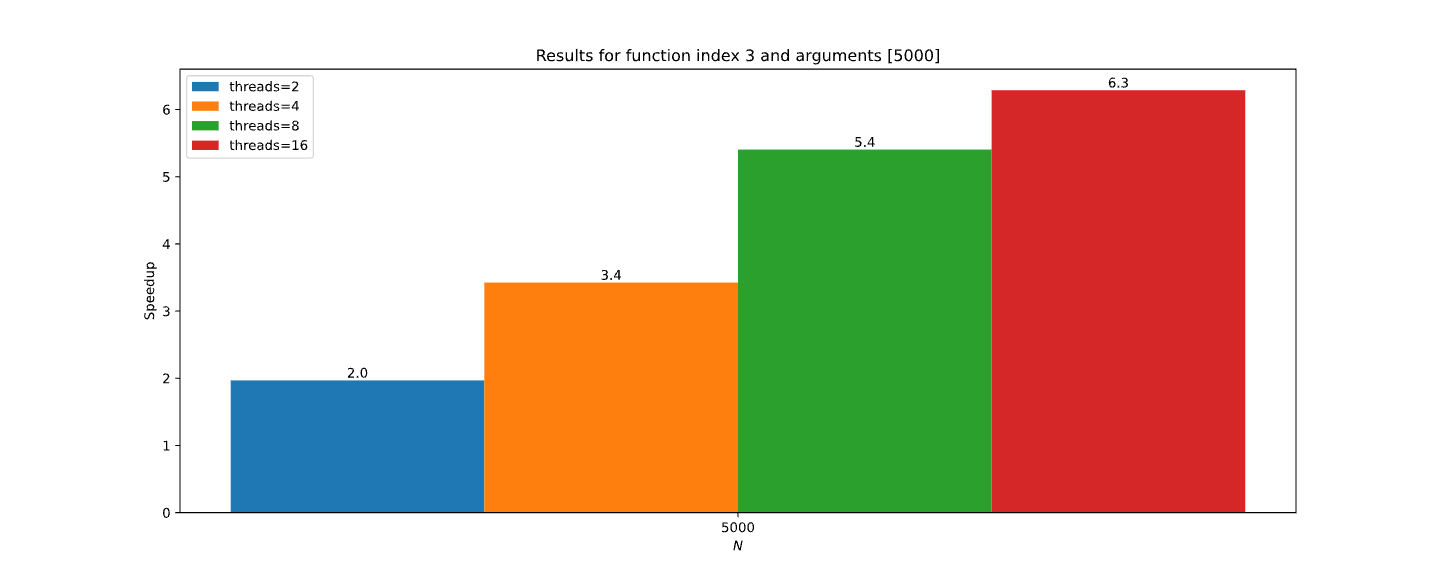
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja 3D problema

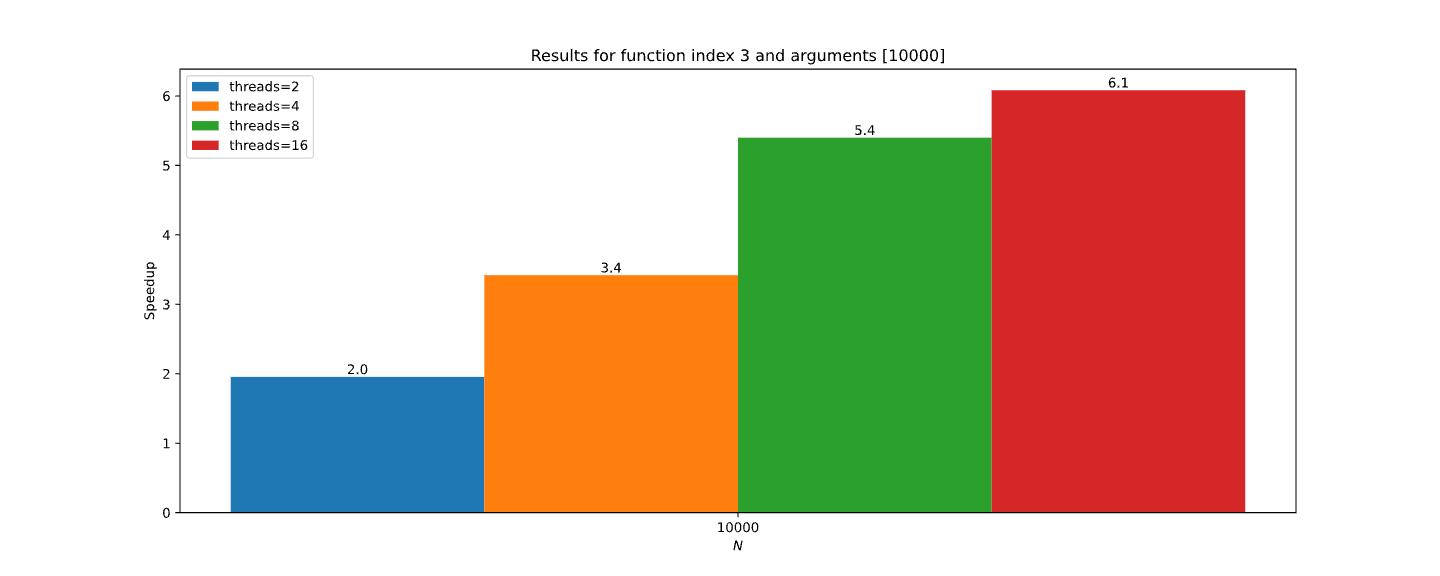
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



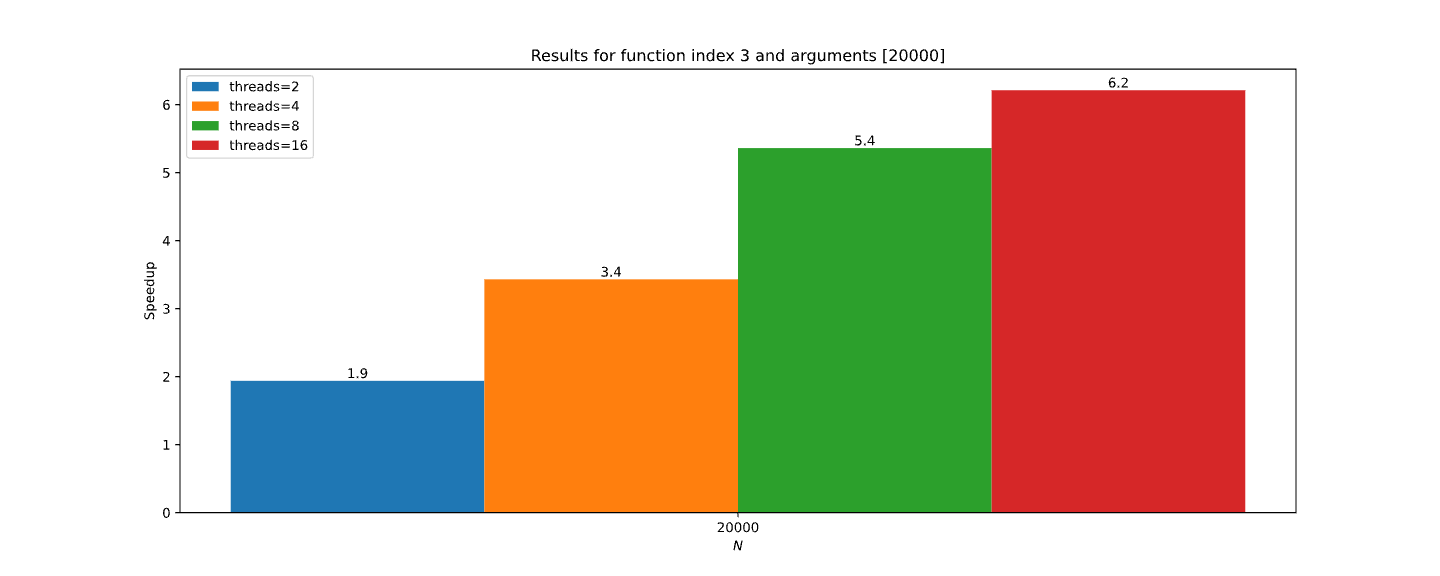
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 5000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 10000.**



**Grafik ubrzanja za različit broj niti, sa argumentom 20000.**

### Diskusija dobijenih rezultata

Grafici ubrzanja ovom metodom na nekim delovima pokazuju nešto veće ubrzanje, u odnosu na prethodnu metodu. Ovu paralelizaciju smatramo uspešnom, a tražena tačnost je takođe u traženom opsegu.