UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

DOMAĆI ZADATAK – PTHREADS

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:

prof. dr Marko Mišić

Student:

Vuk Lužanin 29/2022

Beograd, april 2025.

SADRŽAJ

1. NAPOMENE

Projekat se nalazi na Githubu, na linku: **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/Pthreads/.`** Svaki fajl sadrži rešenje koje je u ovom dokumentu razmatrano.

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

U datom rešenju je korišćena **`omp_get_wtime()`** funkcija za merenje proteklog vremena, ovo se može promeniti.

2. PROBLEM 1 - POASONOVA JEDNAČINA

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

2.1. Tekst problema

Paralelizovati program koji vrši izračunavanje 1D, 2D i 3D Poasonove jednačine korišćenjem Feynman-Kac 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/Pthreads/`** u arhivi koja je priložena uz ovaj dokument.

Rešiti prethodni problem korišćenjem Pthreads biblioteke, pomoću POSIX niti. Obratiti pažnju na eventualnu potrebu za sinhronizacijom.

2.2. Delovi koje treba paralelizovati

2.2.1. Diskusija

Implementacija se nalazi u funkcije, obeležene kao **`feynman_pthreads_DIMENSION()`**, a koje implementiraju sam *Feynman-Kac* algoritam**.**

Eksplicitno korišćenje niti nam daje mogućnost da paralelizujemo sadržaj unutrašnje **`for`** petlje, koji u sebi sadrži još jednu **`while`** petlju sa promenljivim brojem iteracija, a po uzoru na rešenje sa **`OpenMP taskovima`**. Ovo ukazuje na to da niti mogu značajno pomoći u raspodeli posla ukoliko bi se paralelizovalo 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.

2.2.2. Način paralelizacije

Paralelizovana je unutrašnja for petlja, gde će jedna nit biti zadužena za **`N/num_threads +/- remainder`** iteracija. 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.

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 i 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 približno 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;
}
. . .
// koriscenje
int lock_id = get_lock_index(arg->i, arg->j, arg->k);
pthread_mutex_lock(&wt_mutexes[lock_id]);
wt[arg->i][arg->j][arg->k] += w;
pthread_mutex_unlock(&wt_mutexes[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.

Potpis funkcije čije telo će niti izvršavati je:

```
void* trial_worker(void *varg);
```

Nit se kreira na sledeći način:

```
typedef struct
{
      int i, j, k;
      double x0, y0, z0;
      int start_trial;
      int end_trial;
} trial_arg_t;
. . .
// koriscenje
pthread_t threads[num_threads];
      trial_arg_t args[num_threads];
      int trials_per_thread = N / num_threads;
      int remainder = N % num_threads;
      int current = 0;

      for (int t = 0; t < num_threads; t++)
      {
          int start = current;
          int count = trials_per_thread + (t < remainder ? 1 : 0);
          int end = start + count;
          current = end;

          args[t].i = i;
          args[t].j = j;
          args[t].k = k;
          args[t].x0 = x;
          args[t].y0 = y;
          args[t].z0 = z;
          args[t].start_trial = start;
          args[t].end_trial = end;

          pthread_create(&threads[t], NULL, trial_worker, &args[t]);
      }
```

```
for (int t = 0; t < num_threads; t++)
{
    pthread_join(threads[t], NULL);
} . . .
```

Na kraju, koristeći **`pthread_join`**, veštački ćemo napraviti sinhronizaciju na barijeri, kako bi redukovali grešku, a da koristimo konzistentne vrednosti.

2.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije ovog problema.

2.3.1. 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: N=1000, num_threads=1
1000    0.009877    1.313446
TEST END
```

Izvršavanje za argumente 1000 i jednu nit

```
TEST: N=1000, num_threads=2
1000    0.009877    0.665669
TEST END
```

Izvršavanje za argumente 1000 i dve niti

```
TEST: N=1000, num_threads=4
1000    0.009877    0.358691
TEST END
```

Izvršavanje za argumente 1000 i četiri niti

```
TEST: N=1000, num_threads=8
1000    0.009877    0.203794
TEST END
```

Izvršavanje za argumente 1000 i osam niti

```
TEST: N=1000, num_threads=16
1000    0.009877    0.175904
TEST END
```

Izvršavanje za argumente 1000 i šestnaest niti

```
TEST: N=5000, num_threads=1
5000    0.005196    6.424804
TEST END
```

Izvršavanje za argumente 5000 i jednu nit

```
TEST: N=5000, num_threads=2
5000    0.005196    3.265266
TEST END
```

Izvršavanje za argumente 5000 i dve niti

```
TEST: N=5000, num_threads=4
5000    0.005196    1.730919
TEST END
```

Izvršavanje za argumente 5000 i četiri niti

```
TEST: N=5000, num_threads=8
5000    0.005196    1.104875
TEST END
```

Izvršavanje za argumente 5000 i osam niti

```
TEST: N=5000, num_threads=16
5000    0.005196    0.915042
TEST END
```

Izvršavanje za argumente 5000 i šestnaest niti

```
TEST: N=10000, num_threads=1
10000    0.005853    12.922155
TEST END
```

Izvršavanje za argumente 10000 i jednu nit

```
TEST: N=10000, num_threads=2
10000    0.005853    6.504489
TEST END
```

Izvršavanje za argumente 10000 i dve niti

```
TEST: N=10000, num_threads=4
10000    0.005853    3.436260
TEST END
```

Izvršavanje za argumente 10000 i četiri niti

```
TEST: N=10000, num_threads=8
10000    0.005853    2.172282
TEST END
```

```
```

Izvršavanje za argumente 10000 i osam niti

```
TEST: N=10000, num_threads=16
10000    0.005853    1.782619
TEST END
```

Izvršavanje za argumente 10000 i šestnaest niti

```
TEST: N=20000, num_threads=1
20000    0.004591    25.731262
TEST END
```

Izvršavanje za argumente 20000 i jednu nit

```
TEST: N=20000, num_threads=2
20000    0.004591    12.923482
TEST END
```

Izvršavanje za argumente 20000 i dve niti

```
TEST: N=20000, num_threads=4
20000    0.004591    7.037356
TEST END
```

Izvršavanje za argumente 20000 i četiri niti

```
TEST: N=20000, num_threads=8
20000    0.004591    4.305316
TEST END
```

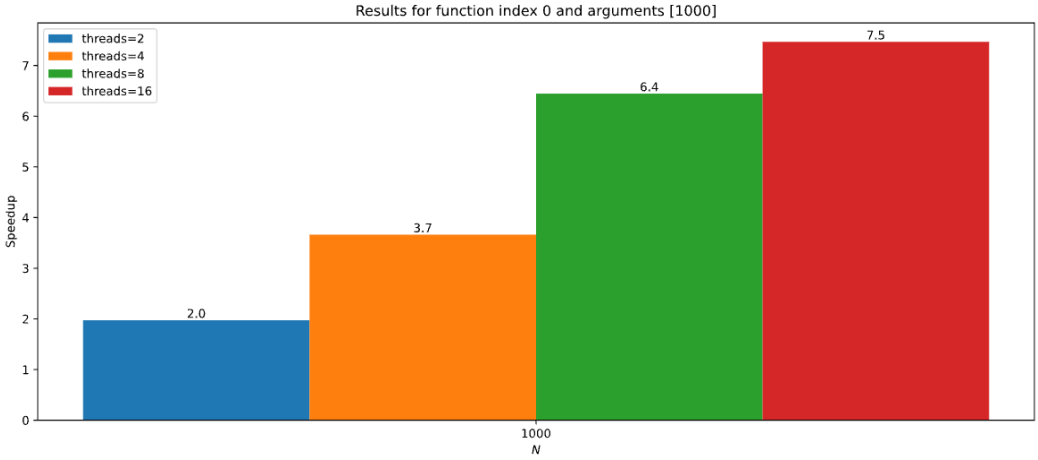
Izvršavanje za argumente 20000 i osam niti

```
TEST: N=20000, num_threads=16
20000    0.004591    3.507318
TEST END
```

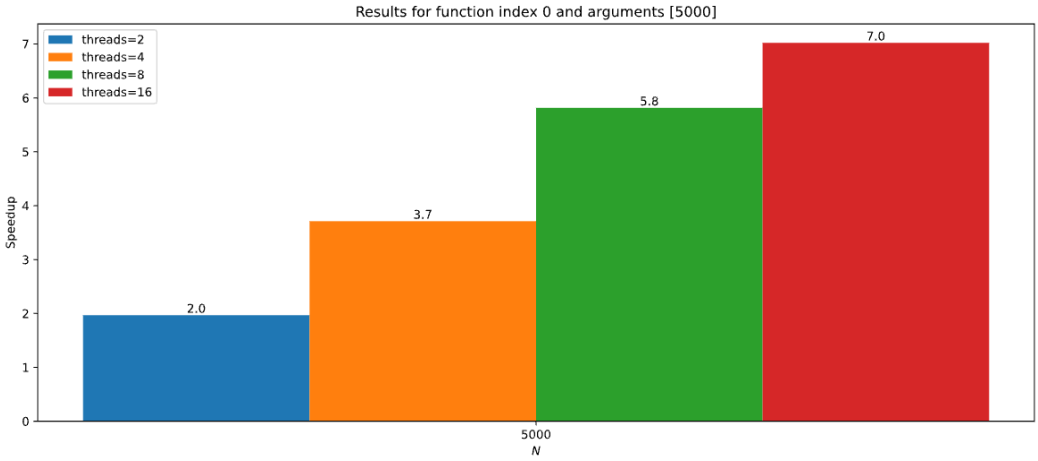
Izvršavanje za argumente 20000 i šestnaest niti

2.3.2. Grafici ubrzanja 1D problema

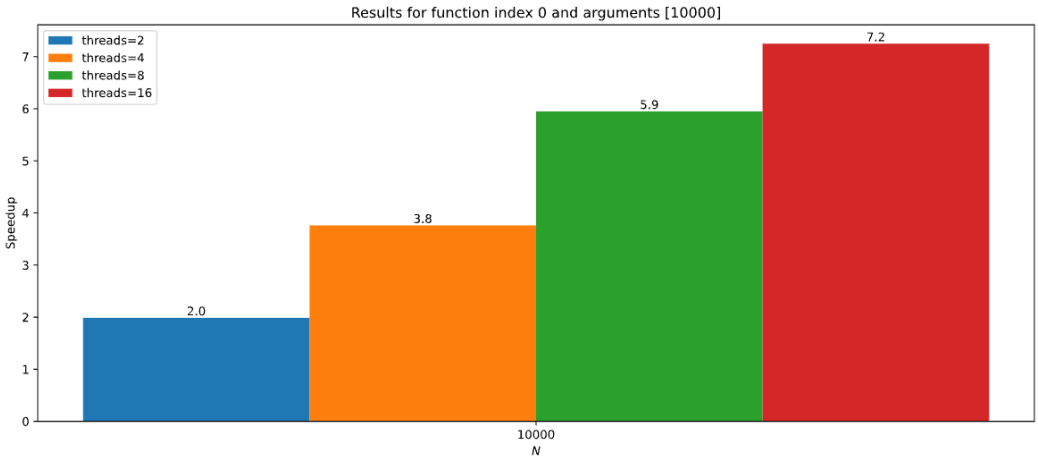
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



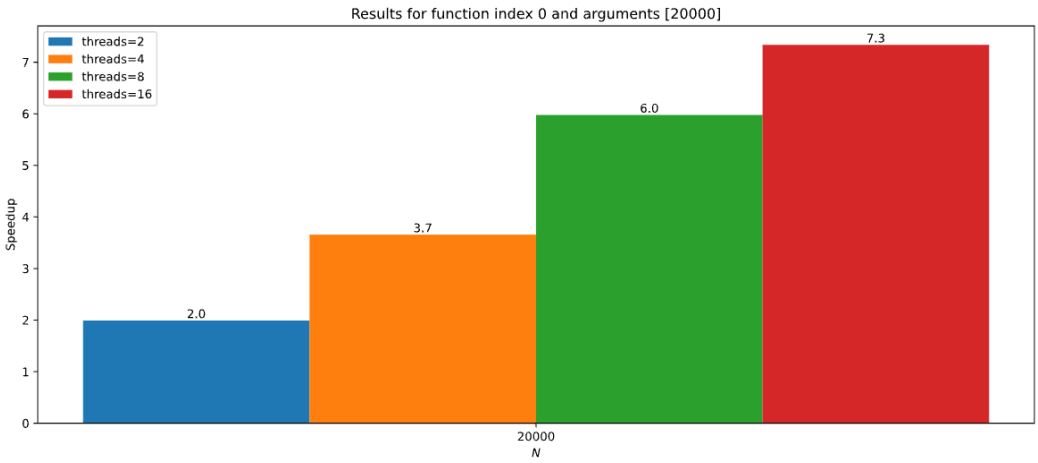
Grafik ubrzanja za različit broj niti, sa argumentom 1000.



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.

2.3.3. 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: N=1000, num_threads=1
1000    0.025061    1.019947
TEST END
```

Izvršavanje za argumente 1000 i jednu nit

```
TEST: N=1000, num_threads=2
1000    0.025061    0.544432
TEST END
```

Izvršavanje za argumente 1000 i dve niti

```
TEST: N=1000, num_threads=4
1000    0.025061    0.282622
TEST END
```

Izvršavanje za argumente 1000 i četiri niti

```
TEST: N=1000, num_threads=8
1000    0.025061    0.158677
TEST END
```

Izvršavanje za argumente 1000 i osam niti

```
TEST: N=1000, num_threads=16
1000    0.025061    0.112929
TEST END
```

Izvršavanje za argumente 1000 i šestnaest niti

```
TEST: N=5000, num_threads=1
5000    0.022837    4.892267
TEST END
```

Izvršavanje za argumente 5000 i jednu nit

```
TEST: N=5000, num_threads=2
5000    0.022837    2.498233
TEST END
```

Izvršavanje za argumente 5000 i dve niti

```
TEST: N=5000, num_threads=4
5000    0.022837    1.301423
TEST END
```

Izvršavanje za argumente 5000 i četiri niti

```
TEST: N=5000, num_threads=8
5000    0.022837    0.828646
TEST END
```

Izvršavanje za argumente 5000 i osam niti

```
TEST: N=5000, num_threads=16
5000    0.022837    0.582808
TEST END
```

Izvršavanje za argumente 5000 i šestnaest niti

```
TEST: N=10000, num_threads=1
10000    0.022576    9.685227
TEST END
```

Izvršavanje za argumente 10000 i jednu nit

```
TEST: N=10000, num_threads=2
10000    0.022576    4.932821
TEST END
```

Izvršavanje za argumente 10000 i dve niti

```
TEST: N=10000, num_threads=4
10000    0.022576    2.649138
TEST END
```

Izvršavanje za argumente 10000 i četiri niti

```
TEST: N=10000, num_threads=8
10000    0.022576    1.705549
TEST END
```

Izvršavanje za argumente 10000 i osam niti

```
TEST: N=10000, num_threads=16
10000    0.022576    1.117289
TEST END
```

Izvršavanje za argumente 10000 i šestnaest niti

```
TEST: N=20000, num_threads=1
20000    0.021923    19.430700
TEST END
```

Izvršavanje za argumente 20000 i jednu nit

```
TEST: N=20000, num_threads=2
20000    0.021923    9.845072
TEST END
```

Izvršavanje za argumente 20000 i dve niti

```
TEST: N=20000, num_threads=4
20000    0.021923    5.416759
TEST END
```

Izvršavanje za argumente 20000 i četiri niti

```
TEST: N=20000, num_threads=8
20000    0.021923    3.387119
TEST END
```

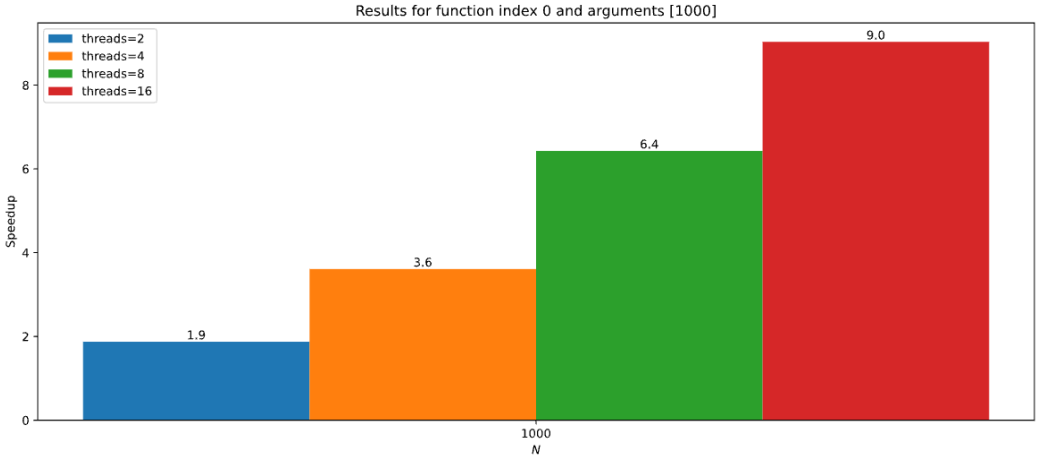
Izvršavanje za argumente 20000 i osam niti

```
TEST: N=20000, num_threads=16
20000    0.021923    2.180354
TEST END
```

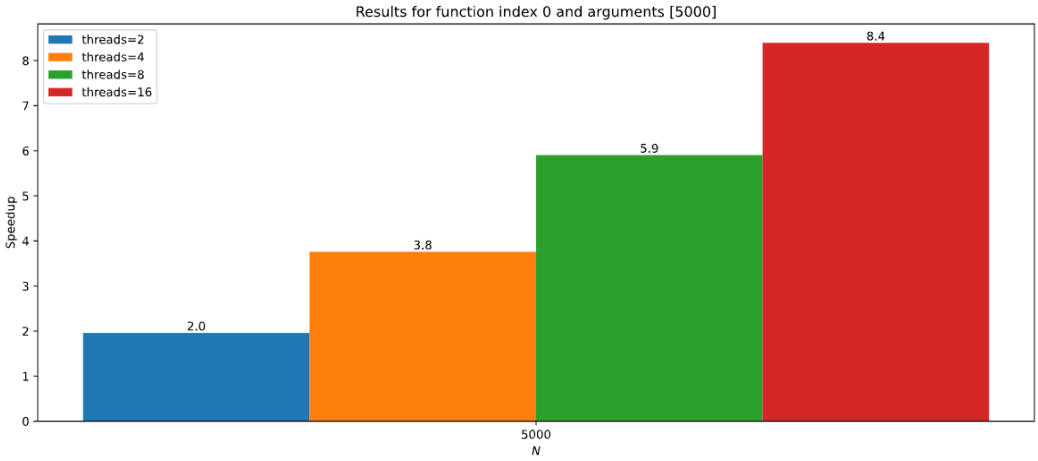
Izvršavanje za argumente 20000 i šestnaest niti

2.3.4. Grafici ubrzanja 2D problema

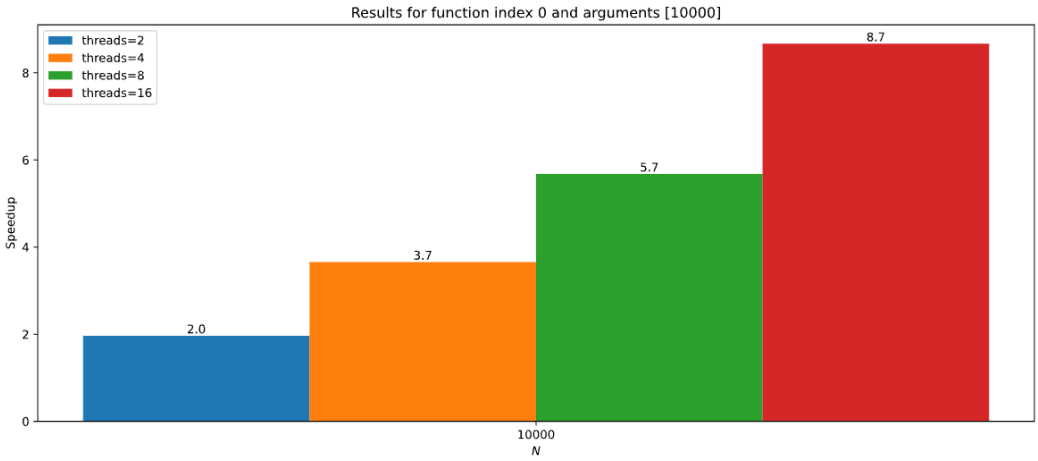
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



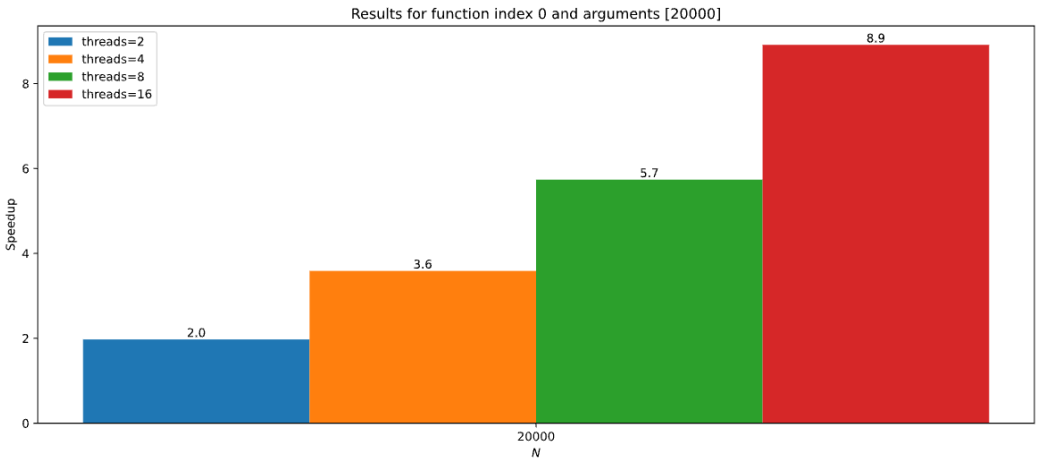
Grafik ubrzanja za različit broj niti, sa argumentom 1000.



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.

2.3.5. 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: N=1000, num_threads=1
1000    0.027319    3.109661
TEST END
```

Izvršavanje za argumente 1000 i jednu nit

```
TEST: N=1000, num_threads=2
1000    0.027319    1.704591
TEST END
```

Izvršavanje za argumente 1000 i dve niti

```
TEST: N=1000, num_threads=4
1000    0.027319    0.887540
TEST END
```

Izvršavanje za argumente 1000 i četiri niti

```
TEST: N=1000, num_threads=8
1000    0.027319    0.497415
TEST END
```

Izvršavanje za argumente 1000 i osam niti

```
TEST: N=1000, num_threads=16
1000    0.027319    0.435561
TEST END
```

Izvršavanje za argumente 1000 i šestnaest niti

```
TEST: N=5000, num_threads=1
5000    0.021057    15.270293
TEST END
```

Izvršavanje za argumente 5000 i jednu nit

```
TEST: N=5000, num_threads=2
5000    0.021057    7.773718
TEST END
```

Izvršavanje za argumente 5000 i dve niti

```
TEST: N=5000, num_threads=4
5000    0.021057    4.351290
TEST END
```

Izvršavanje za argumente 5000 i četiri niti

```
TEST: N=5000, num_threads=8
5000    0.021057    2.794920
TEST END
```

Izvršavanje za argumente 5000 i osam niti

```
TEST: N=5000, num_threads=16
5000    0.021057    1.861610
TEST END
```

Izvršavanje za argumente 5000 i šestnaest niti

```
TEST: N=10000, num_threads=1
10000    0.020074    30.357086
TEST END
```

Izvršavanje za argumente 10000 i jednu nit

```
TEST: N=10000, num_threads=2
10000    0.020074    15.369304
TEST END
```

Izvršavanje za argumente 10000 i dve niti

```
TEST: N=10000, num_threads=4
10000    0.020074    8.721972
TEST END
```

Izvršavanje za argumente 10000 i četiri niti

```
TEST: N=10000, num_threads=8
10000    0.020074    5.428834
TEST END
```

Izvršavanje za argumente 10000 i osam niti

```
TEST: N=10000, num_threads=16
10000    0.020074    3.594014
TEST END
```

Izvršavanje za argumente 10000 i šestnaest niti

```
TEST: N=20000, num_threads=1
20000    0.020383    60.179913
TEST END
```

Izvršavanje za argumente 20000 i jednu nit

```
TEST: N=20000, num_threads=2
20000    0.020383    30.840454
TEST END
```

Izvršavanje za argumente 20000 i dve niti

```
TEST: N=20000, num_threads=4
20000    0.020383    17.410725
TEST END
```

Izvršavanje za argumente 20000 i četiri niti

```
TEST: N=20000, num_threads=8
20000    0.020383    10.750601
TEST END
```

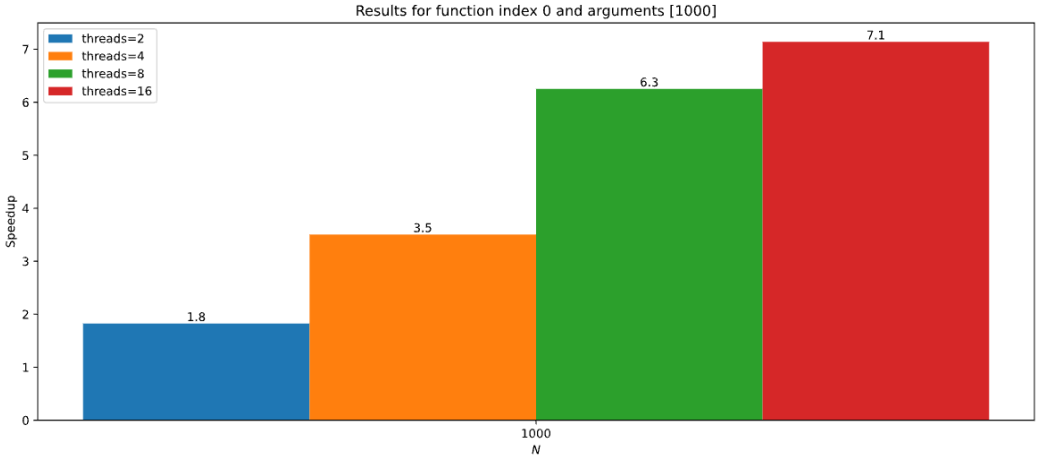
Izvršavanje za argumente 20000 i osam niti

```
TEST: N=20000, num_threads=16
20000    0.020383    7.111311
TEST END
```

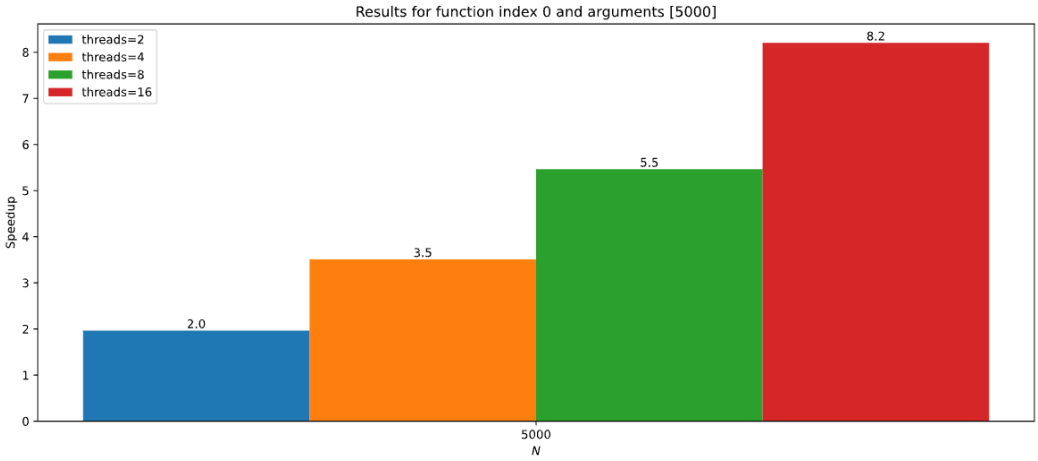
Izvršavanje za argumente 20000 i šestnaest niti

2.3.6. Grafici ubrzanja 3D problema

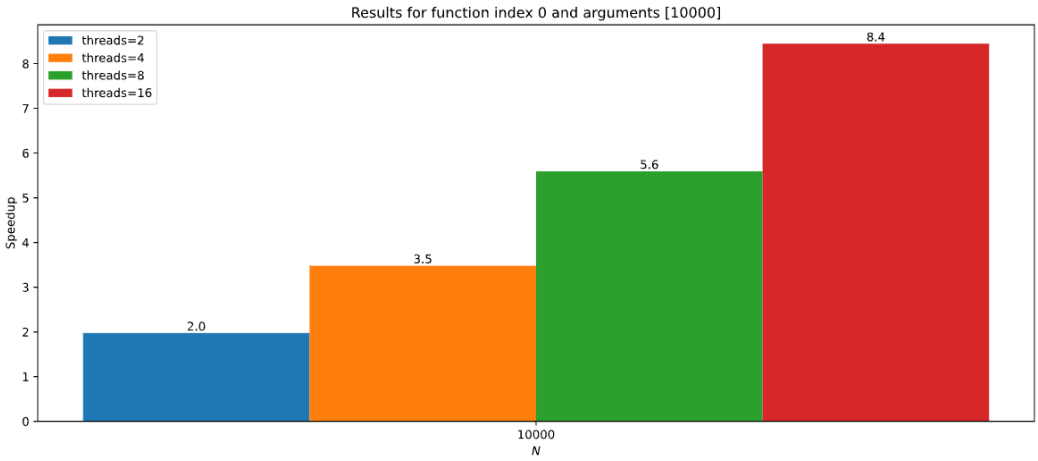
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



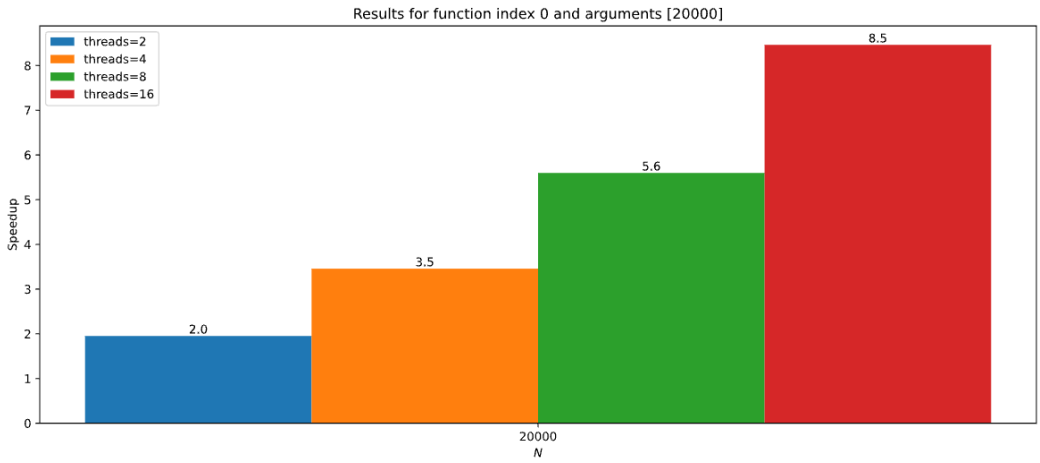
Grafik ubrzanja za različit broj niti, sa argumentom 1000.



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.

2.3.7. Diskusija dobijenih rezultata

Grafici ubrzanja ovom metodom na nekim delovima pokazuju značajno veće ubrzanje u odnosu na rešenje koje koristi OpenMP taskove, a koje najviše liči na priloženo. Jasno se vidi da problem dobro skalira i da bi uz bolju raspodelu brava dobili bolje performanse. Ovu paralelizaciju smatramo uspešnom, a tražena tačnost je takođe u traženom opsegu.