UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

FEYNMAN KAC FORMULA

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:

prof. dr Marko Mišić

Student:

Vuk Lužanin 29/2022

Beograd, januar 2026.

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/.`** Diskusije rešenja su predstavljeni za 3D verziju problema, bez umanjenja opštosti. U svim paralelnim rešenjima, nit generiše slučajne brojeve na osnovu svog lokalnog seed-a, kako bi dobili uniformnije rezultate.

Zbog obima, *.log* fajlovi nisu uključeni u ovaj dokument, dok su svi ključni podaci jasno predstavljeni kroz grafike.

U zavisnosti od trenutnog pokretanja, zabeležena su i nešto veća ubrzanja od navedenih. S obzirom na to da izvođenje skripte koja pokreće sve testove traje znatno dugo, nisu prikazani najbolji pojedinačni rezultati. Prikazani podaci predstavljaju vreme izvršavanja u prosečnom slučaju.

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

Prilikom prevođenja, korišćeni su sledeći komplajlerski flegovi:

- **-Ofast** – maksimalna optimizacija (uključuje -O3 i -ffast-math).
- **-flto** – optimizacija pri linkovanju .
- **-march=native** – kompajlira za arhitekturu procesora na kojem se izvršava, koristeći sve dostupne instrukcije.
- **-funroll-loops** – razmotavanje petlji.
- **-Wno-maybe-uninitialized** – isključivanje upozorenja o potencijalno neinicijalizovanim promenljivama.

- **-floop-interchange** – preuređivanje ugnježdenih petlji (loop interchange).
- **-floop-block**
- **-floop-strip-mine**
- **-fprefetch-loop-arrays**

2. PROBLEM 1 - POASONOVA JEDNAČINA - OPENMP

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 1 korišćenjem Feynman-Kac algoritma za 1D, 2D i 3D verzije ovog problema korišćenjem OpenMP.

2.1. Način paralelizacije [FOR + NOWAIT + BARRIER + MANUELNA REDUKCIJA] – NAJEFIKASNIJE REŠENJE

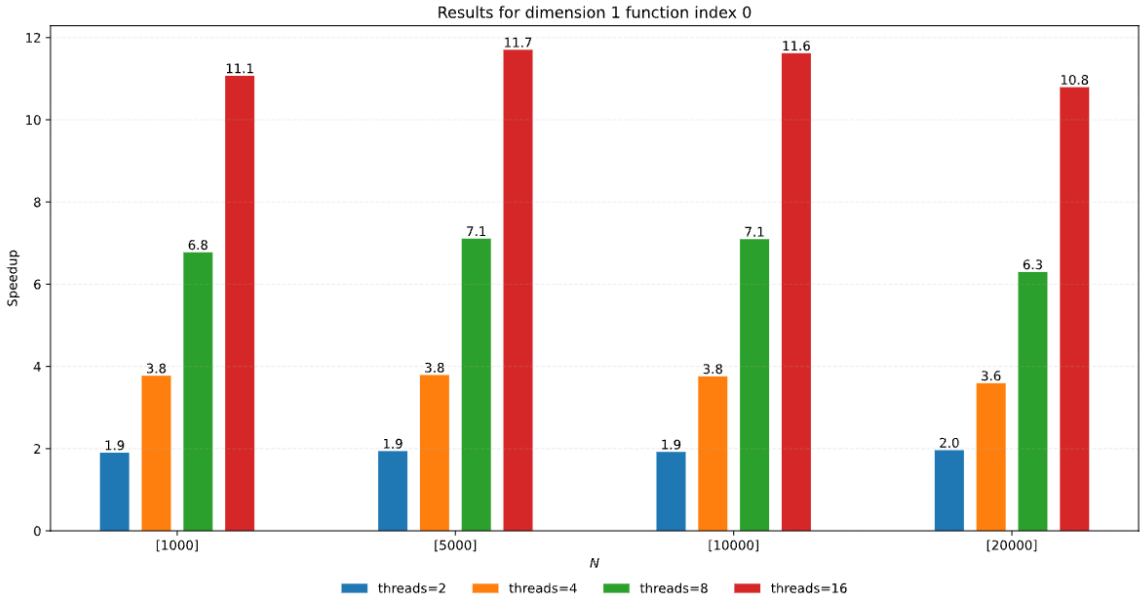
Obilazak elipsoida (iteracije po `i, j, k`) vrše sve niti paralelnog regiona bez podele posla po tačkama mreže. Umesto toga, paralelizacija je ostvarena unutar `trial` petlje, tako da se broj Monte Carlo putanja za svaku tačku deli između niti. Svaka nit računa deo `N` putanja za datu tačku, a rezultati se sabiraju redukcijom u `wt[i][j][k]`. Tačnu vrednost funkcije, ali i brojanje tačaka koje se nalaze unutar elipsoida, izračunaće 1 nit, `single` direktivom sa `nowait` opcijom. Petlja čije iteracije predstavljaju nezavisne putanje iz trenutne tačke, paralelizovana je `for worksharing` direktivom sa `nowait` opcijom. Na kraju petlje se zbog toga nalazi eksplicitna barijera. Sinhronizacija niti je postignuta sabirajućom redukcijom nad elementom rezultujuće matrice `wt[i][j][k]`.

Na kraju, za računanje greške, ponovo obilazimo sve tačke elipse `for worksharing` direktivom sa `collapse(3)` opcijom i redukcijom nad promenljivom `err`.

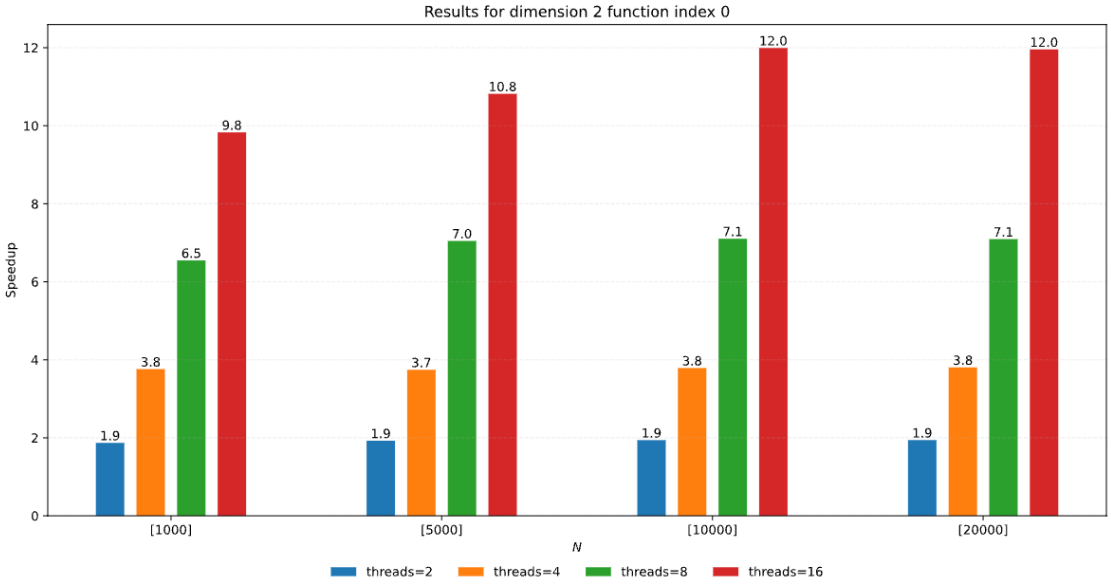
Ovo rešenje loše skalira ako je reč o memorijskom zauzeću, budući da sami radimo redukciju, za svaku tačku elipse moramo alocirati 1 element koji bi čuvao aproksimaciju u svakoj tački. Zbog toga imamo matricu `wt[i][j][k]`.

2.1.1. Grafici dobijenih rezultata

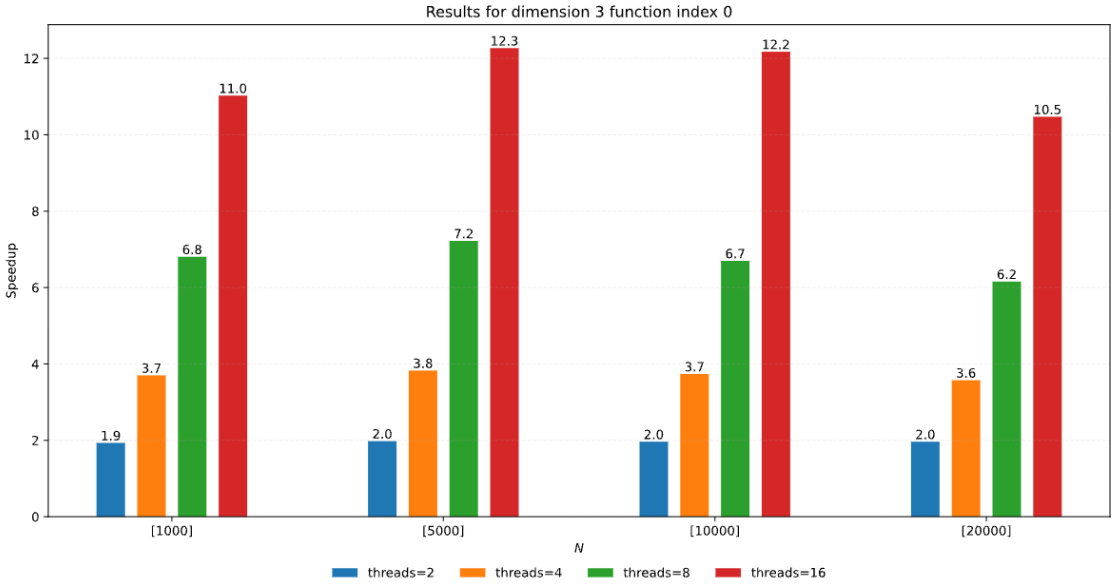
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

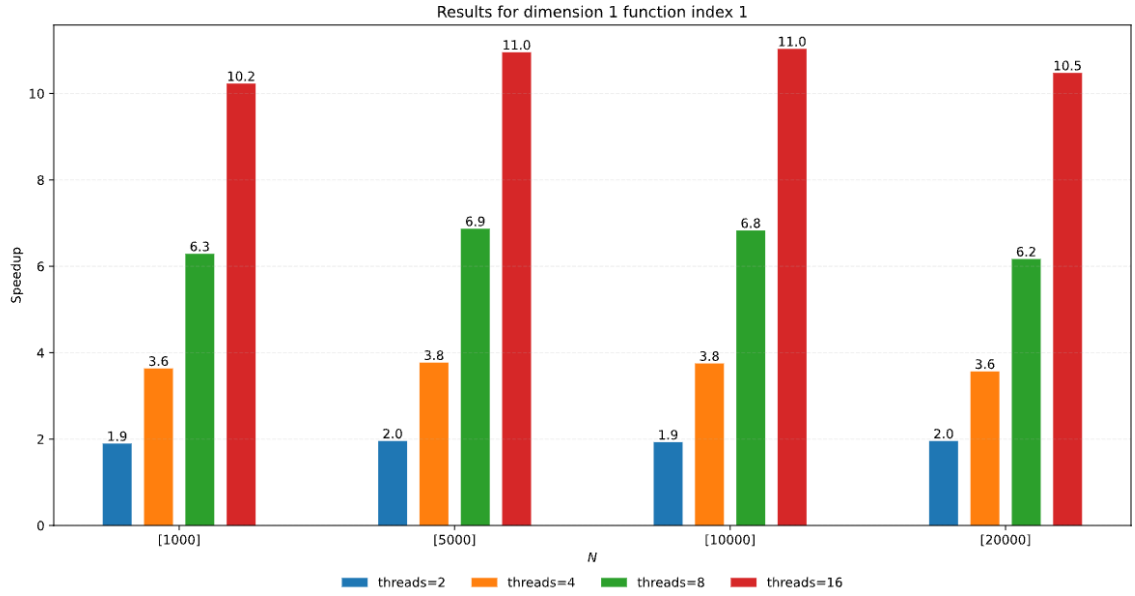
Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** . Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati.

2.2. Način paralelizacije [LOKALNA SUMA + UGRAĐENA REDUKCIJA]

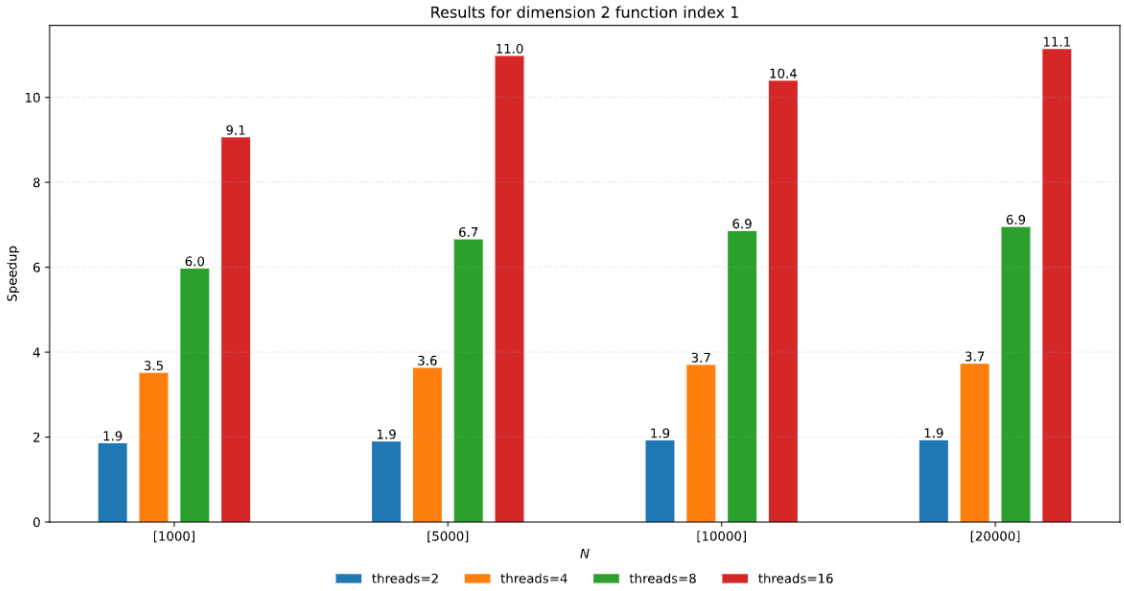
Obilazak tačaka elipsoida nije paralelizovan , već paralelni region pravimo tek nad unutrašnjom petljom koja sumulira putanje iz trenutne tačke. Vrši se podela posla `for` direktivom, uz redukciju nad promenljivom `local_sum`, koja predstavlja aproksimaciju rešenja u toj tački. Za razliku od prethodnog rešenja, u ovom se oslanjamo da redukciju koja je ugrađena u OpenMP, zato nemamo potrebu za dodatnom strukturom u kojoj bismo čuvali rezultate.

2.2.1. Grafici dobijenih rezultata

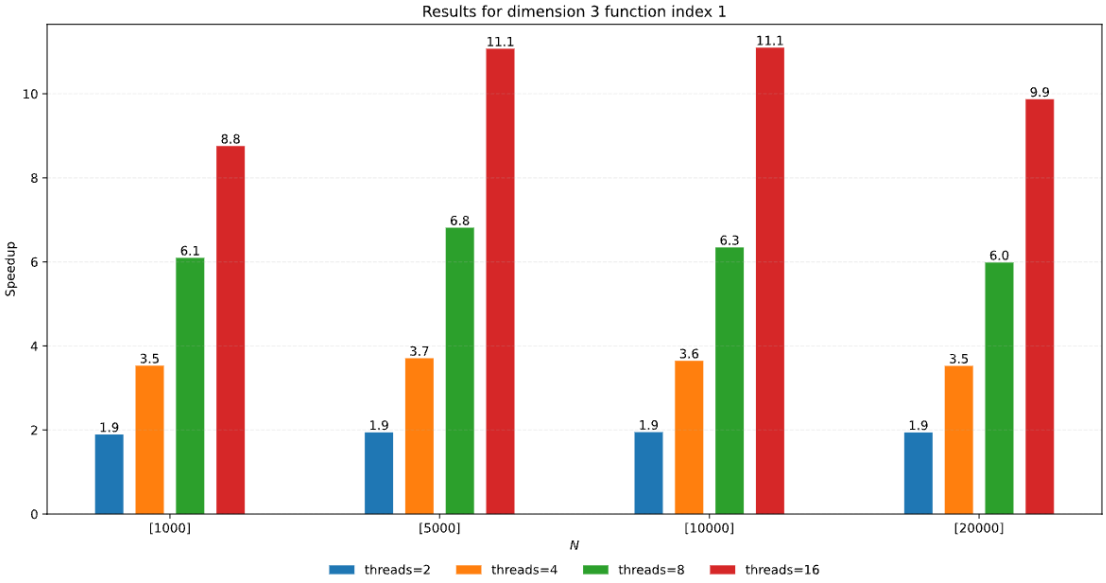
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** . Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati. Ovo rešenje predstavlja 2. po redu najbolje rešenje, kada je reč o vremenu izvršavanja.

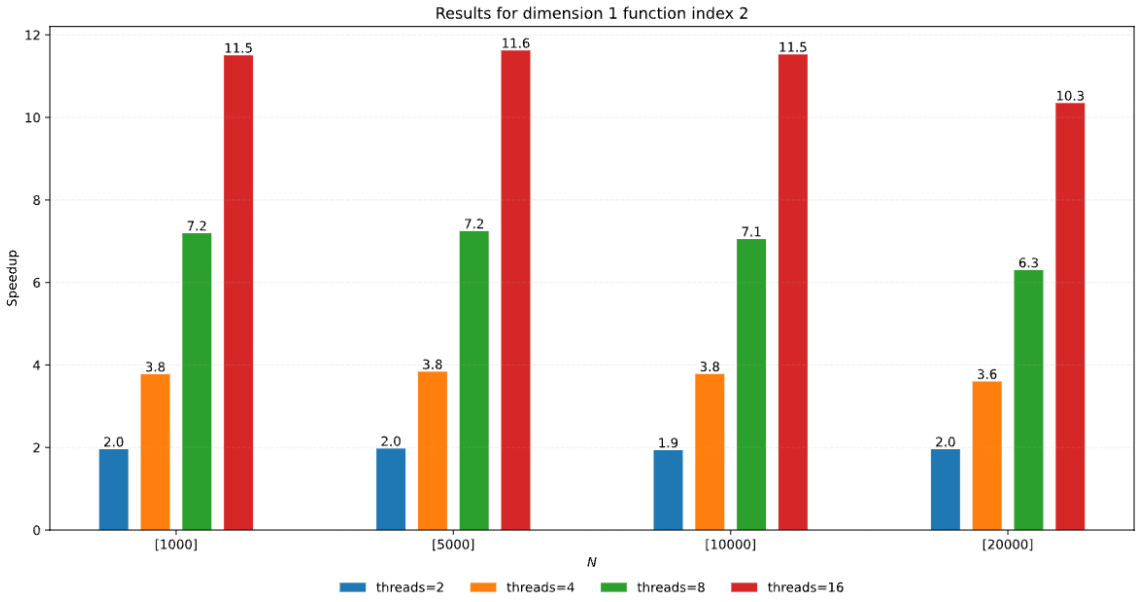
2.3. Način paralelizacije [TASK + ATOMIC]

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 kada je reč o zauzeću memorije. 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.

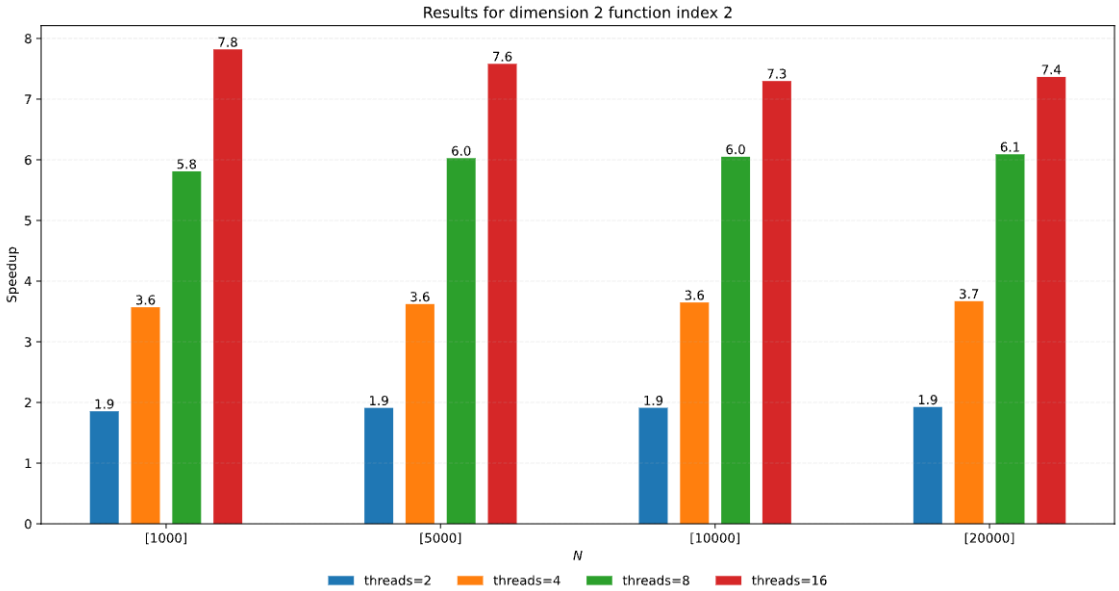
Ovako, 1 putanja iz trenutne tačke predstavlja 1 `task` koji će biti dodeljen niti na izvršavanje. Zbog malih dimenzija problema, ali i zato što želimo rešenje korišćenjem samo `task`-ova, manuelna redukcija na kraju nije paralelizovana.

2.3.1. Grafici dobijenih rezultata

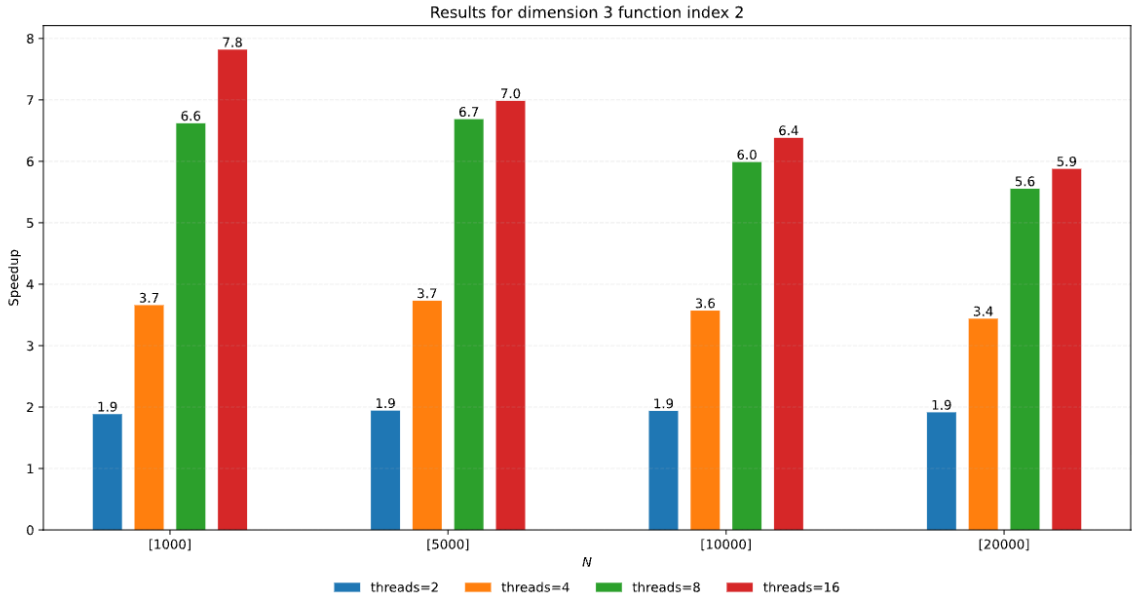
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** . Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati. Uočeno je da, povećavanje broja putanja iz svake tačke, usporava paralelna rešenja kada je reč o 2D i 3D problemu.

2.4. Način paralelizacije [TASK + locks distributed across multiple threads]

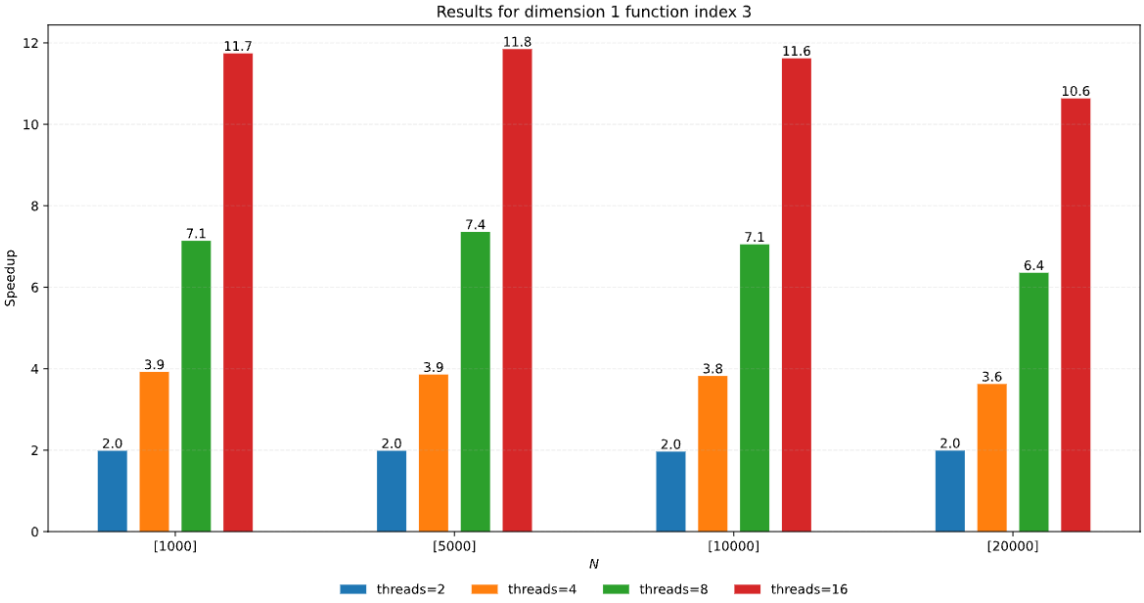
Razlika u odnosu na prethodno rešenje se ogleda u tome što sada postoji fiksan broj **NUM_LOCKS (trenutno 256 – mogu se ispobati i druge vrednosti)** brava na kojima se sinhronizuju niti. Kada nit dođe do kritične sekcije, uzeće indeks brave, koja joj pripada na osnovu pozicije u rešetki (brojača **i, 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 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(i, j, k);
    omp_set_lock(&locks[lock_id]);
    wt[i][j][k] += w;
    omp_unset_lock(&locks[lock_id]);
. . .
```

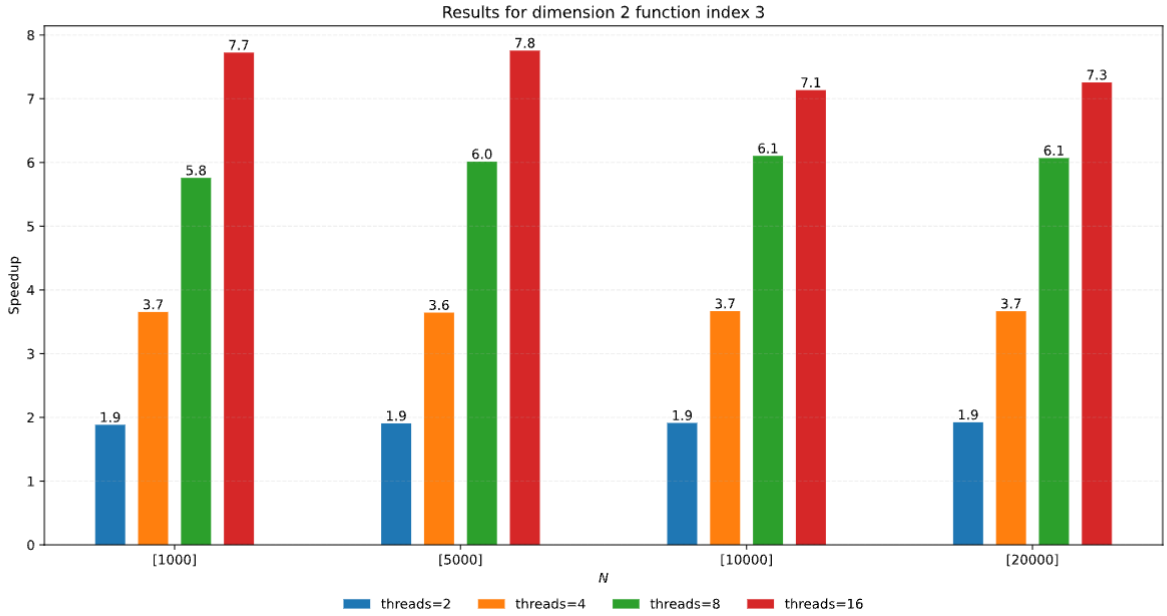
Kao što je navedeno, rešenje se može unaprediti odabirom bolje heš funkcija koju koristi **get_lock_index()**. U razmatranje je potrebno uzeti i prirodu problema, budući da interpoliranjem, za indekse **i, j i k,** brave se više koriste, što je tačka bliža koordinatnom početku, tj. kada su vrednosti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

2.4.1. Grafici dobijenih rezultata

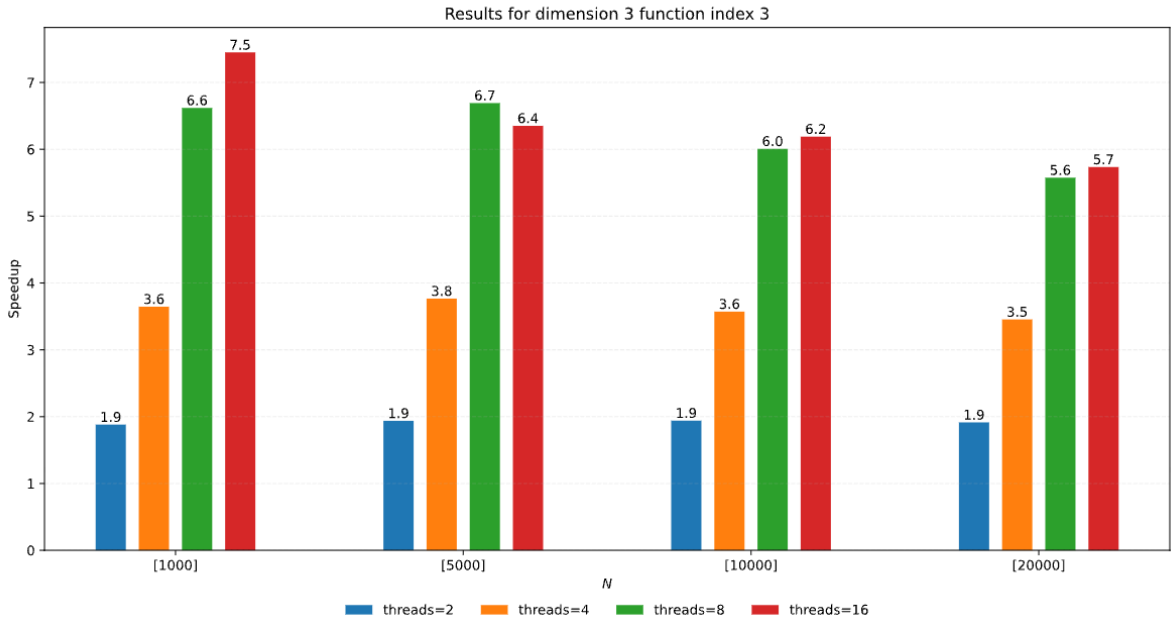
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

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

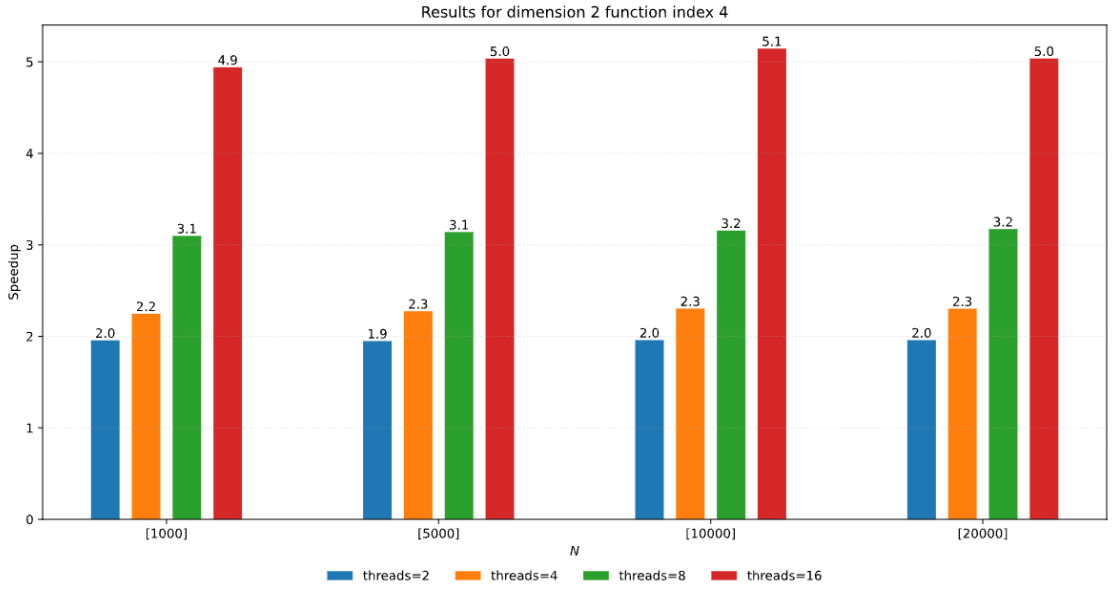
2.5. Način paralelizacije [for collapse of outer loops and reduction of error]

U ovom rešenju je korišćena `for collapse(3)` direktiva nad spoljašnjim petljama koje obilaze tačke elipsoida, dok se redukcija radi nad `n_inside i err` promenljivama. Budući da dimenzija samog prolema skalira u vidu porasta broja putanja iz jedne tačke, a ne u vidu porasta broja tačaka, ovo rešenje ne skalira dobro, jer niti međusobno dobijaju neujednačenu količinu posla, jer za tačke koje se nalaze blizu „ivica“ elipse možemo očekivati značajno manji broj koraka od onih koji se nalaze u njenom centru. Ipak, ukoliko je obrnut slučaj, moguće je očekivati povoljnije rezultate.

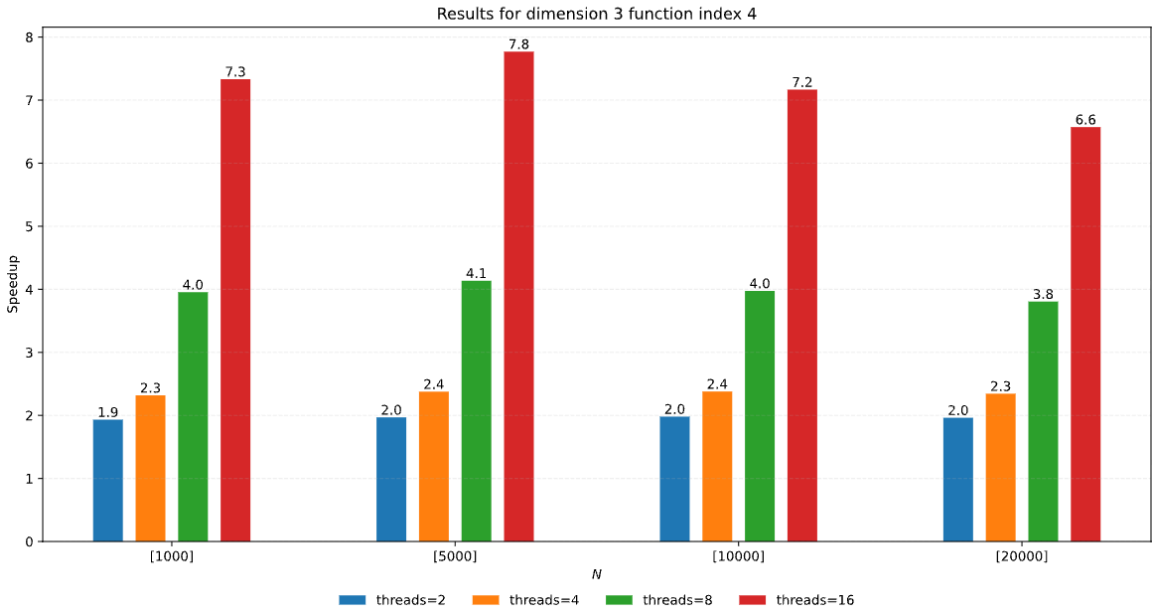
2.5.1. Grafici dobijenih rezultata

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.

Grafik ubrzanja za različit broj niti i putanja za 1D verziju ne postoji, jer se svodi na rešenje koje je razmatrano u sledecem primeru



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

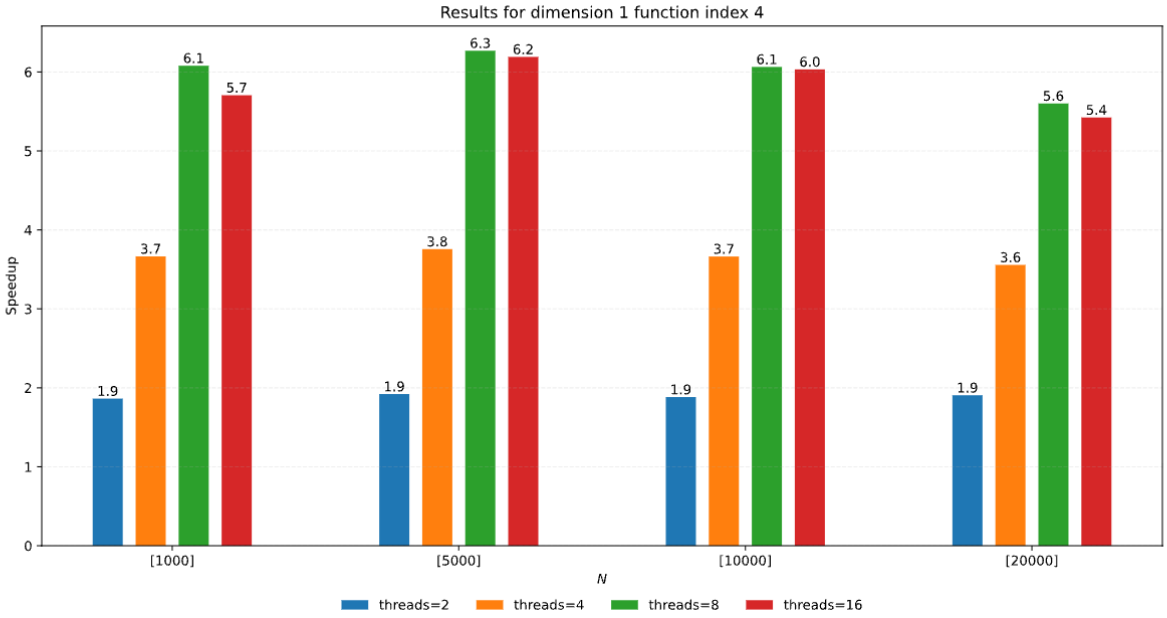
Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** . Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati.

2.6. Način paralelizacije [FOR DYNAMIC]

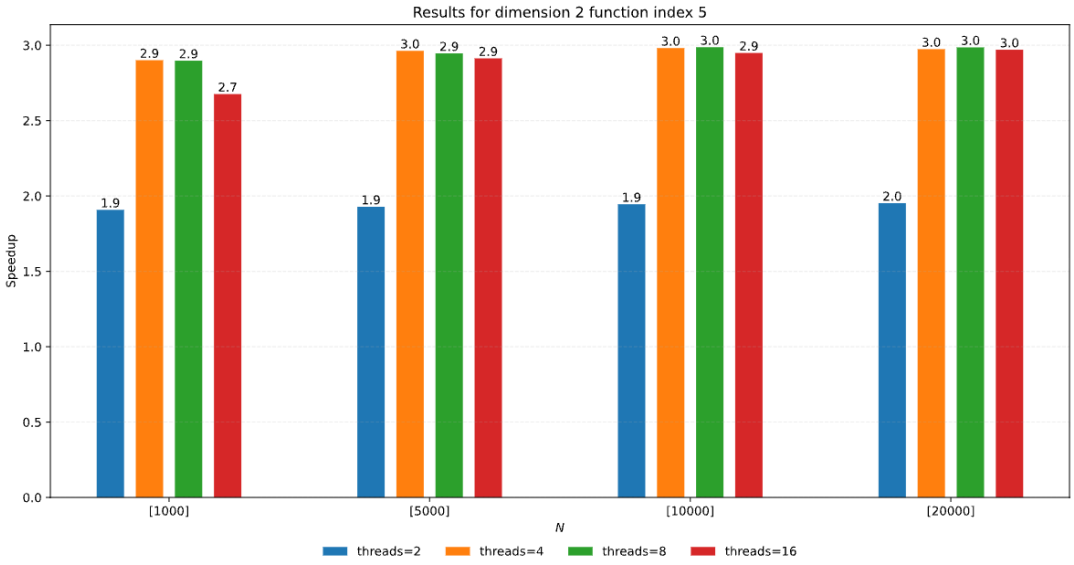
Ovo je verovatno najjednostavnije rešenje koje ima parallel for direktivu samo nad spoljnom petljom koja predstavlja obilazak elipse po jednoj dimenziji i sa schedule(dynamic), dinamički raspoređuje posao nitima. Kao i prethodno, ovo rešenje ne daje dobre rezultate.

2.6.1. Grafici dobijenih rezultata

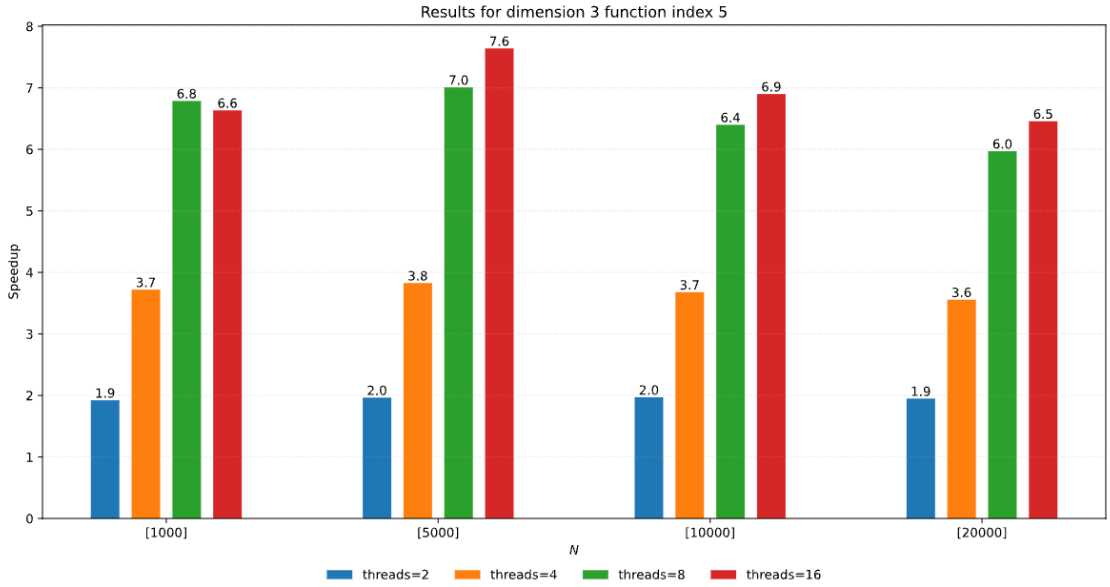
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

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

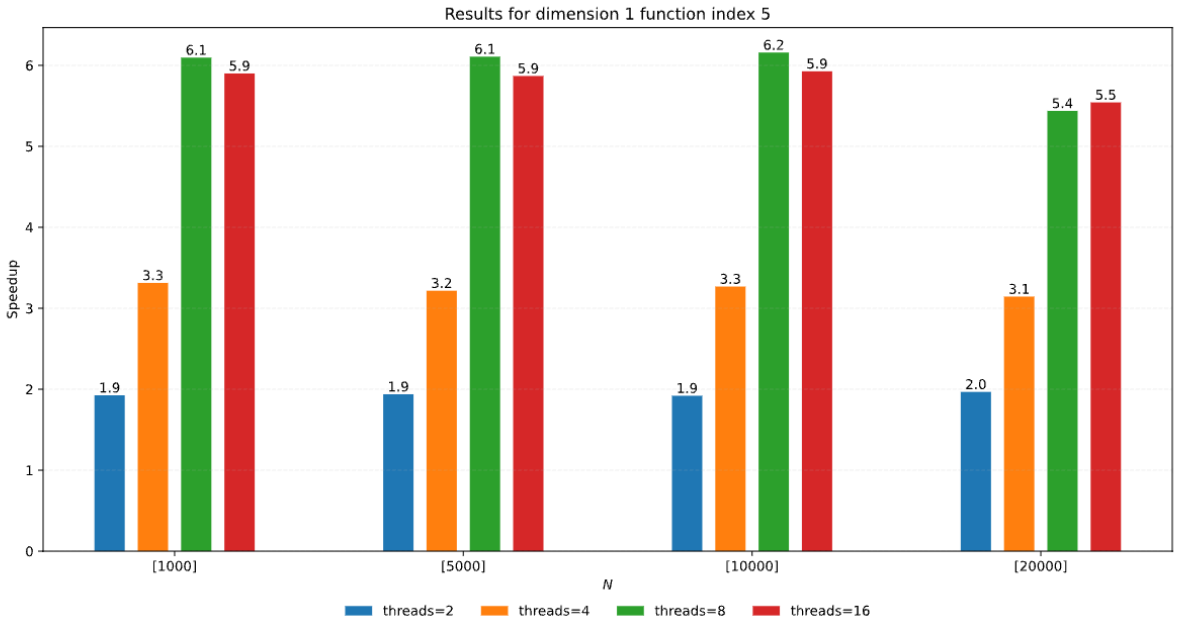
2.7. Način paralelizacije [Spirala]

Struktura algoritma je delimično izmenjena: umesto klasičnog iterativnog obilaska elipse kroz ugnježdene petlje, tačke elipse se obrađuju tako da se prvo prolazi kroz one na njenom obodu, a zatim se nastavlja ka centru. Ovaj pristup omogućava ravnomerniju raspodelu posla među nitima, jer je realno očekivati da će putanje koje počinju na obodu elipse završavati ranije od onih koje kreću iz unutrašnjih tačaka.

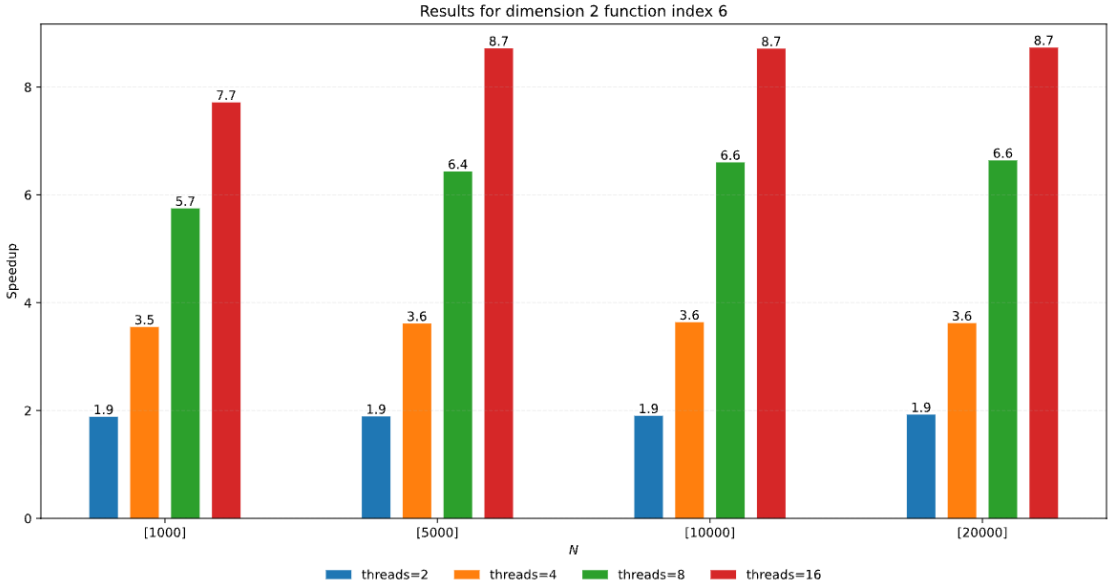
Zbog kompleksnosti rešenja, rešenje za 3D verziju nije završeno.

2.7.1. Grafici dobijenih rezultata

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju

Grafik ubrzanja za različit broj niti i putanja za 3D verziju – nije još urađeno

Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY.** Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati. Za izuzetno neravnomernu količinu posla između niti, moguće je dobiti značajno ubrzanje, međutim, dobijena ubrzanja često variraju u zavisnosti od pokretanja.

3. PROBLEM 2 - POASONOVA JEDNAČINA - PTHREADS

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 2 korišćenjem Feynman-Kac algoritma za 1D, 2D i 3D verzije ovog problema korišćenjem Pthreads.

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.

3.1. Način paralelizacije [statička podela posla]

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:

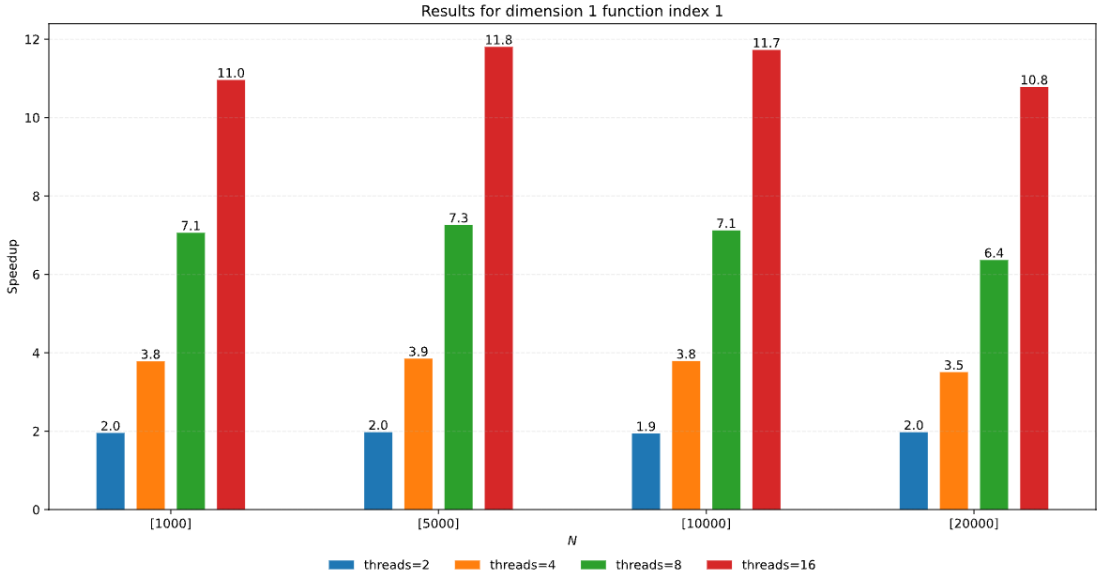
```
    typedef struct
    {
              int i, j, k;
              double x0, y0, z0;
              int start_trial;
              int end_trial;
    } trial_arg_t;
    . . .
  // koriscenje
  pthread_t threads[num_threads];
          trial_arg_t args[num_threads];
          int trials_per_thread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

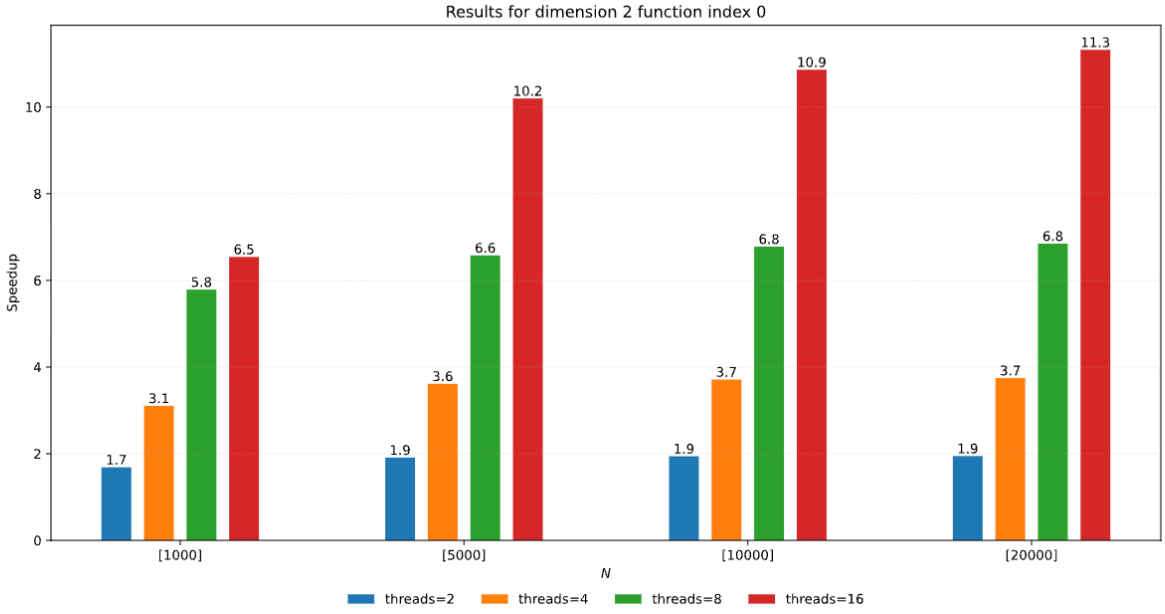
Na kraju, koristeći **`pthread_join`**, veštački ćemo napraviti sinhronizaciju na barijeri, kako bi redukovali grešku, a da koristimo konzistentne vrednosti.

3.1.1. Grafici dobijenih rezultata

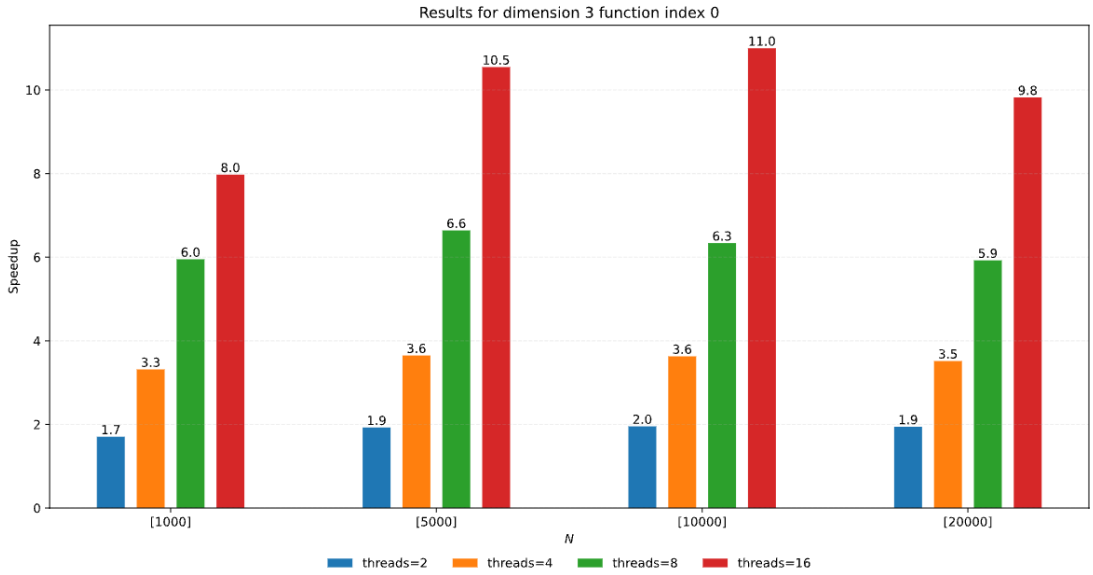
Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju



Grafik ubrzanja za različit broj niti i putanja za 2D verziju



Grafik ubrzanja za različit broj niti i putanja za 3D verziju

Diskusija dobijenih rezultata

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

3.2. Način paralelizacije [dinamička podela posla – samo 1D za sada]

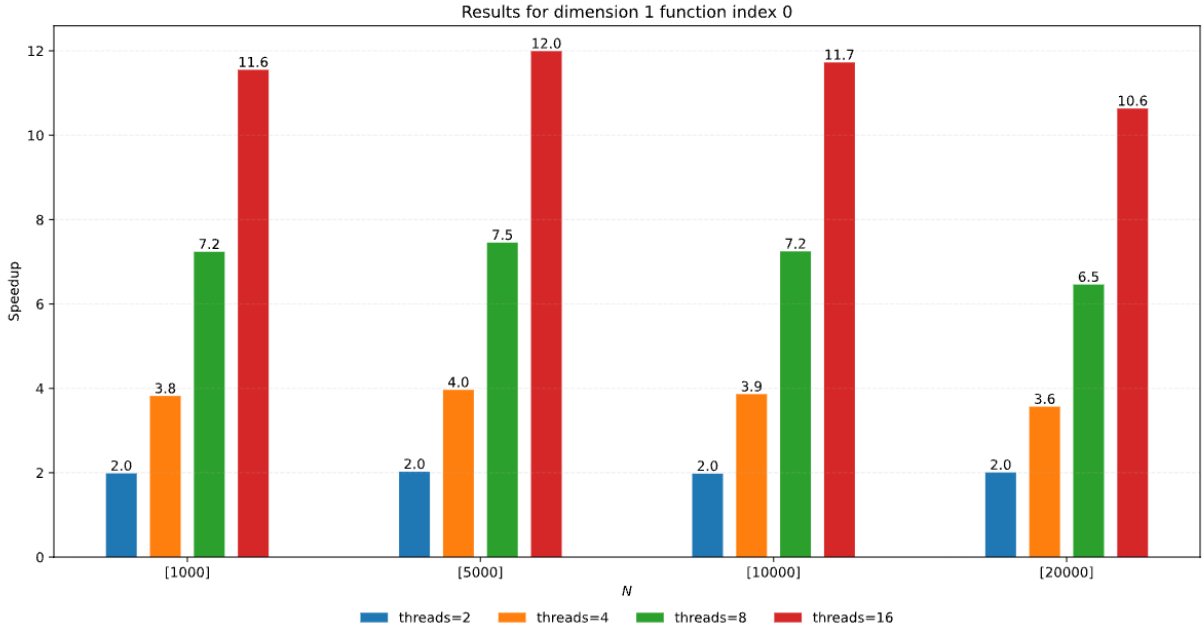
Za razliku od prethodnog rešenja, sada dinamički nitima raspoređujemo putanje iz tačke koje treba izvšiti. Ovo je omogućeno atomičnom funkcijom koja služi kao globalni deljeni brojač oko koga će se niti sinhronizovati.

```
while ((t = __atomic_fetch_add(&global_trial_index, 1, __ATOMIC_RELAXED)) < total_trials) {…}
```

Ovo rešenje pokazuje dobre rezultate jer bolje amortizuje neuravnoteženost količine posla. Budući da rešenje daje slične rezultate, kao ono sa statičkom podelom, za sada nisu implementirane verzije za 2D I 3D verzije problema.

3.2.1. Grafici dobijenih rezultata

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za 1D verziju

Grafik ubrzanja za različit broj niti i putanja za 2D verziju- nema

Grafik ubrzanja za različit broj niti i putanja za 3D verziju - nema

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. Razumno je očekivati bolje performanse za 2D i 3D verzije problema.

4. PROBLEM 3 – PRIMENE FEYNMAN KAC FORMULE

U okviru ovog poglavlja je dat kratak izveštaj u vezi realnih primena Feynman Kac formule.

4.1. Heat equation – 4. sa spiska

https://github.com/urbainvaes/computational_stochastic_processes/blob/master/w6_applications_of_sdes.py

U ovoj implementaciji razmatra se primena Feynman Kac formule za numeričko rešavanje jednačine toplote. Simulacija se pokreće iz više početnih prostornih tačaka, označenih sa n_{mc}, koje su ravnomerno raspoređene u posmatranom intervalu. Iz svake početne tačke generiše se Mnezavisnih realizacija Brownovog kretanja, pri čemu svaka putanja ima unapred zadat fiksan broj koraka N.

Za razliku od prethodnih implementacija, kretanje čestice se ne zaustavlja pri izlasku iz određenog domena, već se putanje uvek prate do kraja simulacije. U svakom vremenskom koraku i za svaku početnu tačku računa se srednja vrednost početnog uslova evaluiranog duž svih nezavisnih putanja. Kako je početni uslov indikatorska funkcija intervala $[-1,1]$, dobijena srednja vrednost predstavlja procenu verovatnoće da se čestica u tom vremenskom trenutku nalazi unutar navedenog intervala.

Algoritam je direktno preveden iz Python-a u programski jezik C. Struktura algoritma ostala je nepromenjena, dok su pojedini detalji prilagođeni načinu rada u jeziku C.

Za generisanje slučajnih brojeva koristi se standardna funkcija `rand()`, koja proizvodi uniformno raspodeljene brojeve, u kombinaciji sa Box-Muller transformacijom kako bi se dobili slučajni brojevi sa standardnom normalnom raspodelom (tako je I u originalnom rešenju). Ovakav generator je neophodan za simulaciju Brownovog kretanja, jer njegovi inkrementi zahtevaju normalno raspodeljene slučajne promenljive. Generator je inicijalizovan fiksnim seed-om, čime je obezbeđena ponovljivost rezultata.

Paralelizacije je izvršena korišćenjem OpenMP.

U paralelnoj verziji svaka nit poseduje lokalni akumulator u koji skladišti svoj deo rezultata. Paralelizovana je spoljna petlja koja obilazi putanje iz trenutne tačke (po M), korišćenjem `for schedule(dynamic)` direktive. Na taj način, jedna nit će obrađivati jednu, i-tu putanju za sve tačke, kroz N koraka. Sledeća, i+1… Redukcija lokalnih rezultata je obezbeđena korišćenjem `critical` direktive.

U sekvencijalnoj verziji korišćen je standardni `rand()` generator, koji nije bezbedan za rad sa više niti, pa je u paralelnoj verziji zamenjen thread safe generator zasnovan na `rand_r`, pri čemu svaka nit koristi svoj jedinstveni seed. Takođe, inicijalizacija i normalizacija rezultujuće matrice takođe su paralelizovane (`for collapse(2)`) kako bi se dodatno smanjilo vreme izvršavanja, ali i podstaklo bolje raspoređivanje podataka po keševima procesora.

Ovo su počene vrednosti parametara koje su navedene u originalnom problemu:

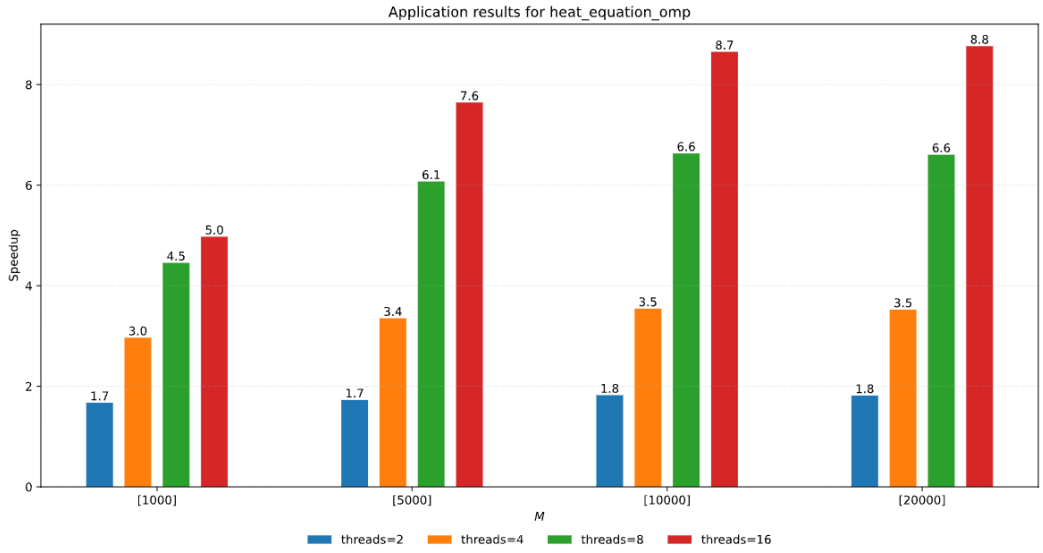
```
const int N = 100; /* time steps, number of steps until the end of movement */

const int M  = 1000;     /* Monte Carlo paths from one point */

const int n_mc = 20;  /* spatial points, number of start points */
```

Kako bi testirali kako paralelno rešenje skalira, svaki od parametara je skaliran, dok su ostala dva fiksirana na početne vrednosti. Ovo je urađeno i kako bi videli kako povećanje svakog od parametara doprinosi konačnom ubrzanju. Da smo sve parametre istovremeno povećali, ne bismo bili u mogućnosti da zaključimo tako nešto.

4.1.1. Grafici dobijenih rezultata – variramo broj putanja iz svake tačke – M, dok su:

```
const int N = 100;

const int n_mc = 20;
```

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi M, broj putanja iz svake tačke.



Grafik ubrzanja za različit broj niti i putanja za heat equation problem

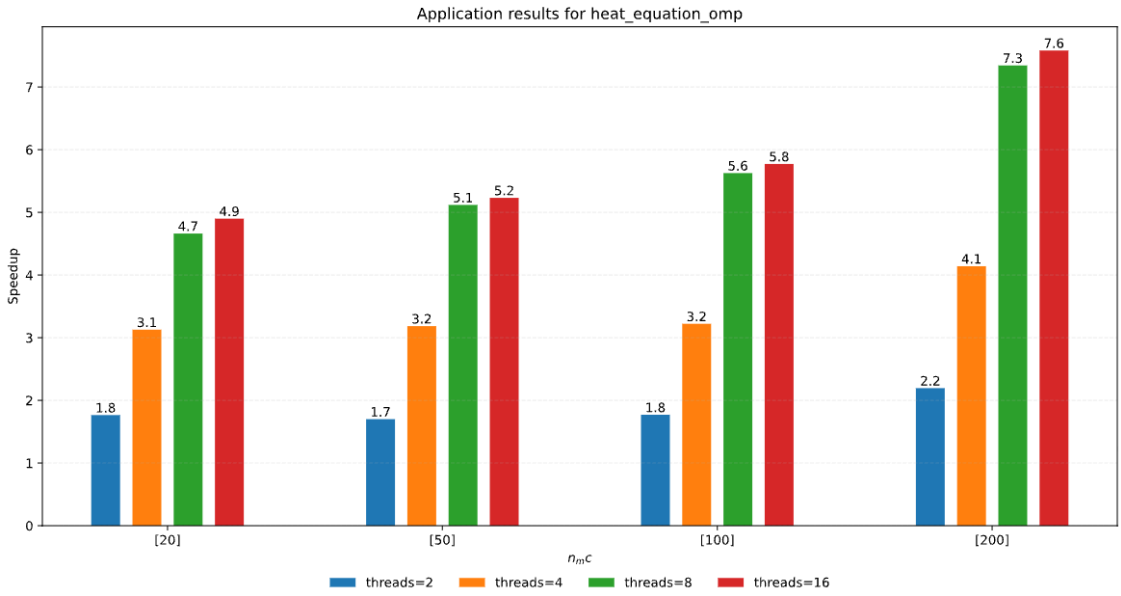
Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY**. Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati, što je i očekivano, budući da podelu posla vršimo nad spoljnom petljom, koja obilazi putanje-M. Za vrednosti oko M=1000 putanja iz svake tačke, primećeno je da se dobijaju lošije performanse sa većim brojem niti. Jasno se može zaključiti da je dimenzija početnog problema premala za paralelizaciju, pa sam overhead kreiranja niti troši značajan deo vremena izvršavanja. Zbog toga je odlučeno variranje parametara u cilju sagledavanja kako problem skalira.

4.1.3. Grafici dobijenih rezultata – variramo broj tačaka iz kojih započinjemo kretanje – n_mc, dok su:

```
const int N = 100;
const int M = 1000;
```

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi n_mc, broj tačaka.



Grafik ubrzanja za različit broj niti i početnih tačaka za heat equation problem

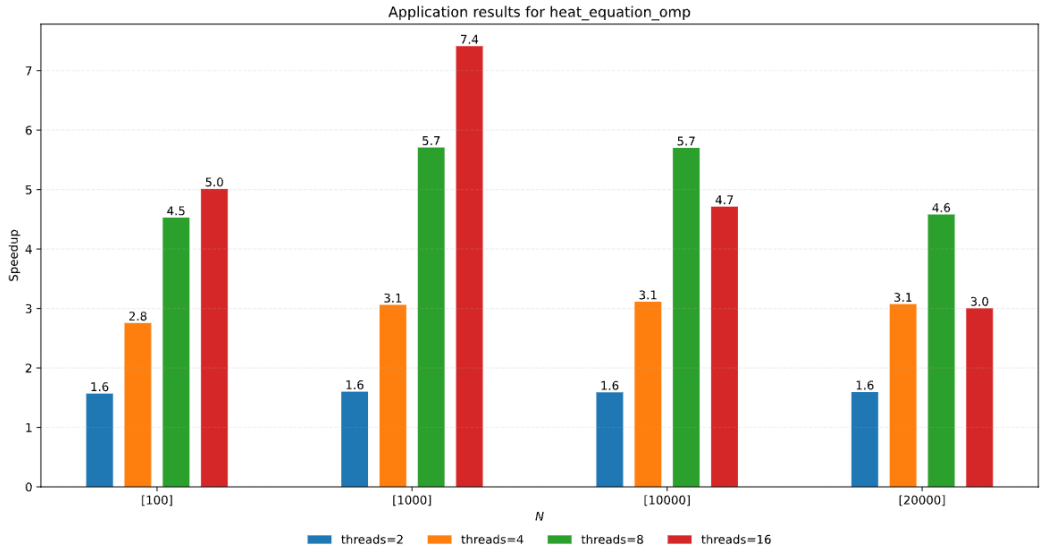
Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY**. Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati. Za vrednosti oko n_mc=20 putanja iz svake tačke, primećeno je da se dobijaju lošije performanse sa većim brojem niti. Jasno se može zaključiti da je dimenzija početnog problema premala za paralelizaciju, pa sam overhead kreiranja niti troši značajan deo vremena izvršavanja. Zbog toga je odlučeno variranje parametara u cilju sagledavanja kako problem skalira. Vidimo da tek ako ovu dimenziju problema povećamo za red veličine, dobijamo značajnije ubrzanje, pa postoji potencijal za daljim povećanjem broja tačaka.

4.1.5. Grafici dobijenih rezultata – variramo broj koraka simulacije kretanja – N, dok su:

```
const int n_mc = 20;
const int M = 1000;
```

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj tačaka.



Grafik ubrzanja za različit broj niti i broja koraka za heat equation problem

Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu odstupanja ±**ACCURACY** . Za veći broj niti dobija se vidljivo veće ubrzanje, što pokazuje da se problem može uspešno skalirati. Jasno se vidi pad performansi kada se broj koraka poveća na 10000. Očigledno, to unosi preveliku neuravnoteženost u podeli posla, jer ovaj obilazak predstavlja unutrašnju petlju, dok paralelizujemo spoljašnju. Zbog toga, N i n_mc moraju biti srazmerni veličini M, ukoliko želimo bolje performanse.

4.2. Girsanov importance sampling– 4. sa spiska

https://github.com/urbainvaes/computational_stochastic_processes/blob/master/w6_applications_of_sdes.py

Ovaj problem se razlikuje od prethodnog po tome što sada ne računamo verovatnoću da čestica bude unutar intervala $[-1,1]$ u određenom trenutku, već procenjujemo verovatnoću da maksimum (supremum) putanje Brownovog kretanja dostigne ili pređe granicu Ktokom celog vremenskog intervala $[0, T]$. – Verovatnoća retkog događaja.

Ovaj put problem je postavljen tako da se uvek krećemo iz jedne početne tačke x_0. Iz te tačke se generiše $\boldsymbol{M}$ nezavisnih Brownovih putanja, pri čemu svaka putanja ima fiksnih $\boldsymbol{N}$ koraka u vremenskom intervalu $[0, T]$. Tokom simulacije, za svaku putanju proverava se da li je u bilo kom trenutku prešla zadati prag K. Na taj način procenjuje se verovatnoća događaja da maksimum putanje dostigne ili premaši Ku toku celog kretanja.

Drift je deterministički deo u dinamici procesa (dodaje "smer" kretanju). U ovom problemu se testira i bez drifta (standardni Brownov proces, $b(x) = 0$) i sa driftom (kod importance sampling-a), gde drift "gura" putanje ka pragu Kkako bi se retki događaj češće desio i smanjila varijansa procene.

U ovom primeru, skaliramo broj koraka simulacije N, kako bismo dobili bolju diskretizaciju vremena.

Paralelizacija je urađena slično kao u prethodnoj primeni.

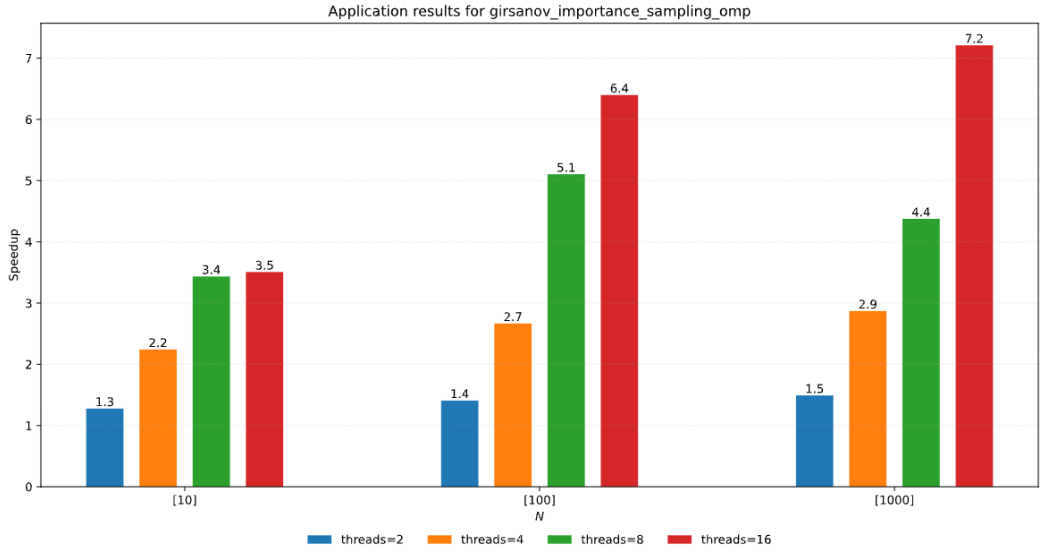
Drift u ovom problemu “gura” putanje prema pragu K, što povećava verovatnoću da se retki događaj (prelazak praga) dogodi ranije. Na taj način se dobijaju relevantnije putanje i smanjuje se varijansa procene, pa je potrebno manje simulacija da bi se postigla ista tačnost.

Kao smer za dalje istraživanje, može se razmotriti paralelizovanje simulacija **bez drifta**, ali sa znatno većim brojem putanja Mili većom dimenzijom problema. Cilj bi bio da paralelna verzija, zahvaljujući većem obimu simulacija, postigne sličnu tačnost kao sekvencijalna verzija sa driftom, i to u približno istom vremenu. Na taj način bi se ispitala stvarna korist paralelizacije u scenarijima kada drift nije korišćen

Grafici su prikazani sa I bez korišćenja drift-a, iako to ne utiče na vreme izvršavanja, budući da uvek izvršavamo fiksnu količinu posla.

4.2.1. Grafici dobijenih rezultata, BEZ DRIFT-A– variramo broj koraka simulacije kretanja

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj tačaka.



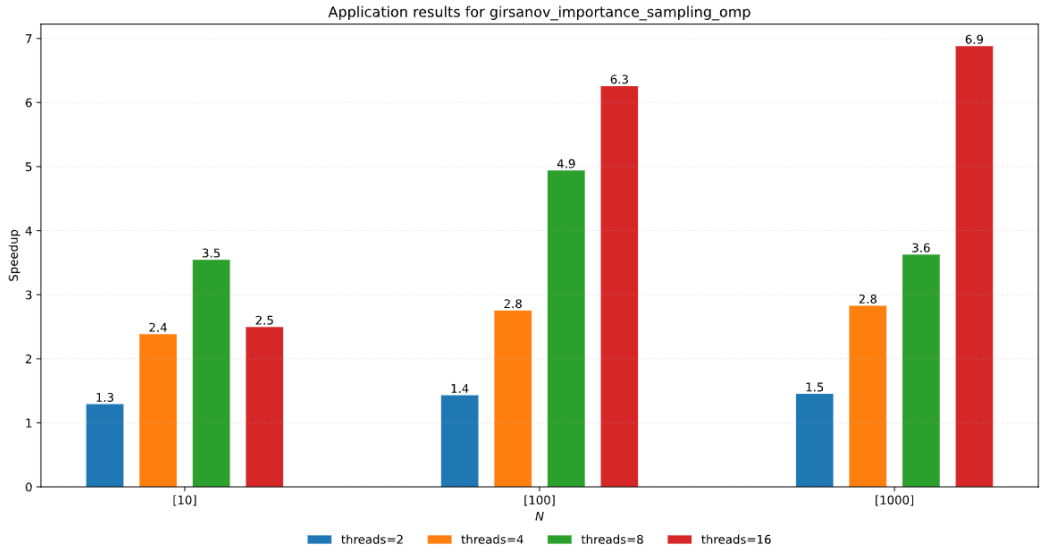
Grafik ubrzanja za različit broj niti i broja koraka za girsanov importance sampling problem bez drifta

Diskusija dobijenih rezultata

Paralelizacija je uspešno izvršena i svi rezultati su bili u dozvoljenom opsegu. Problem dobro skalira. Za male vrednosti N, značajno je manji doprinos paralelizacije zbog premale dimenzije problema.

4.2.3. Grafici dobijenih rezultata, SA DRIFT-OM– variramo broj koraka simulacije kretanja

Na y osi je predstavljeno ubrzanje, dok se na x osi nalazi N, broj tačaka.



Grafik ubrzanja za različit broj niti i broja koraka za girsanov importance sampling problem bez drifta

Diskusija dobijenih rezultata

Isto kao i bez drift-a.