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**](#_gjdgxs) **2**

[**1. Napomena**](#_whh48jyf2pmp) **4** 5

[**2. Problem 1 - Prosti brojevi**](#_di4i2gbygcxy) **6**

[2.1. Tekst problema](#_vknee325s76w) 6

[2.2. Delovi koje treba paralelizovati](#_kvdzxc5g27vo) 6

[2.2.1. Diskusija](#_bnshe78xgy5a) 6

[2.2.2. Način paralelizacije](#_nig1fa805twx) 6

[2.3. Rezultati](#_byj0jek7gbav) 6

[2.3.1. Logovi izvršavanja](#_luw590hk5nfx) 6

[2.3.2. Grafici ubrzanja](#_jnzbyagasp4q) 6

[**3. Problem 2 - Poasonova jednačina**](#_wwrdanvsf0t4) **7**

[3.1. Tekst problema](#_72nvtjtneby2) 7

[3.2. Delovi koje treba paralelizovati](#_7jva9t6gsg7r) 7

[3.2.1. Diskusija](#_b4xjgk5d7mpw) 7

[3.2.2. Način paralelizacije](#_864z1uogm8ng) 7

[3.3. Rezultati](#_mk3fn7q2dzdq) 7

[3.3.1. Logovi izvršavanja](#_tvyr7hgq1do7) 7

[3.3.2. Grafici ubrzanja](#_hh3mcjaxxqxr) 7

[**4. Problem 3 - Poasonova jednačina**](#_p8ceo6dmm3vm) **8**

[4.1. Tekst problema](#_kssgh9tdbpeu) 8

[4.2. Delovi koje treba paralelizovati](#_33fis6tnd9bq) 8

[4.2.1. Diskusija](#_vgyjlklb9vjp) 8

[4.2.2. Način paralelizacije](#_yhvkfqfwxglk) 8

[4.3. Rezultati](#_u9fbev38m432) 8

[4.3.1. Logovi izvršavanja](#_lf3aozhxdu5r) 8

[4.3.2. Grafici ubrzanja](#_r7eg38p1dect) 8

[**5. Problem 4 - Molekularna dinamika**](#_beyfevy14bc) **9**

[5.1. Tekst problema](#_31jambafqn56) 9

[5.2. Delovi koje treba paralelizovati](#_r9pvkhy5r4o) 9

[5.2.1. Diskusija](#_4xaetw4g17ph) 9

[5.2.2. Način paralelizacije](#_690xmhq4nhlc) 9

[5.3. Rezultati](#_8m5dxqqhtepg) 9

[5.3.1. Logovi izvršavanja](#_w2r8w0p6qupx) 9

[5.3.2. Grafici ubrzanja](#_792694xp5hul) 9

# Napomena

Navedeni rezultati izvršavanja se odnose na pokretanje na MacBook Pro M1.

# Problem 1 - Prosti brojevi

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

## Tekst problema

Paralelizovati program koji vrši određivanje ukupnog broja prostih brojeva u zadatom opsegu. Program se nalazi u datoteci **prime.c** u arhivi koja je priložena uz ovaj dokument. Prilikom paralelizacije nije dozvoljeno koristiti direktive za podelu posla (*worksharing* direktive), već je iteracije petlje koja se paralelizuje potrebno raspodeliti ručno. Obratiti pažnju na ispravno deklarisanje svih promenljivih prilikom paralelizacije. Program testirati sa parametrima koji su dati u datoteci **run**. [1, N]

## Delovi koje treba paralelizovati

### Diskusija

Mogućnost paralelizacije izvršenja uočena je u funkciji **prime\_number()**, koja pomoću naivnog algoritma računa broj prostih brojeva u opsegu od 2 do *n*, gde je *n* argument funkcije. Funkcija se sastoji od 2 ugnježdene petlje - spoljna prolazi kroz opseg brojeva, a unutrašnja proverava da li je trenutni broj spoljne petlje prost, i ako jeste, uvećava brojač prostih brojeva za 1.

### Način paralelizacije

Treba uočiti da se za svaku iteraciju spoljne petlje dužina iteracije unutrašnje poveća za jedan, odnosno za još jedan broj više čiji ostatak treba da se pojavi. Ovo čini trougaonu petlju, i zahteva cikličnu dekompoziciju problema između niti da bi se balansiralo opterećenje između njih.

Paralelizovana je spoljna petlja, tako što svaka nit počinje od broja pomerenog za svoj identifikator i iteracija se povećava za ukupan broj niti. Time je obezbeđena ciklična dekompozicija.

Za sam brojač prostih brojeva **total** korišćena je sabirajuća redukcija.

Iako bi se sam program značajno ubrzao ukoliko bi se umesto ovakvog naivnog algoritma koristio algoritam gde se prosti brojevi traže do korena zadate granice, umesto do cele granice, razlika između izvršavanja takve sekvencijalne implementacije sa jednom i više niti ne bi bila mnogo različita, odnosno ne bi se postiglo veće ubrzanje paralelne u odnosu na sekvencijalnu implementaciju. Štaviše, efekti paralelizacije bi postali značajno manji na zadatim parametrima, pošto bi se i program na jednoj niti izvršavao dosta brže. Zbog ovoga, ovakva izmena sekvencijalnog algoritma nije urađena.

## 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. Radi dobijanja tačnih rezultata, u funkciji **test** promenjen je način računanja vremena sa CPU vremena na wall vreme, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=0, lo=1, hi=131072, factor=2, num\_threads=1**

**1 0 0.000026**

**2 1 0.000000**

**4 2 0.000001**

**8 4 0.000000**

**16 6 0.000001**

**32 11 0.000000**

**64 18 0.000001**

**128 31 0.000002**

**256 54 0.000005**

**512 97 0.000019**

**1024 172 0.000063**

**2048 309 0.000220**

**4096 564 0.000862**

**8192 1028 0.002866**

**16384 1900 0.010483**

**32768 3512 0.038575**

**65536 6542 0.168822**

**131072 12251 0.654493**

**Izvršavanje za argumente 1 131072 2 i jednu nit**

**TEST: func=0, lo=1, hi=131072, factor=2, num\_threads=2**

**1 0 0.000029**

**2 1 0.000006**

**4 2 0.000010**

**8 4 0.000008**

**16 6 0.000017**

**32 11 0.000008**

**64 18 0.000009**

**128 31 0.000012**

**256 54 0.000013**

**512 97 0.000027**

**1024 172 0.000075**

**2048 309 0.000226**

**4096 564 0.000783**

**8192 1028 0.002962**

**16384 1900 0.010527**

**32768 3512 0.038307**

**65536 6542 0.167586**

**131072 12251 0.653688**

**Izvršavanje za argumente 1 131072 2 i dve niti**

**TEST: func=0, lo=1, hi=131072, factor=2, num\_threads=4**

**1 0 0.000058**

**2 1 0.000017**

**4 2 0.000016**

**8 4 0.000014**

**16 6 0.000015**

**32 11 0.000013**

**64 18 0.000017**

**128 31 0.000013**

**256 54 0.000013**

**512 97 0.000019**

**1024 172 0.000048**

**2048 309 0.000128**

**4096 564 0.000436**

**8192 1028 0.001465**

**16384 1900 0.005485**

**32768 3512 0.019466**

**65536 6542 0.085877**

**131072 12251 0.338080**

**Izvršavanje za argumente 1 131072 2 i četiri niti**

**TEST: func=0, lo=1, hi=131072, factor=2, num\_threads=8**

**1 0 0.000157**

**2 1 0.000078**

**4 2 0.000060**

**8 4 0.000110**

**16 6 0.000062**

**32 11 0.000041**

**64 18 0.000065**

**128 31 0.000043**

**256 54 0.000033**

**512 97 0.000062**

**1024 172 0.000065**

**2048 309 0.000145**

**4096 564 0.000319**

**8192 1028 0.000854**

**16384 1900 0.002868**

**32768 3512 0.010233**

**65536 6542 0.044305**

**131072 12251 0.176343**

**Izvršavanje za argumente 1 131072 2 i osam nit**

**TEST: func=0, lo=1, hi=131072, factor=2, num\_threads=16**

**1 0 0.000203**

**2 1 0.000088**

**4 2 0.000101**

**8 4 0.000068**

**16 6 0.000079**

**32 11 0.000119**

**64 18 0.000112**

**128 31 0.000098**

**256 54 0.000185**

**512 97 0.000110**

**1024 172 0.000135**

**2048 309 0.000127**

**4096 564 0.000260**

**8192 1028 0.000812**

**16384 1900 0.001552**

**32768 3512 0.005713**

**65536 6542 0.022959**

**131072 12251 0.088691**

**Izvršavanje za argumente 1 131072 2 i šestnaest niti**

**TEST: func=0, lo=5, hi=500000, factor=10, num\_threads=1**

**5 3 0.000006**

**50 15 0.000001**

**500 95 0.000018**

**5000 669 0.001120**

**50000 5133 0.096989**

**500000 41538 8.625681**

**Izvršavanje za argumente 5 500000 10 i jednu nit**

**TEST: func=0, lo=5, hi=500000, factor=10, num\_threads=2**

**5 3 0.000033**

**50 15 0.000008**

**500 95 0.000029**

**5000 669 0.001140**

**50000 5133 0.097226**

**500000 41538 8.843688**

**Izvršavanje za argumente 5 500000 10 i dve niti**

**TEST: func=0, lo=5, hi=500000, factor=10, num\_threads=4**

**5 3 0.000046**

**50 15 0.000016**

**500 95 0.000032**

**5000 669 0.000609**

**50000 5133 0.050411**

**500000 41538 4.442671**

**Izvršavanje za argumente 5 500000 10 i četiri niti**

**TEST: func=0, lo=5, hi=500000, factor=10, num\_threads=8**

**5 3 0.000156**

**50 15 0.000089**

**500 95 0.000082**

**5000 669 0.000415**

**50000 5133 0.025732**

**500000 41538 2.278442**

**Izvršavanje za argumente 5 500000 10 i osam nit**

**TEST: func=0, lo=5, hi=500000, factor=10, num\_threads=16**

**5 3 0.000271**

**50 15 0.000121**

**500 95 0.000161**

**5000 669 0.000369**

**50000 5133 0.013743**

**500000 41538 1.151832**

**Izvršavanje za argumente 5 500000 10 i šestnaest niti**

**TEST: func=0, lo=1, hi=65536, factor=4, num\_threads=1**

**1 0 0.000005**

**4 2 0.000001**

**16 6 0.000000**

**64 18 0.000001**

**256 54 0.000006**

**1024 172 0.000062**

**4096 564 0.000777**

**16384 1900 0.010573**

**65536 6542 0.167660**

**Izvršavanje za argumente 1 65536 4 i jednu nit**

**TEST: func=0, lo=1, hi=65536, factor=4, num\_threads=2**

**1 0 0.000038**

**4 2 0.000007**

**16 6 0.000009**

**64 18 0.000009**

**256 54 0.000014**

**1024 172 0.000069**

**4096 564 0.000785**

**16384 1900 0.010553**

**65536 6542 0.167709**

**Izvršavanje za argumente 1 65536 4 i dve niti**

**TEST: func=0, lo=1, hi=65536, factor=4, num\_threads=4**

**1 0 0.000045**

**4 2 0.000016**

**16 6 0.000045**

**64 18 0.000024**

**256 54 0.000030**

**1024 172 0.000067**

**4096 564 0.000447**

**16384 1900 0.005453**

**65536 6542 0.086229**

**Izvršavanje za argumente 1 65536 4 i četiri niti**

**TEST: func=0, lo=1, hi=65536, factor=4, num\_threads=8**

**1 0 0.000152**

**4 2 0.000059**

**16 6 0.000041**

**64 18 0.000041**

**256 54 0.000035**

**1024 172 0.000124**

**4096 564 0.000242**

**16384 1900 0.002779**

**65536 6542 0.044002**

**Izvršavanje za argumente 1 65536 4 i osam nit**

**TEST: func=0, lo=1, hi=65536, factor=4, num\_threads=16**

**1 0 0.000239**

**4 2 0.000114**

**16 6 0.000094**

**64 18 0.000092**

**256 54 0.000139**

**1024 172 0.000113**

**4096 564 0.000230**

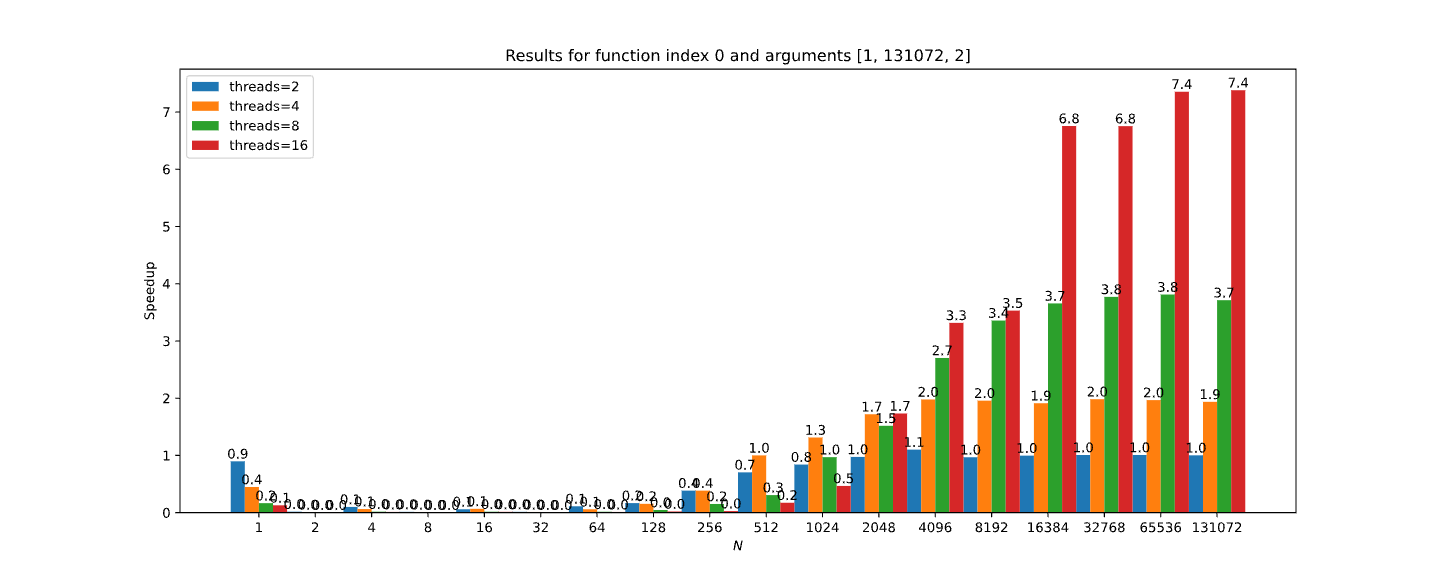
**16384 1900 0.001510**

**65536 6542 0.025112**

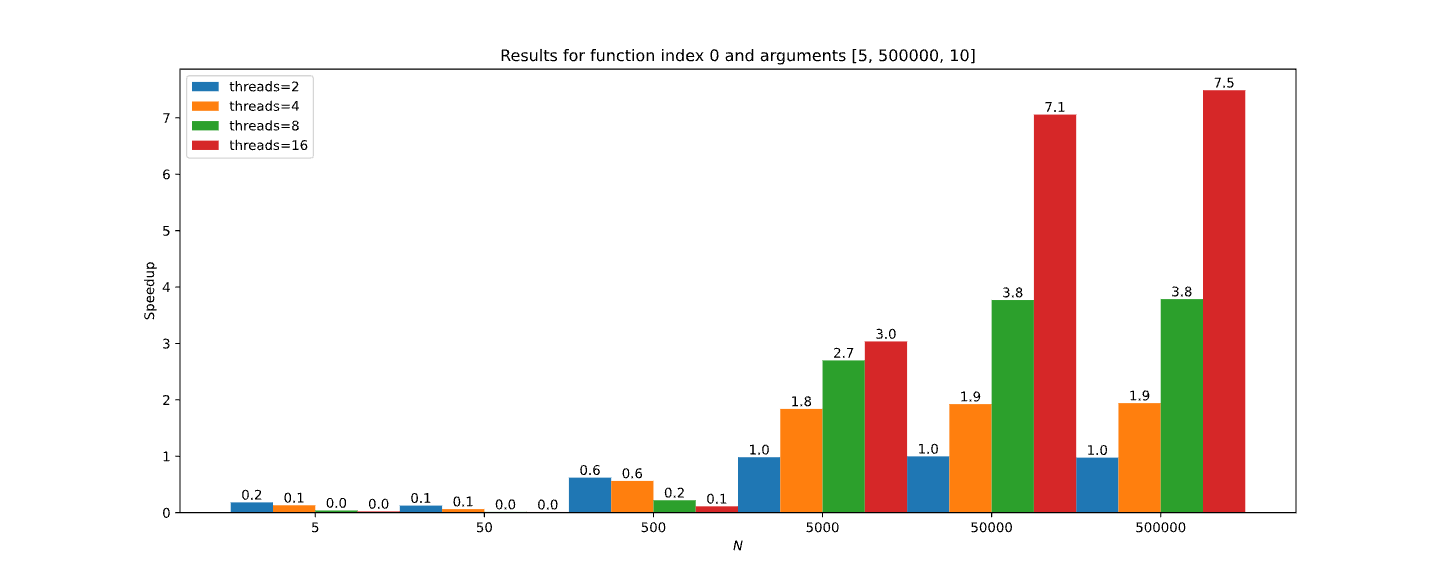
**Izvršavanje za argumente 1 65536 4 i šestnaest niti**

### Grafici ubrzanja

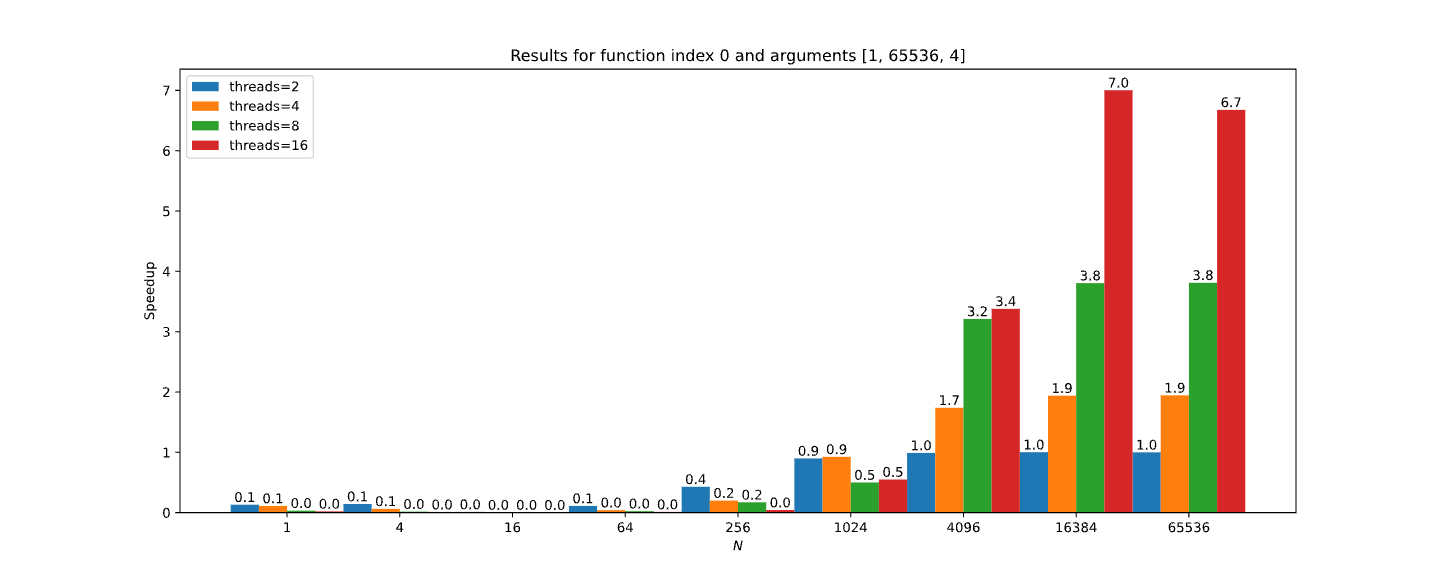
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 1, 131072, 2.**



**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 5, 5000000, 10.**



**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 1, 65536, 4.**

### Diskusija dobijenih rezultata

Paralelizacija je vidno uspešna za veće N. Dalja ubrzanja dostižna su pomenutim optimizacijama samog algoritma otkrivanja da li je broj prost, ali bi ubrzanje dobijeno paralelizacijom u odnosu na sekvencijalno izvršavanje bilo manje izraženo.

# Problem 2 - Prosti brojevi

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 2.

## Tekst problema

Prethodni program paralelizovati korišćenjem direktiva za podelu posla (*worksharing* direktive). Program testirati sa parametrima koji su dati u datoteci **run**. [1, N]

## Delovi koje treba paralelizovati

### Diskusija

Diskusija je ista kao i za prethodni problem.

### Način paralelizacije

Program je paralelizovan jednom **for** *worksharing* direktivom, sa *static* raspoređivanjem i *chunksize*-om 1. Odluka da se koristi *static* raspoređivanje dolazi od toga da je razlika u dužini iteracija ravnomerna, pa je takav način raspoređivanja idealan i ne donosi sa sobom znatne režijske troškove. Parametar *chunksize* je vrednosti 1 da bi se dobila ciklična dekompozicija problema, kao što je i predloženo u prethodnom problemu. Veća vrednost parametra *chunksize* bi mogla da dovede do neravnomerne raspodele posla, i generalno nije poboljšala performanse – empirijski je primećeno.

Kao i prethodnom problemu, koristi se redukcija sabiranjem za brojač **total**.

## 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. Radi dobijanja tačnih rezultata, u funkciji **test** promenjen je način računanja vremena sa CPU vremena na wall vreme, koristeći OpenMP rutinu **omp\_get\_wtime()**.

**TEST: func=1, lo=1, hi=131072, factor=2, num\_threads=1**

**1 0 0.000008**

**2 1 0.000001**

**4 2 0.000000**

**8 4 0.000001**

**16 6 0.000000**

**32 11 0.000001**

**64 18 0.000001**

**128 31 0.000002**

**256 54 0.000005**

**512 97 0.000015**

**1024 172 0.000053**

**2048 309 0.000187**

**4096 564 0.000693**

**8192 1028 0.002552**

**16384 1900 0.009225**

**32768 3512 0.033851**

**65536 6542 0.126282**

**131072 12251 0.473765**

**Izvršavanje za argumente 1 131072 2 i jednu nit**

**TEST: func=1, lo=1, hi=131072, factor=2, num\_threads=2**

**1 0 0.000092**

**2 1 0.000112**

**4 2 0.000040**

**8 4 0.000015**

**16 6 0.000029**

**32 11 0.000010**

**64 18 0.000011**

**128 31 0.000024**

**256 54 0.000030**

**512 97 0.000036**

**1024 172 0.000104**

**2048 309 0.000209**

**4096 564 0.000708**

**8192 1028 0.002520**

**16384 1900 0.009180**

**32768 3512 0.033909**

**65536 6542 0.126419**

**131072 12251 0.474634**

**Izvršavanje za argumente 1 131072 2 i dve niti**

**TEST: func=1, lo=1, hi=131072, factor=2, num\_threads=4**

**1 0 0.000081**

**2 1 0.000035**

**4 2 0.000040**

**8 4 0.000040**

**16 6 0.000025**

**32 11 0.000033**

**64 18 0.000029**

**128 31 0.000023**

**256 54 0.000027**

**512 97 0.000040**

**1024 172 0.000073**

**2048 309 0.000137**

**4096 564 0.000378**

**8192 1028 0.001352**

**16384 1900 0.004768**

**32768 3512 0.017573**

**65536 6542 0.065280**

**131072 12251 0.246356**

**Izvršavanje za argumente 1 131072 2 i četiri niti**

**TEST: func=1, lo=1, hi=131072, factor=2, num\_threads=8**

**1 0 0.000221**

**2 1 0.000117**

**4 2 0.000098**

**8 4 0.000074**

**16 6 0.000119**

**32 11 0.000071**

**64 18 0.000115**

**128 31 0.000051**

**256 54 0.000097**

**512 97 0.000110**

**1024 172 0.000089**

**2048 309 0.000188**

**4096 564 0.000269**

**8192 1028 0.000758**

**16384 1900 0.002613**

**32768 3512 0.009215**

**65536 6542 0.033452**

**131072 12251 0.127555**

**Izvršavanje za argumente 1 131072 2 i osam nit**

**TEST: func=1, lo=1, hi=131072, factor=2, num\_threads=16**

**1 0 0.000309**

**2 1 0.000161**

**4 2 0.000150**

**8 4 0.000150**

**16 6 0.000146**

**32 11 0.000224**

**64 18 0.000166**

**128 31 0.000155**

**256 54 0.000203**

**512 97 0.000233**

**1024 172 0.000190**

**2048 309 0.000293**

**4096 564 0.000394**

**8192 1028 0.000567**

**16384 1900 0.001492**

**32768 3512 0.008870**

**65536 6542 0.017366**

**131072 12251 0.064492**

**Izvršavanje za argumente 1 131072 2 i šestnaest niti**

**TEST: func=1, lo=5, hi=500000, factor=10, num\_threads=1**

**5 3 0.000006**

**50 15 0.000001**

**500 95 0.000016**

**5000 669 0.000991**

**50000 5133 0.076023**

**500000 41538 6.242606**

**Izvršavanje za argumente 5 500000 10 i jednu nit**

**TEST: func=1, lo=5, hi=500000, factor=10, num\_threads=2**

**5 3 0.000118**

**50 15 0.000017**

**500 95 0.000032**

**5000 669 0.001062**

**50000 5133 0.076004**

**500000 41538 6.167683**

**Izvršavanje za argumente 5 500000 10 i dve niti**

**TEST: func=1, lo=5, hi=500000, factor=10, num\_threads=4**

**5 3 0.000065**

**50 15 0.000026**

**500 95 0.000045**

**5000 669 0.000548**

**50000 5133 0.039014**

**500000 41538 3.179488**

**Izvršavanje za argumente 5 500000 10 i četiri niti**

**TEST: func=1, lo=5, hi=500000, factor=10, num\_threads=8**

**5 3 0.000172**

**50 15 0.000061**

**500 95 0.000091**

**5000 669 0.000360**

**50000 5133 0.020069**

**500000 41538 1.639120**

**Izvršavanje za argumente 5 500000 10 i osam nit**

**TEST: func=1, lo=5, hi=500000, factor=10, num\_threads=16**

**5 3 0.000418**

**50 15 0.000245**

**500 95 0.000177**

**5000 669 0.000338**

**50000 5133 0.012121**

**500000 41538 0.827429**

**Izvršavanje za argumente 5 500000 10 i šestnaest niti**

**TEST: func=1, lo=1, hi=65536, factor=4, num\_threads=1**

**1 0 0.000005**

**4 2 0.000000**

**16 6 0.000000**

**64 18 0.000001**

**256 54 0.000004**

**1024 172 0.000053**

**4096 564 0.000691**

**16384 1900 0.009283**

**65536 6542 0.126418**

**Izvršavanje za argumente 1 65536 4 i jednu nit**

**TEST: func=1, lo=1, hi=65536, factor=4, num\_threads=2**

**1 0 0.000043**

**4 2 0.000016**

**16 6 0.000016**

**64 18 0.000014**

**256 54 0.000018**

**1024 172 0.000067**

**4096 564 0.000699**

**16384 1900 0.009331**

**65536 6542 0.126510**

**Izvršavanje za argumente 1 65536 4 i dve niti**

**TEST: func=1, lo=1, hi=65536, factor=4, num\_threads=4**

**1 0 0.000064**

**4 2 0.000040**

**16 6 0.000027**

**64 18 0.000030**

**256 54 0.000031**

**1024 172 0.000074**

**4096 564 0.000400**

**16384 1900 0.004847**

**65536 6542 0.065178**

**Izvršavanje za argumente 1 65536 4 i četiri niti**

**TEST: func=1, lo=1, hi=65536, factor=4, num\_threads=8**

**1 0 0.000214**

**4 2 0.000079**

**16 6 0.000039**

**64 18 0.000083**

**256 54 0.000054**

**1024 172 0.000143**

**4096 564 0.000276**

**16384 1900 0.002504**

**65536 6542 0.033660**

**Izvršavanje za argumente 1 65536 4 i osam nit**

**TEST: func=1, lo=1, hi=65536, factor=4, num\_threads=16**

**1 0 0.000281**

**4 2 0.000118**

**16 6 0.000157**

**64 18 0.000130**

**256 54 0.000128**

**1024 172 0.000149**

**4096 564 0.000262**

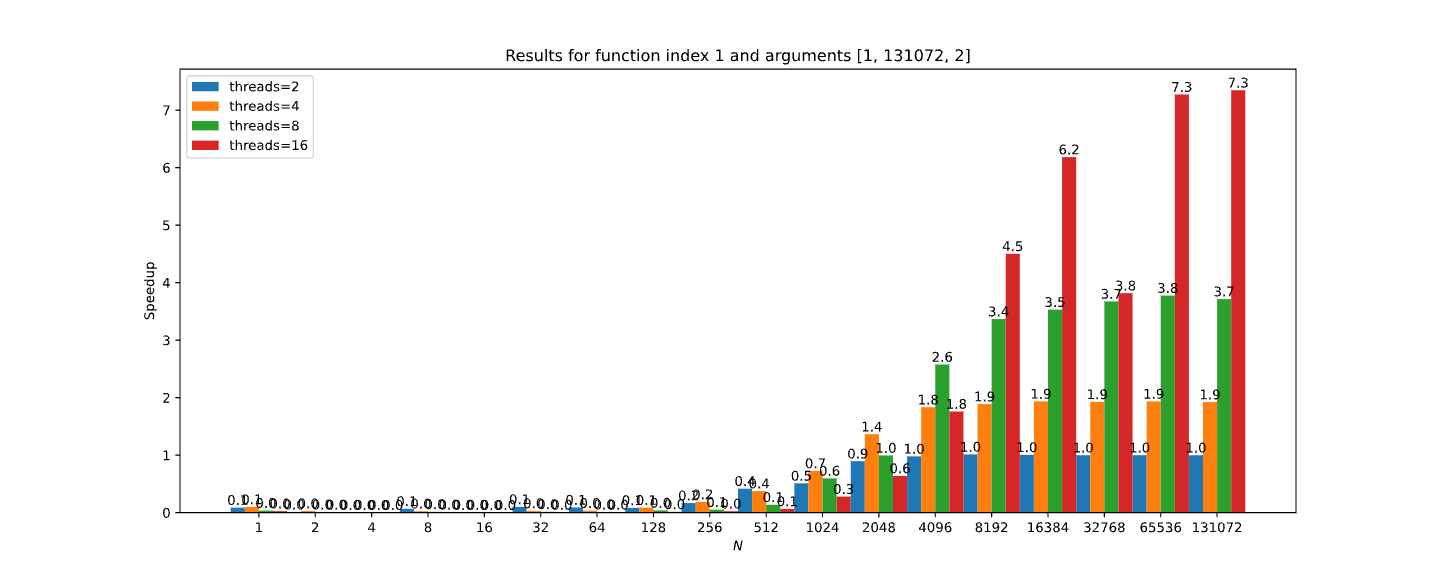
**16384 1900 0.001721**

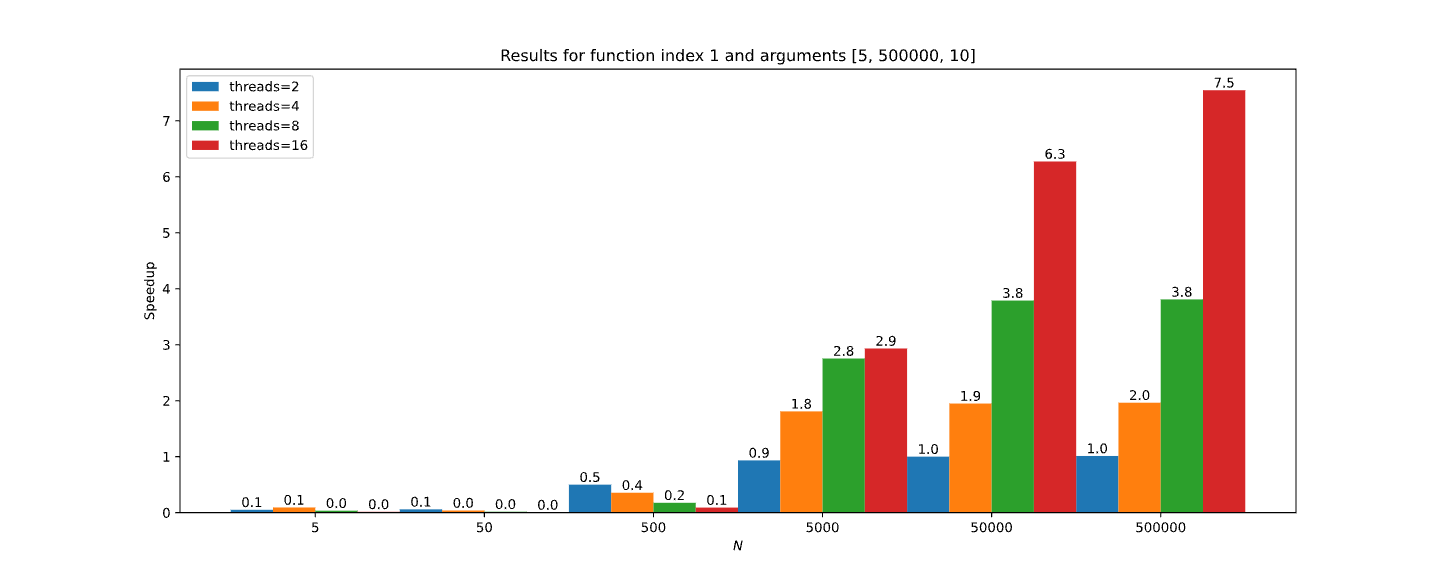
**65536 6542 0.017738**

**Izvršavanje za argumente 1 65536 4 i šestnaest niti**

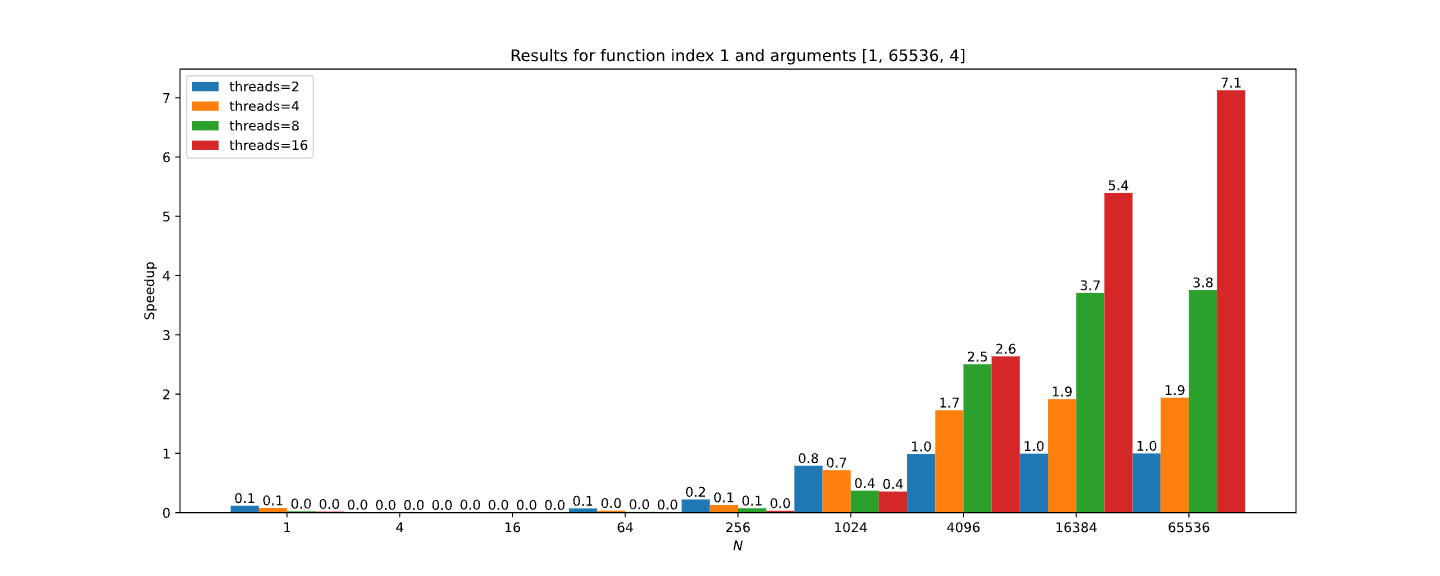
### Grafici ubrzanja

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



**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 1, 131072, 2.** ****

**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 5, 5000000, 10.**



**Grafik zavisnosti ubrzanja programa od granice do koje se traže prosti brojevi, podeljenu po nitima, za argumente 1, 65536, 4.**

### Diskusija dobijenih rezultata

Paralelizacija je vidno uspešna za veće N. Rezultati su slični kao iz prethodnog problema. Dalja ubrzanja dostižna su pomenutim optimizacijama samog algoritma otkrivanja da li je broj prost, ali bi ubrzanje dobijeno paralelizacijom u odnosu na sekvencijalno izvršavanje bilo manje izraženo.

# Problem 3 - Poasonova jednačina

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 3.

## Tekst problema

Paralelizovati program koji vrši izračunavanje 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 **feyman.c** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **run**. [1, N]

## Delovi koje treba paralelizovati

### Diskusija

Mogućnost paralelizacije izvršenja uočena je u funkciji **feynman()**, koja implementira sam *Feynman-Kac* algoritam. Specifičnost ovog algoritma je nasumično generisanje putanja tačaka, za šta se koristi generator pseudoslučajnih brojeva po uniformnoj raspodeli u funkciji **r8\_uniform\_01()**. 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

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()**.

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

**1000 0.021717 4.131337**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

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

**1000 0.021614 2.082415**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

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

**1000 0.021846 1.817548**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

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

**1000 0.021634 1.022370**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

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

**1000 0.022469 0.604707**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

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

**5000 0.021273 20.522961**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

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

**5000 0.021125 10.457520**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

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

**5000 0.021317 8.584322**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

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

**5000 0.020931 4.957851**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

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

**5000 0.020941 3.212632**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

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

**10000 0.021100 40.813259**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

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

**10000 0.020984 20.798146**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

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

**10000 0.021043 17.112431**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

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

**10000 0.021000 9.861268**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

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

**10000 0.020995 5.661726**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

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

**20000 0.021027 81.066985**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

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

**20000 0.020978 41.448141**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

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

**20000 0.021046 34.250091**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

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

**20000 0.020947 19.690891**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

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

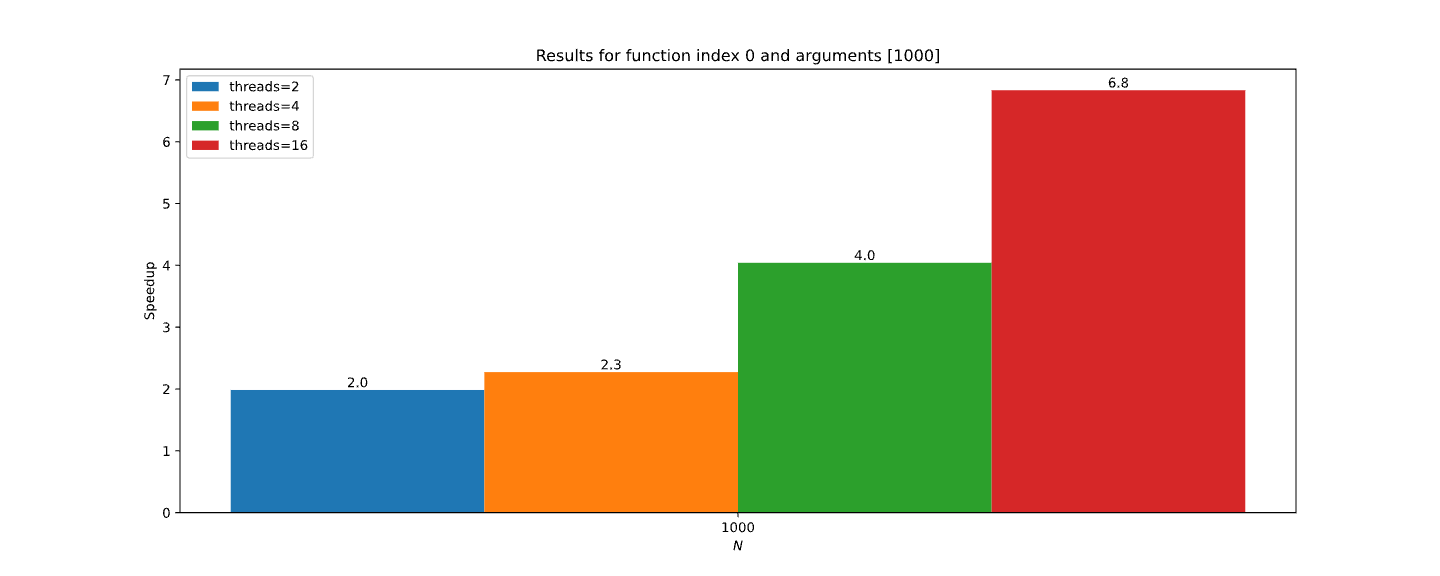
**20000 0.020964 11.411134**

**TEST END**

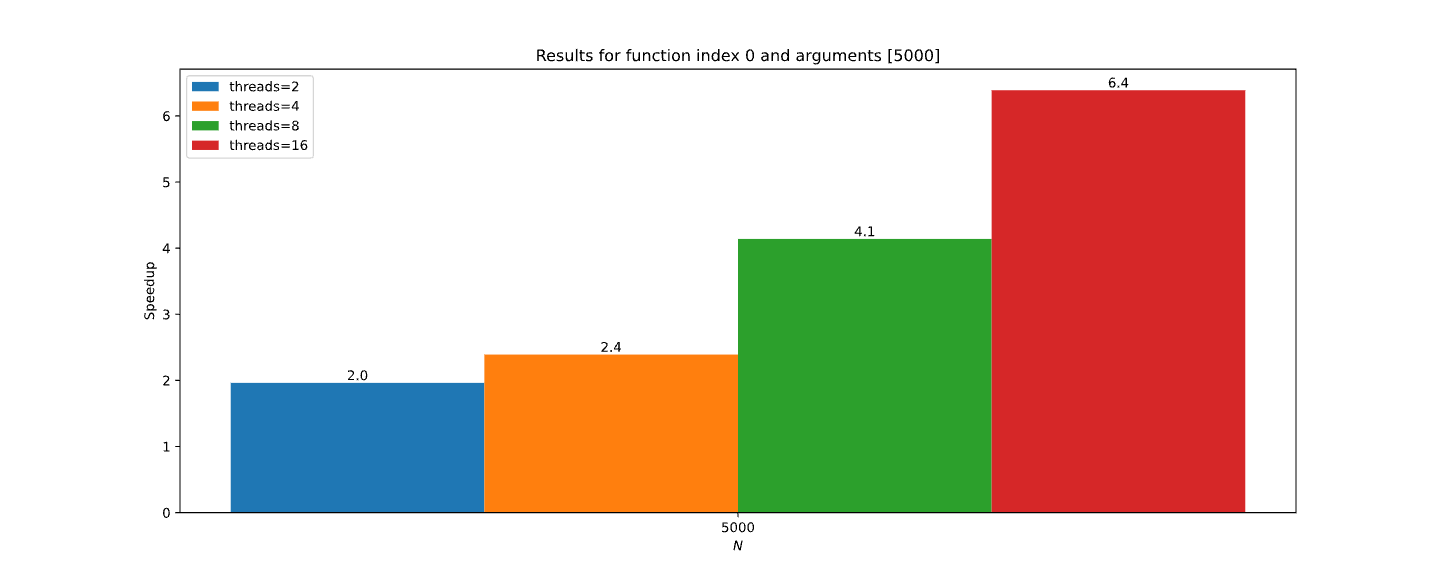
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja

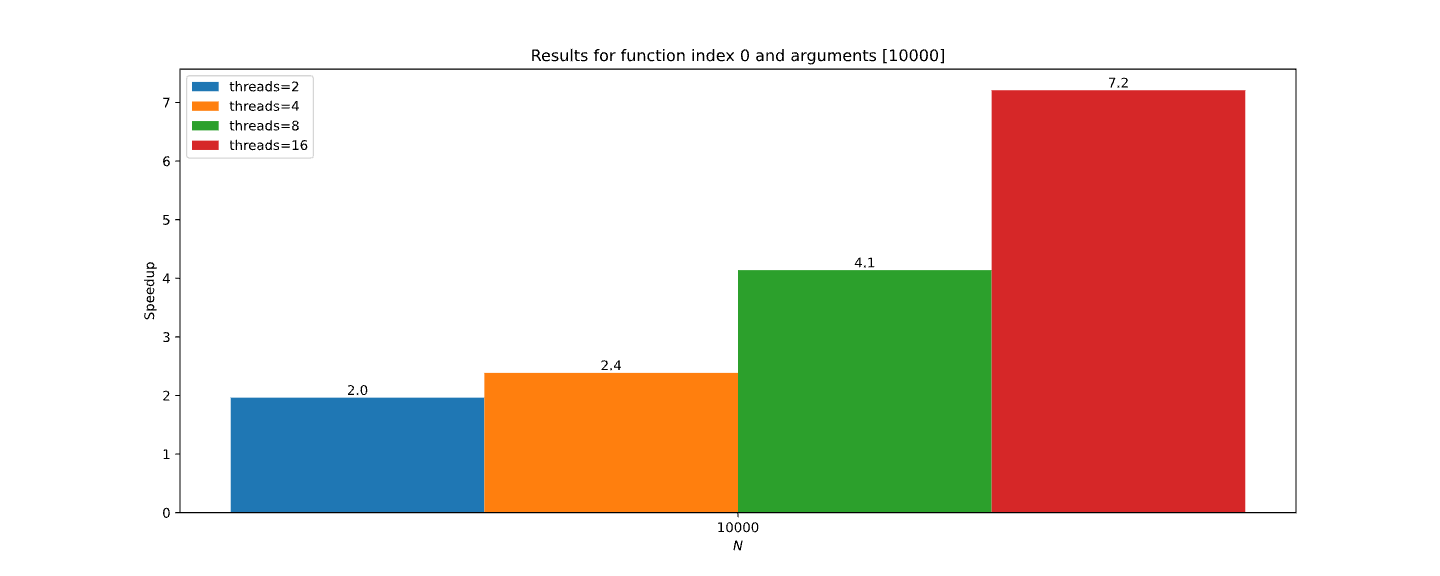
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.

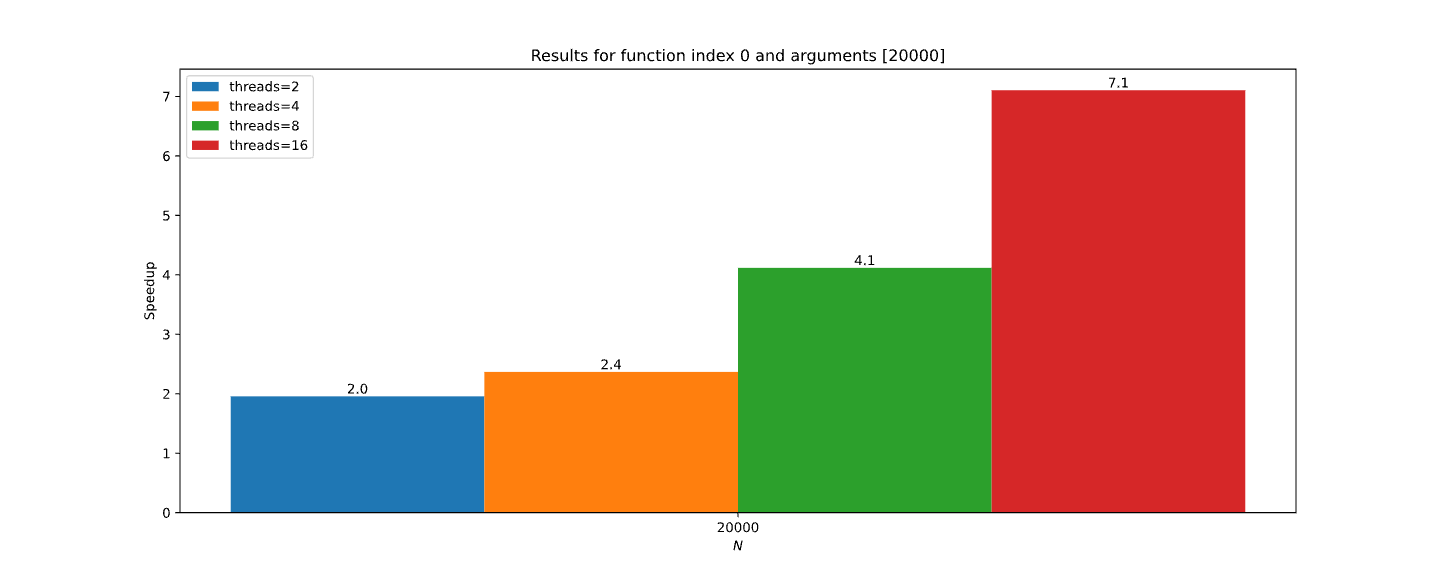


**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**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 može da ukazuje na to da se problem može dalje skalirati.

# Problem 4 - 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. Program testirati sa parametrima koji su dati u datoteci **run**. [1, N]

## Delovi koje treba paralelizovati

### Diskusija

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

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. 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.

## 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()**.

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

**1000 0.021712 4.024811**

**TEST END**

**Izvršavanje za argumente 1000 i jednu nit**

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

**1000 0.021669 2.086967**

**TEST END**

**Izvršavanje za argumente 1000 i dve niti**

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

**1000 0.021484 1.087751**

**TEST END**

**Izvršavanje za argumente 1000 i četiri niti**

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

**1000 0.022419 0.668714**

**TEST END**

**Izvršavanje za argumente 1000 i osam niti**

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

**1000 0.022461 0.737575**

**TEST END**

**Izvršavanje za argumente 1000 i šestnaest niti**

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

**5000 0.021272 20.489973**

**TEST END**

**Izvršavanje za argumente 5000 i jednu nit**

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

**5000 0.021018 10.478945**

**TEST END**

**Izvršavanje za argumente 5000 i dve niti**

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

**5000 0.021126 5.449140**

**TEST END**

**Izvršavanje za argumente 5000 i četiri niti**

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

**5000 0.021035 3.304991**

**TEST END**

**Izvršavanje za argumente 5000 i osam niti**

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

**5000 0.021179 3.570575**

**TEST END**

**Izvršavanje za argumente 5000 i šestnaest niti**

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

**10000 0.021099 40.358381**

**TEST END**

**Izvršavanje za argumente 10000 i jednu nit**

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

**10000 0.020766 20.964848**

**TEST END**

**Izvršavanje za argumente 10000 i dve niti**

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

**10000 0.021035 11.065931**

**TEST END**

**Izvršavanje za argumente 10000 i četiri niti**

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

**10000 0.020970 6.904453**

**TEST END**

**Izvršavanje za argumente 10000 i osam niti**

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

**10000 0.020933 6.965258**

**TEST END**

**Izvršavanje za argumente 10000 i šestnaest niti**

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

**20000 0.021027 80.615823**

**TEST END**

**Izvršavanje za argumente 20000 i jednu nit**

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

**20000 0.020982 41.855304**

**TEST END**

**Izvršavanje za argumente 20000 i dve niti**

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

**20000 0.020967 21.833881**

**TEST END**

**Izvršavanje za argumente 20000 i četiri niti**

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

**20000 0.020940 13.855232**

**TEST END**

**Izvršavanje za argumente 20000 i osam niti**

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

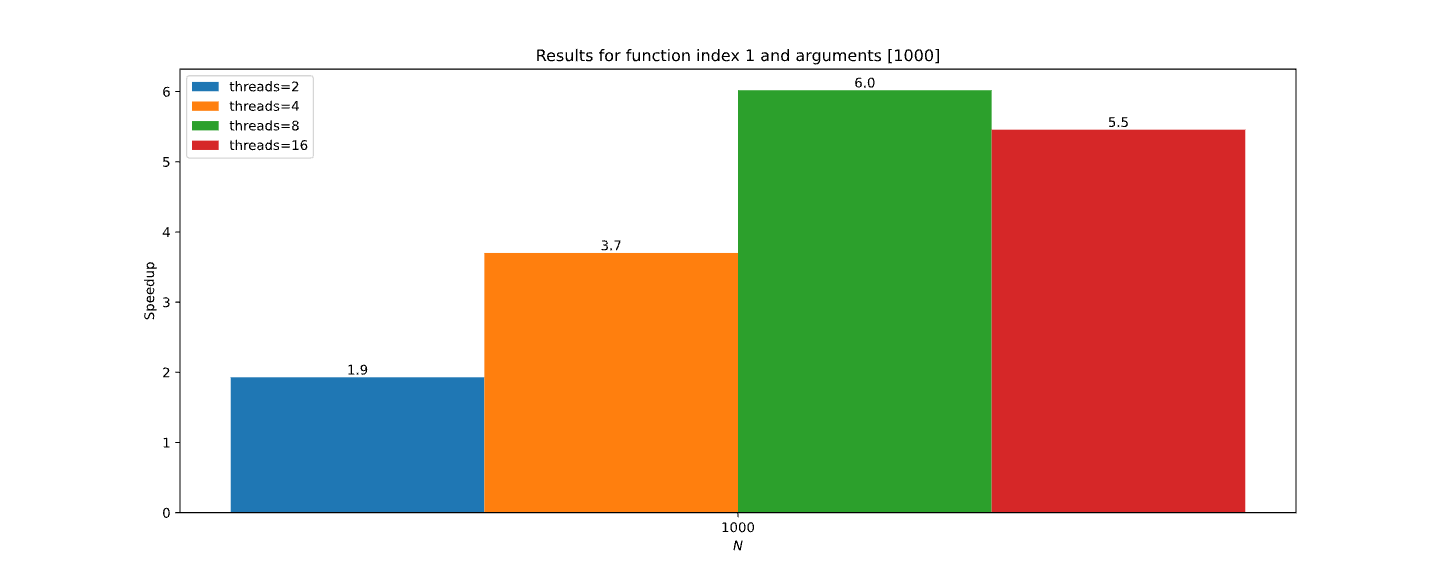
**20000 0.020928 14.158208**

**TEST END**

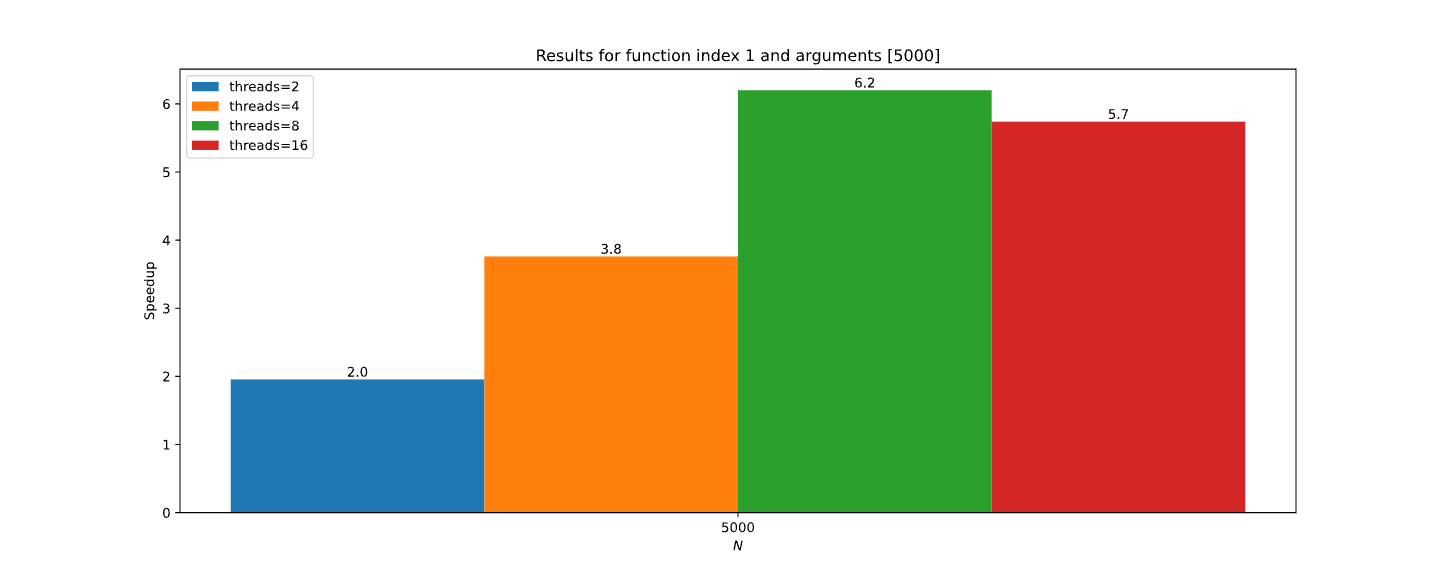
**Izvršavanje za argumente 20000 i šestnaest niti**

### Grafici ubrzanja

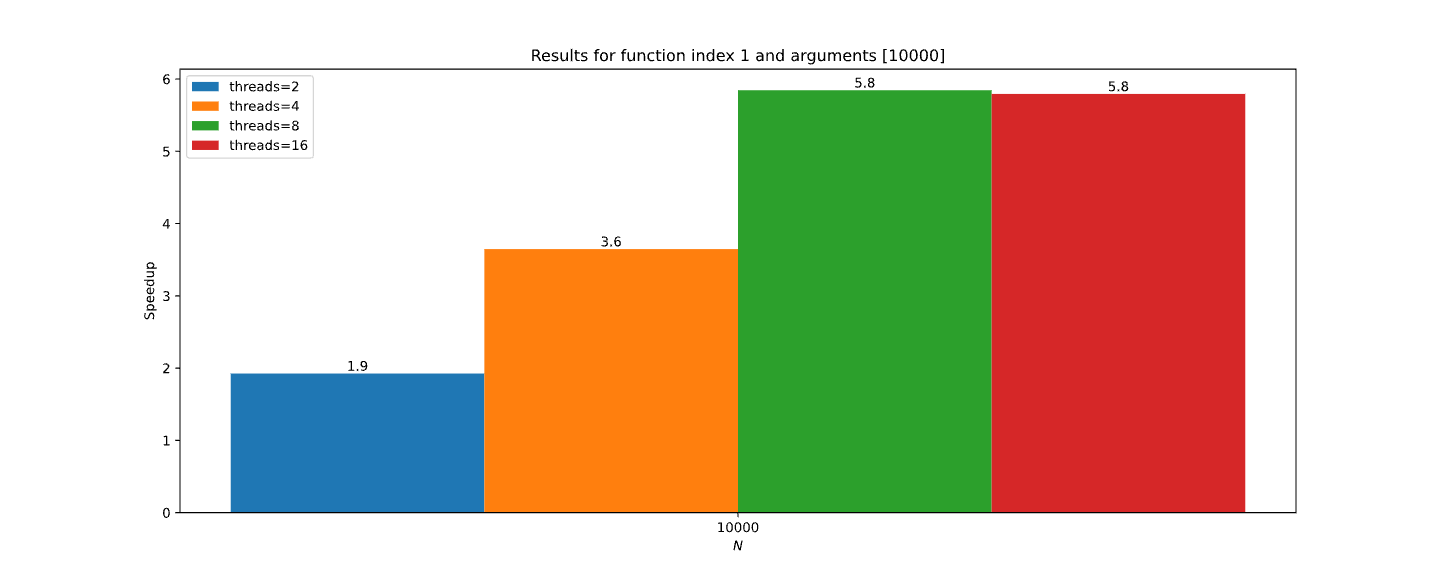
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



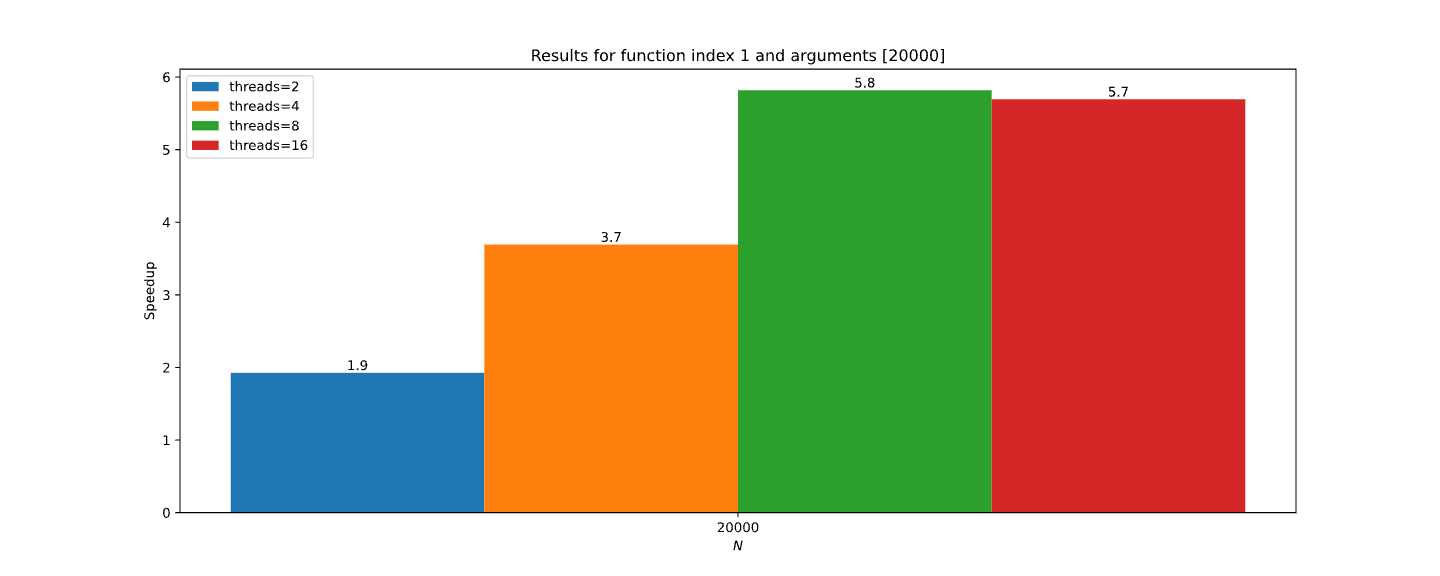
**Grafik ubrzanja za različit broj niti, sa argumentom 1000.**



**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.

### 

# Problem 5 - Molekularna dinamika

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 5.

## Tekst problema

Paralelizovati jednostavan program koji se bavi molekularnom dinamikom. Kod predstavlja simulaciju molekularne dinamike argonovog atoma u ograničenom prozoru (prostoru) sa periodičnim graničnim uslovima. Atomi se inicijalno nalaze raspoređeni u pravilnu mrežu, a zatim se tokom simulacije dešavaju interakcije između njih. U svakom koraku simulacije u glavnoj petlji se dešava sledeće:

* Čestice (atomi) se pomeraju zavisno od njihovih brzina i brzine se parcijalno ažuriraju u pozivu funkcije **domove**.
* Sile koje se primenjuju na nove pozicije čestica se izračunavaju; takođe, akumuliraju se prosečna kinetička energija (*virial*) i potencijalna energija u pozivu funkcije **forces**.
* Sile se skaliraju, završava ažuriranje brzine i izračunavanje kinetičke energije u pozivu funkcije **mkekin**.
* Prosečna brzina čestice se računa i skaliraju temperature u pozivu funkcije **velavg**.
* Pune potencijalne i prosečne kinetičke energije (virial) se računaju i ispisuju u funkciji prnout.

Program se nalazi u datoteci direktorijumu **MolDyn** u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke **main.c** i **forces.c**, jer se u njima provodi najviše vremena. Analizirati dati kod i obratiti pažnju na redukcione promenljive unutar datoteke **forces.c**. Ukoliko je potrebno međusobno isključenje prilikom paralelizacije programa, koristiti kritične sekcije ili atomske operacije. [1, N]

## Delovi koje treba paralelizovati

### Diskusija

Razmatrana je paralelizacija funkcije **forces()**, koja računaju promene kinetičke i potencijalne energije u molekulima. Funkcija se sastoji od dve ugnježdene for petlje, koje izvršavaju floating-point operacije i rezultate (promene sila) upisuju u niz. Kao bočni efekat, računaju ukupnu kinetičku i potencijalnu energiju u globalnim promenljivama **vir** i **epot**, respektivno.

### Način paralelizacije

Paralelizacija je izvršena **for** direktivom sa **collapse(2)** opcijom. Slično kao u trećem zadatku, neka izračunavanja su morala da budu pomerena u unutrašnju petlju, ali se ne očekuje da je to izazvalo veće usporenje.

Sinhronizacija je vršena nad nizom **f** tako što je svaka izmena tog niza obuhvaćena pod **atomic** direktivom. Takođe je probana sinhronizacija sa jednom **critical** direktivom umesto više **atomic** direktiva, ali je zaključeno da **atomic** direktiva ovde daje bolje performanse. Bolja sinhronizacija je potencijalno mogla biti postignuta sinhronizacijom nad svakim članom niza, umesto nizom u celosti, korišćenjem mehanizma brava. – **moguće unapređenje**.

Vršena je sabirajuća redukcija nad pomenutim promenljivama **vir** i **epot**.

## 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()**. Delovi gde je rađen ispis na standardni izlaz nisu računati u ukupno vreme izvršavanja.

**TEST: num\_threads=1**

**1 14551.3619 -93836.0953 -79284.7335 0.7186 -6.1148 0.1953 46.3**

**2 14340.7480 -93654.4811 -79313.7331 0.7082 -6.0186 0.1939 45.6**

**3 13968.2189 -93306.7546 -79338.5357 0.6898 -5.8516 0.1914 44.4**

**4 13404.4586 -92759.7121 -79355.2536 0.6620 -5.6090 0.1876 42.6**

**5 12619.1758 -91985.3542 -79366.1784 0.6232 -5.2880 0.1821 39.7**

**6 11599.4183 -90970.4371 -79371.0188 0.5729 -4.8918 0.1748 35.8**

**7 10379.3400 -89747.3564 -79368.0164 0.5126 -4.4367 0.1655 30.7**

**8 9066.7213 -88424.5967 -79357.8754 0.4478 -3.9592 0.1546 24.7**

**9 7831.4466 -87177.8677 -79346.4211 0.3868 -3.5119 0.1434 18.7**

**10 14619.4170 -86181.5919 -71562.1749 0.7220 -2.8265 0.1336 14.1**

**11 13529.0454 -85087.0949 -71558.0496 0.6682 -2.4076 0.1875 42.0**

**12 12687.0299 -84243.7642 -71556.7343 0.6266 -2.0726 0.1812 38.5**

**13 12082.9134 -83640.4681 -71557.5547 0.5967 -1.8201 0.1766 36.2**

**14 11674.2473 -83233.7353 -71559.4879 0.5765 -1.6380 0.1734 34.7**

**15 11405.1707 -82966.3254 -71561.1547 0.5633 -1.5094 0.1714 33.6**

**16 11223.2084 -82785.5933 -71562.3850 0.5543 -1.4177 0.1701 33.0**

**17 11089.5877 -82652.2025 -71562.6148 0.5477 -1.3495 0.1693 32.9**

**18 10982.3523 -82544.7149 -71562.3626 0.5424 -1.2965 0.1687 32.5**

**19 10893.9581 -82455.5630 -71561.6050 0.5380 -1.2544 0.1682 32.2**

**20 10825.0423 -82385.8646 -71560.8222 0.5346 -1.2219 0.1679 32.2**

**20 5.714868**

**Izvršavanje za jednu nit**

**TEST: num\_threads=2**

**1 14551.3619 -93836.0953 -79284.7335 0.7186 -6.1148 0.1953 46.3**

**2 14340.7480 -93654.4811 -79313.7331 0.7082 -6.0186 0.1939 45.6**

**3 13968.2189 -93306.7546 -79338.5357 0.6898 -5.8516 0.1914 44.4**

**4 13404.4586 -92759.7121 -79355.2536 0.6620 -5.6090 0.1876 42.6**

**5 12619.1758 -91985.3542 -79366.1784 0.6232 -5.2880 0.1821 39.7**

**6 11599.4183 -90970.4371 -79371.0188 0.5729 -4.8918 0.1748 35.8**

**7 10379.3400 -89747.3564 -79368.0164 0.5126 -4.4367 0.1655 30.7**

**8 9066.7213 -88424.5967 -79357.8754 0.4478 -3.9592 0.1546 24.7**

**9 7831.4466 -87177.8677 -79346.4211 0.3868 -3.5119 0.1434 18.7**

**10 14619.4170 -86181.5919 -71562.1749 0.7220 -2.8265 0.1336 14.1**

**11 13529.0454 -85087.0949 -71558.0496 0.6682 -2.4076 0.1875 42.0**

**12 12687.0299 -84243.7642 -71556.7343 0.6266 -2.0726 0.1812 38.5**

**13 12082.9134 -83640.4681 -71557.5547 0.5967 -1.8201 0.1766 36.2**

**14 11674.2473 -83233.7353 -71559.4879 0.5765 -1.6380 0.1734 34.7**

**15 11405.1707 -82966.3254 -71561.1547 0.5633 -1.5094 0.1714 33.6**

**16 11223.2084 -82785.5933 -71562.3850 0.5543 -1.4177 0.1701 33.0**

**17 11089.5877 -82652.2025 -71562.6148 0.5477 -1.3495 0.1693 32.9**

**18 10982.3523 -82544.7149 -71562.3626 0.5424 -1.2965 0.1687 32.5**

**19 10893.9581 -82455.5630 -71561.6050 0.5380 -1.2544 0.1682 32.2**

**20 10825.0423 -82385.8646 -71560.8222 0.5346 -1.2219 0.1679 32.2**

**20 2.990249**

**Izvršavanje za dve niti**

**TEST: num\_threads=4**

**1 14551.3619 -93836.0953 -79284.7335 0.7186 -6.1148 0.1953 46.3**

**2 14340.7480 -93654.4811 -79313.7331 0.7082 -6.0186 0.1939 45.6**

**3 13968.2189 -93306.7546 -79338.5357 0.6898 -5.8516 0.1914 44.4**

**4 13404.4586 -92759.7121 -79355.2536 0.6620 -5.6090 0.1876 42.6**

**5 12619.1758 -91985.3542 -79366.1784 0.6232 -5.2880 0.1821 39.7**

**6 11599.4183 -90970.4371 -79371.0188 0.5729 -4.8918 0.1748 35.8**

**7 10379.3400 -89747.3564 -79368.0164 0.5126 -4.4367 0.1655 30.7**

**8 9066.7213 -88424.5967 -79357.8754 0.4478 -3.9592 0.1546 24.7**

**9 7831.4466 -87177.8677 -79346.4211 0.3868 -3.5119 0.1434 18.7**

**10 14619.4170 -86181.5919 -71562.1749 0.7220 -2.8265 0.1336 14.1**

**11 13529.0454 -85087.0949 -71558.0496 0.6682 -2.4076 0.1875 42.0**

**12 12687.0299 -84243.7642 -71556.7343 0.6266 -2.0726 0.1812 38.5**

**13 12082.9134 -83640.4681 -71557.5547 0.5967 -1.8201 0.1766 36.2**

**14 11674.2473 -83233.7353 -71559.4879 0.5765 -1.6380 0.1734 34.7**

**15 11405.1707 -82966.3254 -71561.1547 0.5633 -1.5094 0.1714 33.6**

**16 11223.2084 -82785.5933 -71562.3850 0.5543 -1.4177 0.1701 33.0**

**17 11089.5877 -82652.2025 -71562.6148 0.5477 -1.3495 0.1693 32.9**

**18 10982.3523 -82544.7149 -71562.3626 0.5424 -1.2965 0.1687 32.5**

**19 10893.9581 -82455.5630 -71561.6050 0.5380 -1.2544 0.1682 32.2**

**20 10825.0423 -82385.8646 -71560.8222 0.5346 -1.2219 0.1679 32.2**

**20 1.550238**

**Izvršavanje za četiri niti**

**TEST: num\_threads=8**

**1 14551.3619 -93836.0953 -79284.7335 0.7186 -6.1148 0.1953 46.3**

**2 14340.7480 -93654.4811 -79313.7331 0.7082 -6.0186 0.1939 45.6**

**3 13968.2189 -93306.7546 -79338.5357 0.6898 -5.8516 0.1914 44.4**

**4 13404.4586 -92759.7121 -79355.2536 0.6620 -5.6090 0.1876 42.6**

**5 12619.1758 -91985.3542 -79366.1784 0.6232 -5.2880 0.1821 39.7**

**6 11599.4183 -90970.4371 -79371.0188 0.5729 -4.8918 0.1748 35.8**

**7 10379.3400 -89747.3564 -79368.0164 0.5126 -4.4367 0.1655 30.7**

**8 9066.7213 -88424.5967 -79357.8754 0.4478 -3.9592 0.1546 24.7**

**9 7831.4466 -87177.8677 -79346.4211 0.3868 -3.5119 0.1434 18.7**

**10 14619.4170 -86181.5919 -71562.1749 0.7220 -2.8265 0.1336 14.1**

**11 13529.0454 -85087.0949 -71558.0496 0.6682 -2.4076 0.1875 42.0**

**12 12687.0299 -84243.7642 -71556.7343 0.6266 -2.0726 0.1812 38.5**

**13 12082.9134 -83640.4681 -71557.5547 0.5967 -1.8201 0.1766 36.2**

**14 11674.2473 -83233.7353 -71559.4879 0.5765 -1.6380 0.1734 34.7**

**15 11405.1707 -82966.3254 -71561.1547 0.5633 -1.5094 0.1714 33.6**

**16 11223.2084 -82785.5933 -71562.3850 0.5543 -1.4177 0.1701 33.0**

**17 11089.5877 -82652.2025 -71562.6148 0.5477 -1.3495 0.1693 32.9**

**18 10982.3523 -82544.7149 -71562.3626 0.5424 -1.2965 0.1687 32.5**

**19 10893.9581 -82455.5630 -71561.6050 0.5380 -1.2544 0.1682 32.2**

**20 10825.0423 -82385.8646 -71560.8222 0.5346 -1.2219 0.1679 32.2**

**20 0.844637**

**Izvršavanje za osam niti**

**TEST: num\_threads=16**

**1 14551.3619 -93836.0953 -79284.7335 0.7186 -6.1148 0.1953 46.3**

**2 14340.7480 -93654.4811 -79313.7331 0.7082 -6.0186 0.1939 45.6**

**3 13968.2189 -93306.7546 -79338.5357 0.6898 -5.8516 0.1914 44.4**

**4 13404.4586 -92759.7121 -79355.2536 0.6620 -5.6090 0.1876 42.6**

**5 12619.1758 -91985.3542 -79366.1784 0.6232 -5.2880 0.1821 39.7**

**6 11599.4183 -90970.4371 -79371.0188 0.5729 -4.8918 0.1748 35.8**

**7 10379.3400 -89747.3564 -79368.0164 0.5126 -4.4367 0.1655 30.7**

**8 9066.7213 -88424.5967 -79357.8754 0.4478 -3.9592 0.1546 24.7**

**9 7831.4466 -87177.8677 -79346.4211 0.3868 -3.5119 0.1434 18.7**

**10 14619.4170 -86181.5919 -71562.1749 0.7220 -2.8265 0.1336 14.1**

**11 13529.0454 -85087.0949 -71558.0496 0.6682 -2.4076 0.1875 42.0**

**12 12687.0299 -84243.7642 -71556.7343 0.6266 -2.0726 0.1812 38.5**

**13 12082.9134 -83640.4681 -71557.5547 0.5967 -1.8201 0.1766 36.2**

**14 11674.2473 -83233.7353 -71559.4879 0.5765 -1.6380 0.1734 34.7**

**15 11405.1707 -82966.3254 -71561.1547 0.5633 -1.5094 0.1714 33.6**

**16 11223.2084 -82785.5933 -71562.3850 0.5543 -1.4177 0.1701 33.0**

**17 11089.5877 -82652.2025 -71562.6148 0.5477 -1.3495 0.1693 32.9**

**18 10982.3523 -82544.7149 -71562.3626 0.5424 -1.2965 0.1687 32.5**

**19 10893.9581 -82455.5630 -71561.6050 0.5380 -1.2544 0.1682 32.2**

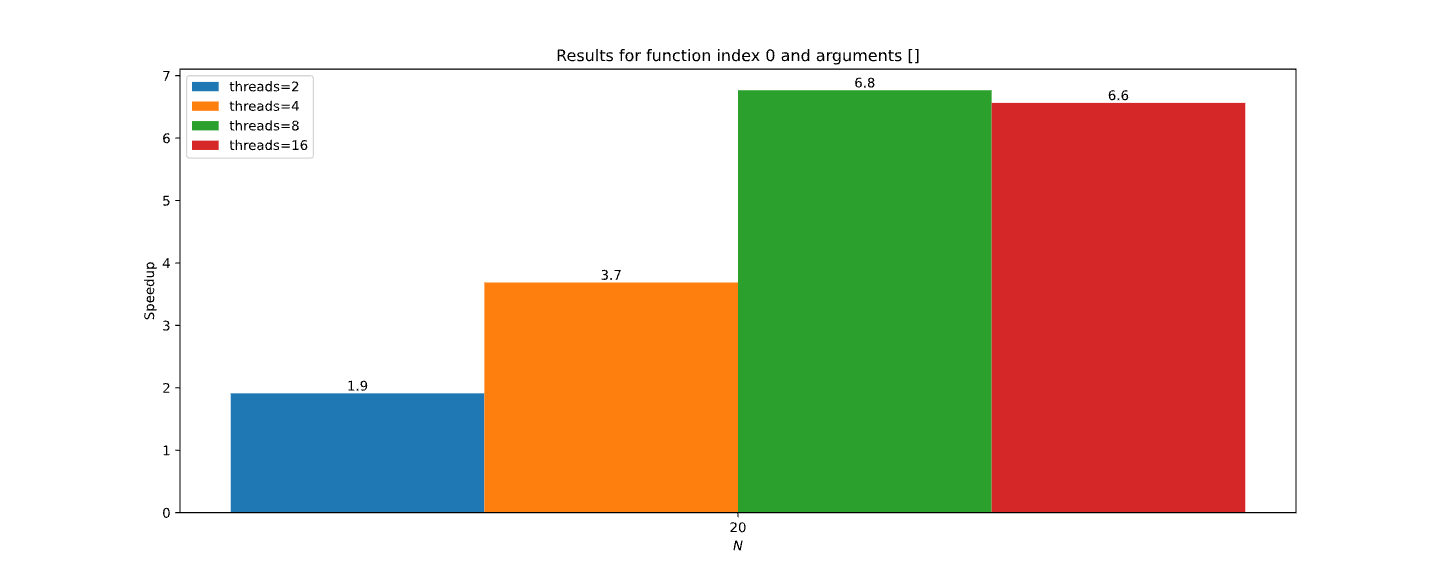
**20 10825.0423 -82385.8646 -71560.8222 0.5346 -1.2219 0.1679 32.2**

**20 0.870708**

**Izvršavanje za šestnaest niti**

### Grafici ubrzanja

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



**Grafik ubrzanja za različit broj niti.**

### Diskusija dobijenih rezultata

Paralelizacija je vidljivo uspešna, sa identičnim rezultatima kao sekvencijalno rešenje.