UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

DOMAĆI ZADATAK 1 – OPENMP

Izveštaj o urađenom domaćem zadatku

Predmetni saradnici:

doc. dr Marko Mišić

dipl. ing. Pavle Divović

Studenti:

Lazar Erić 2019/0235

Aleksa Račić 2019/0235

Beograd, novembar 2022.

SADRŽAJ

1. PROBLEM 1 - PRIME

1.1. 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]

1.2. Delovi koje treba paralelizovati

1.2.1. Diskusija

U okviru ovog zadatka mozemo uociti fise funkcija od kojih je nama od najveceg interesa prime_number. U ostalim funkcijama i main-u ne postoji delova koda koji bi mogli da budu paralelizovani ili smesteni u task-ove.

1.2.2. Način paralelizacije

Kako imamo 2 for petlje u prime_number funkciji i zabranjena je for direktiva, moramo rucno rasporediti iteracije po nitima. Kako je promenljiva total na kraju for-a povecava i samo tu menja, ona ce biti reduction promenljiva. Uočili smo da je najoptimizovanije rešenje ciklično podeliti dati niz i tako dobiti ravnopravnu raspodelu poslova.

1.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1.

1.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Obavezno uključiti u ispis i vremena izvršavanja. Logove pojedinačno uokviriti i obeležiti.

```
--------------------unoptimized--------------------
N      Pi     Time
1      0      0.000125
2      1      0.000001
```

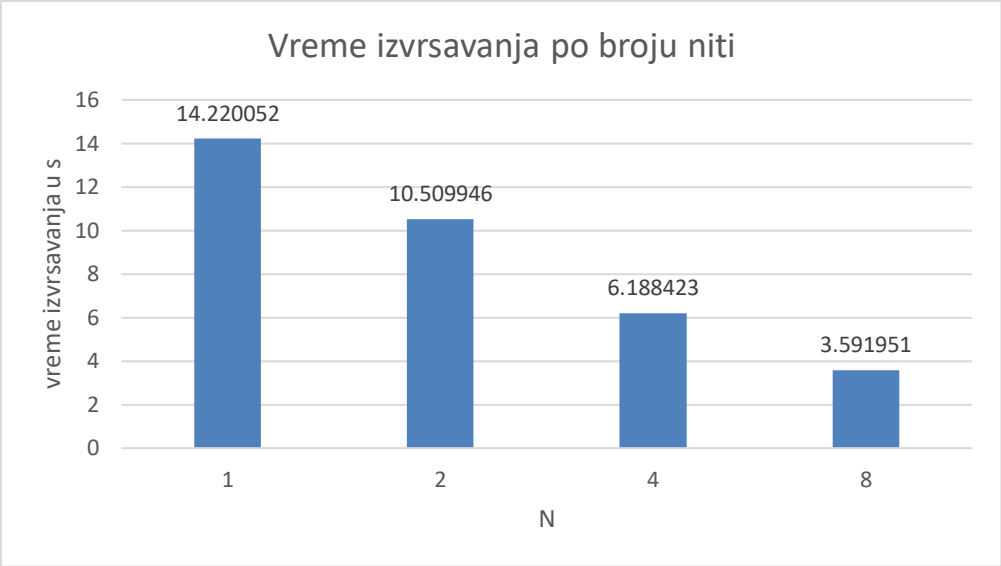
```
4      2      0.000001
8      4      0.000001
16     6      0.000001
32     11     0.000001
64     18     0.000001
128    31     0.000001
256    54     0.000003
512    97     0.000008
1024   172    0.000026
2048   309    0.000092
4096   564    0.000332
8192   1028   0.001218
16384  1900   0.004714
32768  3512   0.016289
65536  6542   0.05972
131072       12251  0.227838

5      3      0.000119
50     15     0.000001
500    95     0.000008
5000   669    0.000511
50000  5133   0.039446
500000       41538  3.176562

1      0      0.000104
4      2      0.000001
16     6      0.000001
64     18     0.000001
256    54     0.000003
1024   172    0.000026
4096   564    0.00033
16384  1900   0.004619
65536  6542   0.059847

TOTAL TIME : 3.591951

--------------------optimized--------------------
N      Pi     Time
1      0      0.00011
```

```
2      1      0.000001
4      2      0.000001
8      4      0.000001
16     6      0.000001
32     11     0.000001
64     18     0.000001
128    31     0.000001
256    54     0.000001
512    97     0.000002
1024   172    0.000002
2048   309    0.000005
4096   564    0.000011
8192   1028   0.000028
16384  1900   0.000071
32768  3512   0.00018
65536  6542   0.000466
131072       12251  0.001225

5      3      0.000069
50     15     0.000001
500    95     0.000002
5000   669    0.000014
50000  5133   0.000313
500000       41538  0.007572

1      0      0.000079
4      2      0.000001
16     6      0.000001
64     18     0.000001
256    54     0.000001
1024   172    0.000003
4096   564    0.000011
16384  1900   0.000067
65536  6542   0.000453

TOTAL TIME : 0.010696
```

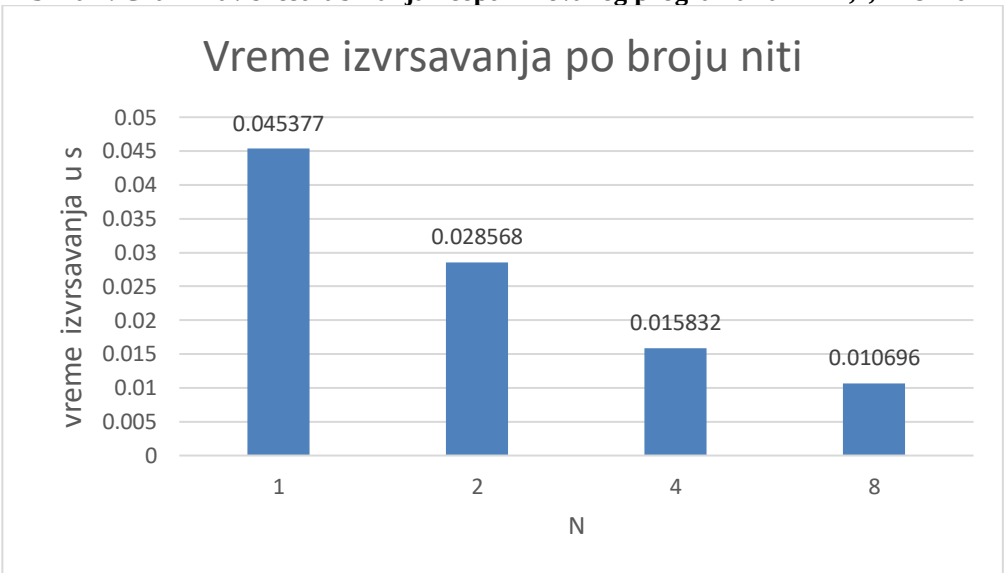
Listing 1. Paralelno neoptimizovano i optimizovano izvršavanje programa

1.3.2. Grafici ubrzanja

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



Slika 1. Grafik zavisnosti ubrzanja neoptimizovanog programa za N = 1,2,4 I 8 niti



Slika 2. Grafik zavisnosti ubrzanja optimizovanog programa za N = 1,2,4 I 8 niti

1.3.3. Diskusija dobijenih rezultata

Sa datih grafika možemo primeniti da je ubrzanje sve uocljivije kako se povecava broj niti i kako se povecava. Takodje, vidi se ogromna razlika izmedju optimizovanog i neoptimizovanog koda.

2. PROBLEM 2 - PRIME

2.1. Tekst problema

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

2.2. Delovi koje treba paralelizovati

2.2.1. Diskusija

Posto se radi o istom zadatku kao problem 1, uzimamo isti deo da paralelizujemo i ostaje isti sekvencijalan deo.

2.2.2. Način paralelizacije

S obzirom da mozemo koristiti worksharing direktive sada koristimo ***for*** direktivu za dinamicko rasporedjivanje posla po nitima, promenljive ostaju u istom scope-u kao i u primeru 1.

2.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 12

2.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Obavezno uključiti u ispis i vremena izvršavanja. Logove pojedinačno uokviriti i obeležiti.

```
--------------------unoptimized--------------------
N      Pi     Time
1      0      0.0001
2      1      0.000002
4      2      0.000001
8      4      0.000001
16     6      0.000002
32     11     0.000001
64     18     0.000001
128    31     0.000002
256    54     0.000003
512    97     0.000009
```

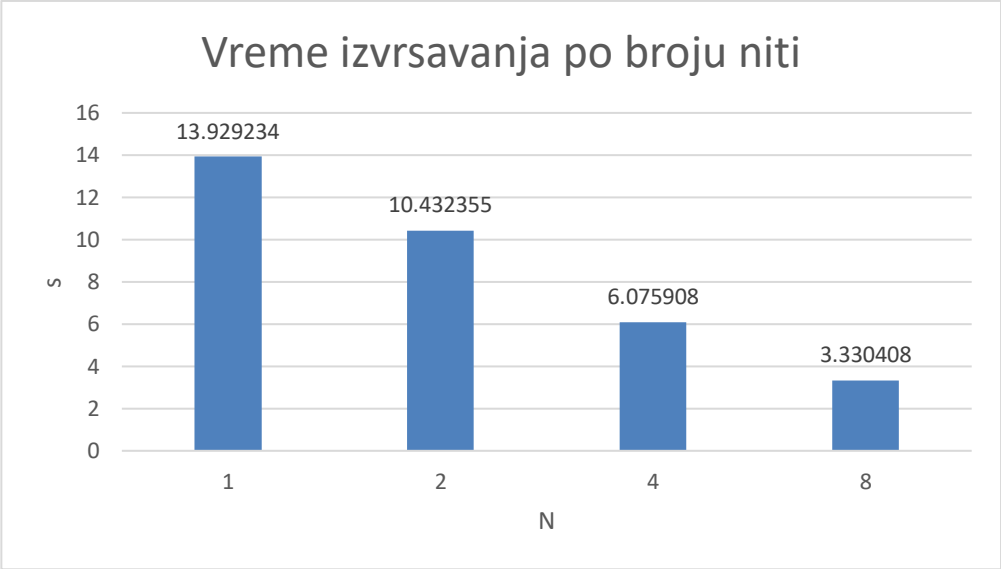
```
1024  172   0.000028
2048  309   0.000093
4096  564   0.00036
8192  1028  0.001297
16384 1900  0.004971
32768 3512  0.017499
65536 6542  0.063252
131072      12251 0.22461

5     3     0.000099
50    15    0.000002
500   95    0.000009
5000  669   0.000476
50000 5133  0.036138
500000      41538 2.912212

1     0     0.000129
4     2     0.000001
16    6     0.000008
64    18    0.000001
256   54    0.000014
1024  172   0.000028
4096  564   0.000363
16384 1900  0.004922
65536 6542  0.063774

TOTAL TIME : 3.330408
--------------------optimized--------------------
N     Pi    Time
1     0     0.000069
2     1     0.000002
4     2     0.000001
8     4     0.000001
16    6     0.000001
32    11    0.000001
64    18    0.000002
128   31    0.000002
256   54    0.000002
512   97    0.000002
```

```
1024  172   0.000003
2048  309   0.000005
4096  564   0.00001
8192  1028  0.000023
16384 1900  0.000057
32768 3512  0.00014
65536 6542  0.000359

131072      12251 0.000945
5     3     0.000103
50    15    0.000002
500   95    0.000002
5000  669   0.000013
50000 5133  0.000251

500000      41538 0.006045
1     0     0.000067
4     2     0.000001
16    6     0.000001
64    18    0.000002
256   54    0.000002
1024  172   0.000003
4096  564   0.000012
16384 1900  0.000072
65536 6542  0.00047

TOTAL TIME : 0.008671
```

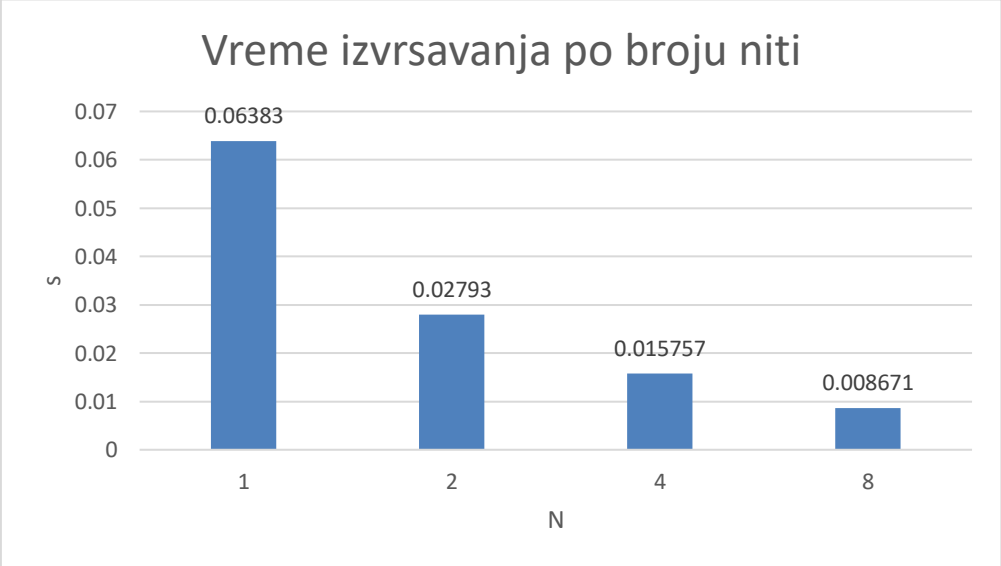
Listing 2. Paralelno neoptimizovano i optimizovano izvršavanje programa

2.3.2. Grafici ubrzanja

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



Slika 3. Grafik zavisnosti ubrzanja neoptimizovanog programa za N = 1,2,4 i 8 niti



Slika 4. Grafik zavisnosti ubrzanja optimizovanog programa za N = 1,2,4 i 8 niti

2.3.3. Diskusija dobijenih rezultata

Sa datih grafika možemo primeniti da je ubrzanje uočljivo i da je sve veće kako je i broj niti veći. Takodje se vidi i ogromna razlika u vremenu izvrsavanja neoptimizovanog i optimizovanog programa.

3. PROBLEM 3 - FEYMAN

3.1. Tekst problema

Paralelizovati program koji vrši izračunavanje 3D Poasonove jednačine korišćenjem Feyman-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 **feyman.c** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci

run. [1, N]

3.2. Delovi koje treba paralelizovati

3.2.1. Diskusija

U okviru ovog problema vidimo veci broj funkcija, ali one su sve pomocne koje nam sluze da izvrsimo neke jednostavne matematicke operacije. Zato se one izvrsavaju sekvencijalno, a najveci fokus je na main-u gde ima 6 ugnjezndenih *for* pelji.

3.2.2. Način paralelizacije

Posmatrajuci prve tri for petlje uocavamo da imaju relativno mali broj iteracija, sve tri ukupno do 150-200. Onda smo se skoncentrisali na inutrasnje 3, koje iterariraju do broja N, uradili smo redukciju podataka steps i wt, a paralelizaciju realizovali uz pomoc direktive ***for*** uz schedule static sa chunksize-om 10 koji smo nasli da je najomptimalniji.

3.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 3.

3.3.1. Logovi izvršavanja

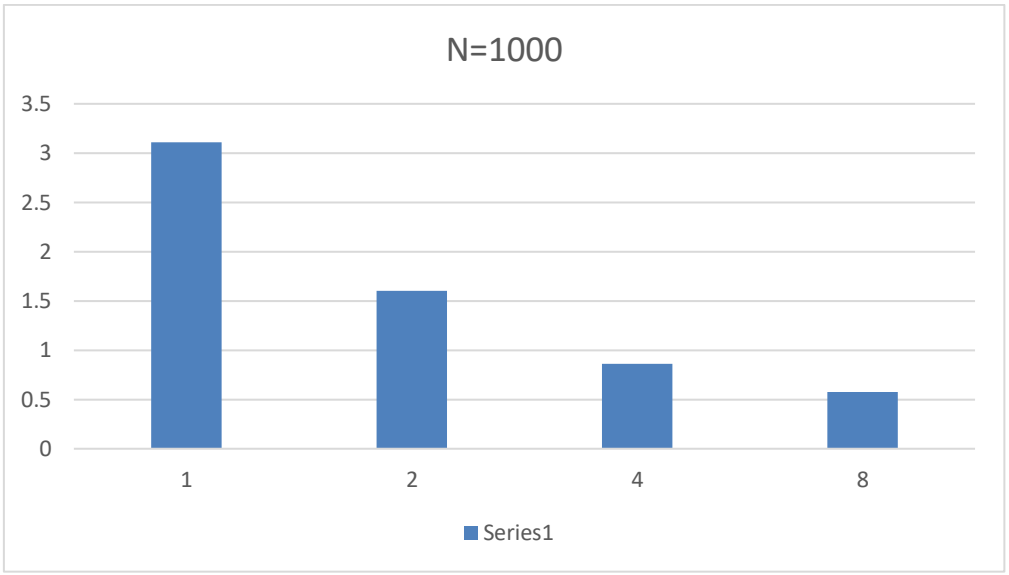
Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Obavezno uključiti u ispis i vremena izvršavanja. Logove pojedinačno uokviriti i obeležiti.

```
Num_Threads  N     RMS    Time

1     1000  0.021717    3.108806
2     1000  0.023437    1.60029
4     1000  0.024621    0.862008
8     1000  0.028722    0.576902

1     5000  0.021273    15.592906
2     5000  0.021362    7.953312
4     5000  0.021628    4.32517
8     5000  0.022805    2.761974

1     10000 0.0211      31.161119
2     10000 0.021273    15.781168
4     10000 0.021362    8.697477
8     10000 0.021628    5.249334

1     20000 0.021027    62.344694
2     20000 0.0211      32.263005
4     20000 0.021273    17.379135
8     20000 0.021362    10.514776
```

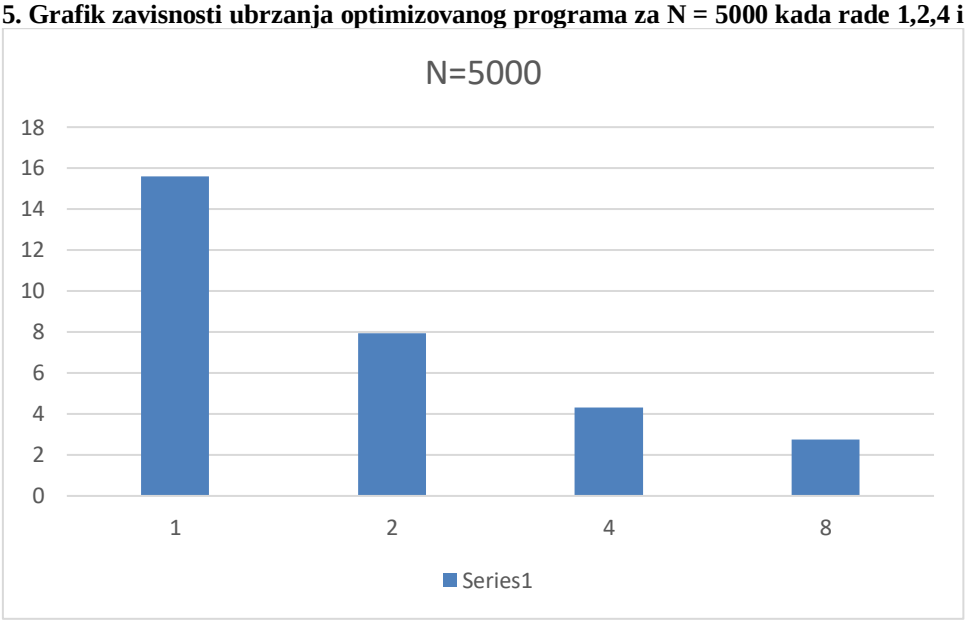
Listing 3.

3.3.2. Grafici ubrzanja

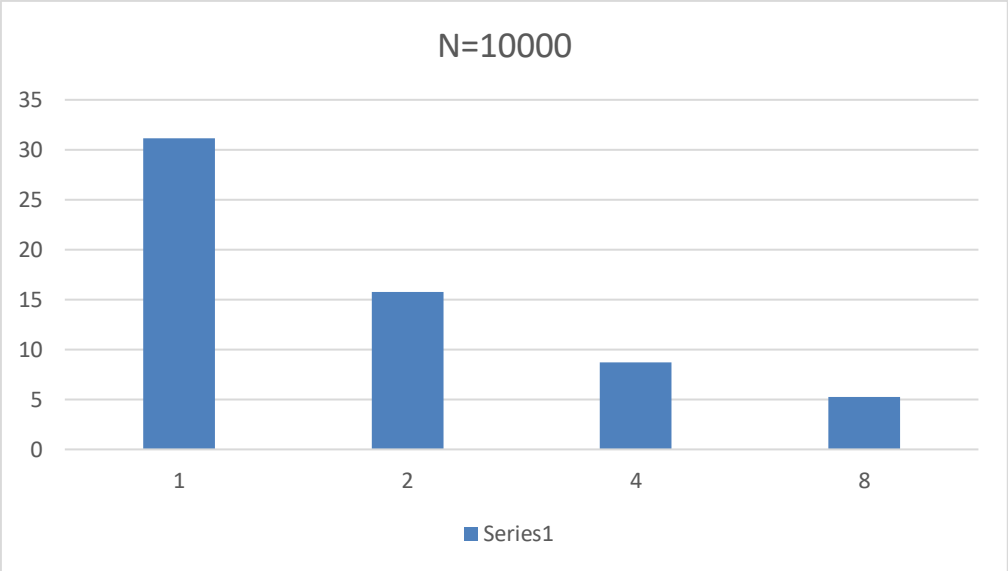
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



Slika 5. Grafik zavisnosti ubrzanja optimizovanog programa za N = 5000 kada rade 1,2,4 i 8 niti



Slika 6. Grafik zavisnosti ubrzanja programa za N = 5000 kada rade 1,2,4 i 8 niti



Slika 7. Grafik zavisnosti ubrzanja programa za N = 10000 kada rade 1,2,4 i 8 niti

Slika 8. Grafik zavisnosti ubrzanja programa za N = 20000 kada rade 1,2,4 i 8 niti

3.3.3. Diskusija dobijenih rezultata

Sa datih grafika možemo primeniti da je ubrzanje uočljivo i da je sve veće kako je i broj niti veći. Takodje, jasno se vidi povecanje vremena izvrsavanja kada se povecava broj N.

4. PROBLEM 4 - FEYMAN

4.1. 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]

4.2. Delovi koje treba paralelizovati

4.2.1. Diskusija

U okviru ovog problema vidimo veci broj funkcija, ali one su sve pomocne koje nam sluze da izvrsimo neke jednostavne matematicke operacije. Zato se one izvrsavaju sekvencijalno, a najveci fokus je na main-u gde ima 6 ugnjezndenih *for* pelji.

4.2.2. Način paralelizacije

Posto je zadatak da se izvrsi uz pomoc task direktive, jedan task smo stavili da budu 3 najvise ugnjezdene for petlje koje imaju najvise iteracija. Kreiramo tim niti pre početka prve for petlje, zatim direktivom single obezbedimo da samo jedna nit prodje kroz prve 3 petlje i kreira taskove. Izvršili smo paralelizaciju najveće petlje, odnosno unutrašnje petlje, koja zavisi od ulaznog podatka N. Paralelizaciju smo izvršili pomoću task direktive. Posto err deljena između taskova, obezbedili smo sinhronizaciju tako sto smo je stavili u atomic region.

4.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 4.

4.3.1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Obavezno uključiti u ispis i vremena izvršavanja. Logove pojedinačno uokviriti i obeležiti.

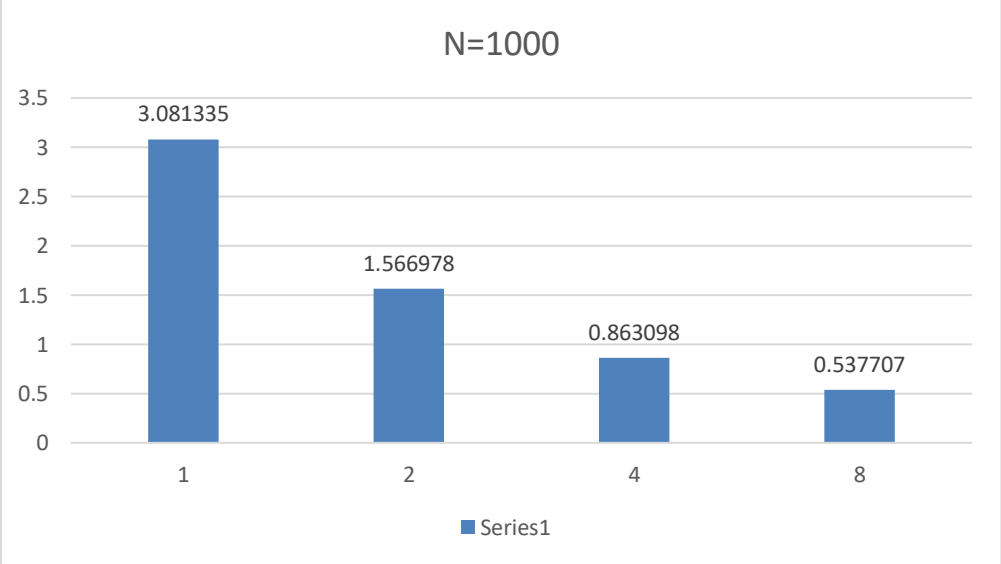
```
Num_Threads  N     RMS    Time
1     1000  0.022279    3.081335
2     1000  0.021339    1.566978
4     1000  0.021693    0.863098
8     1000  0.021828    0.537707
```

```
1       5000   0.021142     15.62424
2       5000   0.021044     7.839491
4       5000   0.020859     4.326347
8       5000   0.020877     2.610634

1       10000  0.021053     31.231582
2       10000  0.020642     15.65313
4       10000  0.021124     8.657841
8       10000  0.020922     5.226691
1       20000  0.020913     61.261271

2       20000  0.020939     31.218166
4       20000  0.020901     17.305086
8       20000  0.020976     10.45544
```

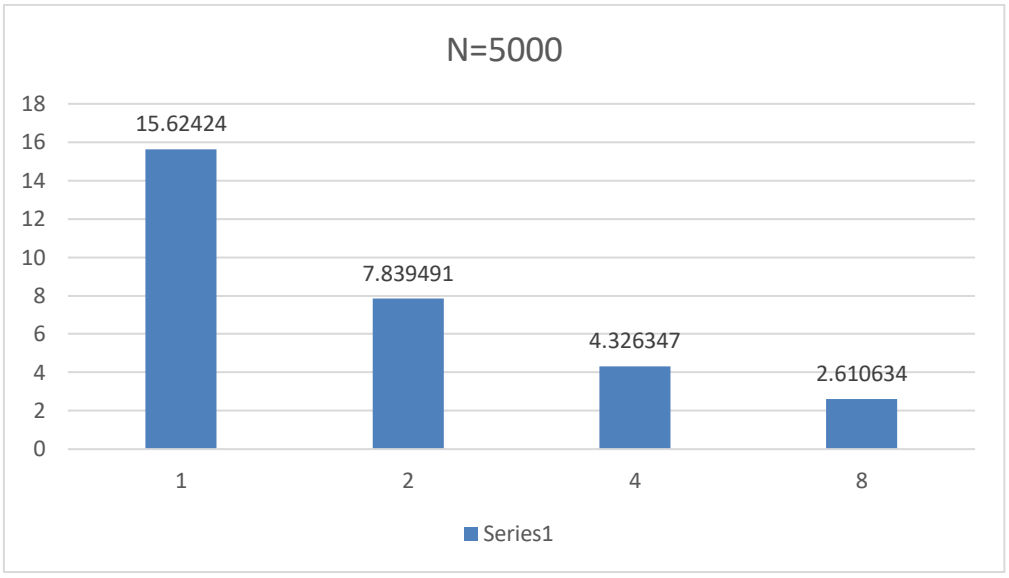
Listing 4.

4.3.2. Grafici ubrzanja

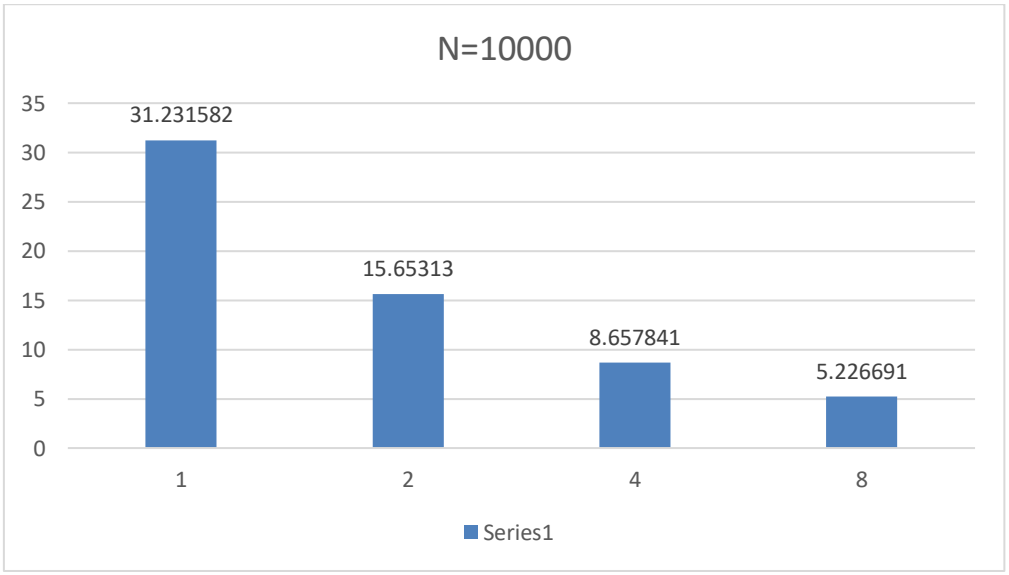
U okviru ove sekcije su dati grafici ubrzanja u odnosu na sekvencijalnu implementaciju.



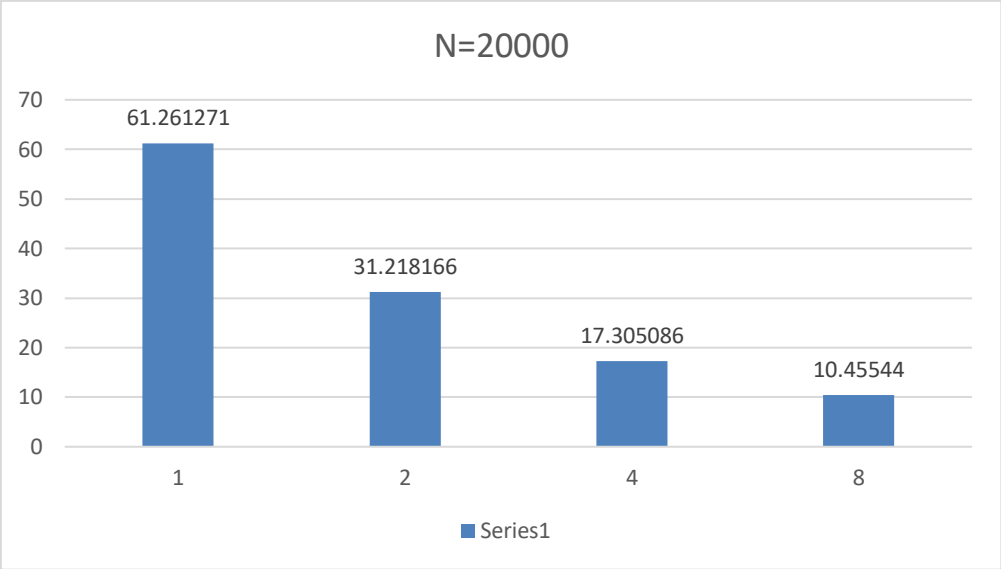
Slika 9. Grafik zavisnosti ubrzanja programa za N = 1000 kada rade 1,2,4 i 8 niti



Slika 10. Grafik zavisnosti ubrzanja programa za N = 5000 kada rade 1,2,4 i 8 niti



Slika 11. Grafik zavisnosti ubrzanja programa za N = 10000 kada rade 1,2,4 i 8 niti



Slika 12. Grafik zavisnosti ubrzanja programa za N = 20000 kada rade 1,2,4 i 8 niti

4.3.3. Diskusija dobijenih rezultata

Sa datih grafika možemo primeniti da je ubrzanje uočljivo i da je sve veće kako je i broj niti veći. Takodje, jasno se vidi povecanje vremena izvrsavanja kada se povecava broj N.

5. PROBLEM 5 - MOLDYN

5.1. 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]

5.2. Delovi koje treba paralelizovati

5.2.1. Diskusija

U okviru ovog zadatka mozemo uociti fise .c fajlova od kojih je nama od najveceg interesa forces.c i main.c. U ostalim fajlovima se najvise nalazi kod koji ne moze biti apralelizovan.

5.2.2. Način paralelizacije

Paralelizaciju smo uradili u jednom fajlu – forces.c. U fajlu forces.c se odvija najvise izracunavanja. Izvrsili smo paralelizaciju for direktivom i redukciju epot i vir promenjivih. Može nastati problem utrkivanja ukoliko ne obezbedimo sinhronizaciju nad deljenim promenjivama, odnosno članovima niza. To smo izvršili pomoću atomic directive, kojih imao ispred svake promene niza f.

5.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 5.

5.3.1. Logovi izvršavanja

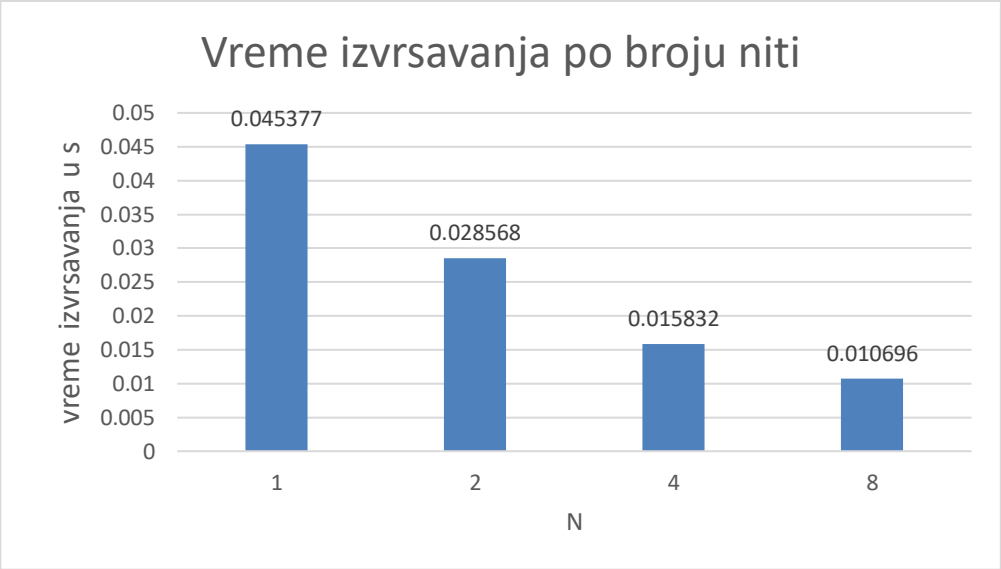
Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti. Obavezno uključiti u ispis i vremena izvršavanja. Logove pojedinačno uokviriti i obeležiti.

```
Thread_Num   Time
1            5.866778
2            3.022172
4            1.558719
8            0.903925
```

Listing 5.

5.3.2. Grafici ubrzanja

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



Slika 13. Grafik zavisnosti ubrzanja programa za N = 1,2,4 I 8 niti

5.3.3. Diskusija dobijenih rezultata

Sa datih grafika možemo primeniti da je ubrzanje sve uocljivije kako se povecava broj niti i kako se povecava.