Univerzitet u Beogradu - Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



**Domaći zadatak 1 – OPENMP**

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni asistent: | Studenti: |
| doc. dr Marko Mišić | Mara Bolić 2017/0614  Edvin Maid 2017/0117 |

Beograd, novembar 2020.

**Sadržaj**

***[Problem 1 – Pi Calc bez podele posla](#_heading=h.goxsgvm9mb7y) 4***

*[Tekst problema](#_heading=h.1fob9te) 4*

*[Diskusija](#_heading=h.2et92p0) 4*

*[Način paralelizacije](#_heading=h.tyjcwt) 4*

*[Rezultati](#_heading=h.3dy6vkm) 4*

*[Logovi izvršavanja](#_heading=h.1t3h5sf) 4*

*[Grafici ubrzanja](#_heading=h.4d34og8) 7*

*[Diskusija dobijenih rezultata](#_heading=h.wdhcxoaaeytx) 8*

***[Problem 2 – Pi Calc sa podelom posla](#_heading=h.shwmku5iyq8g) 9***

*[Tekst problema](#_heading=h.55l41lqr8xjr) 9*

*[Diskusija](#_heading=h.5fd7kwbpk58o) 9*

*[Način paralelizacije](#_heading=h.szgqm9pexgih) 9*

*[Rezultati](#_heading=h.ecevh657ijbk) 9*

*[Logovi izvršavanja](#_heading=h.rw977u49jmmu) 9*

*[Grafici ubrzanja](#_heading=h.lg1hsvfsbv06) 12*

*[Diskusija dobijenih rezultata](#_heading=h.ofhumhj71bgg) 13*

***[Problem 3 – Pi Calc koriscenjem poslova](#_heading=h.i262382c5w6g) 14***

*[Tekst problema](#_heading=h.t7kciam1w0cl) 14*

*[Diskusija](#_heading=h.kb1i2xq6yh0) 14*

*[Način paralelizacije](#_heading=h.cr4remgfckp6) 14*

*[Rezultati](#_heading=h.c4k3mwg44c01) 14*

*[Logovi izvršavanja](#_heading=h.4p4geo45ogg3) 14*

*[Grafici ubrzanja](#_heading=h.cned117ll0cw) 17*

*[Diskusija dobijenih rezultata](#_heading=h.y88q6hj2cyku) 18*

***[Problem 4 – Needleman-Wunsch](#_heading=h.b106xuch51e9) 19***

*[Tekst problema](#_heading=h.v529r8twir76) 19*

*[Diskusija](#_heading=h.nofe6n77aa4g) 19*

*[Način paralelizacije](#_heading=h.t6kjmf7qslk3) 19*

*[Rezultati](#_heading=h.apmny2pou401) 19*

*[Logovi izvršavanja](#_heading=h.5fl4u9rihv44) 19*

*[Grafici ubrzanja](#_heading=h.qep3d97znuuk) 21*

*[Diskusija dobijenih rezultata](#_heading=h.dxzqpdubze22) 22*

***[Problem 5 – Nbody](#_heading=h.ajbjs7t5gljj) 23***

*[Tekst problema](#_heading=h.gsabqr8t8jha) 23*

*[Diskusija](#_heading=h.4w3pfa39q2sp) 23*

*[Način paralelizacije](#_heading=h.xpzc0bnk47i) 23*

*[Rezultati](#_heading=h.okvdgx34yh3o) 23*

*[Logovi izvršavanja](#_heading=h.bdfg7hzc183w) 23*

*[Grafici ubrzanja](#_heading=h.4v746fsn2fxa) 25*

*[Diskusija dobijenih rezultata](#_heading=h.1xplgymh7u7w) 25*

# Problem 1 – Pi Calc bez podele posla

## Tekst problema

Paralelizovati program koji izračunava vrednost broja PI korišćenjem formule: 𝜋 = 4 ∗ ∑ (−1) 𝑘+1 2𝑘−1 𝑛 𝑘=1 . Tačnost izračunavanja direktno zavisi od broja iteracija, a zbog malog radijusa konvergencije serija konvergira veoma sporo. Program se nalazi u datoteci piCalc.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

U ovom zadatku paralelizovana je jedina *for* petlja. Ostatak koda je sekvencijalan i nema potrebe za njegovom paralelizacijom.

### Način paralelizacije

Zbog toga sto je zabranjeno korišćenje direktiva za podelu poslova oslanjamo se na *reduction* dok mi samo vršimo raspodelu poslova korišćenjem funkcija *omp\_get\_num\_threads* i *omp\_get\_thread\_num* kako bismo mogli da raspodelimo da niti sa identitetom *myid* računaju sabirci sa indeksima *myid, myid + num\_threads, myid + 2\*num\_threads,* … i tako postižemo najbolji balans poslova.

## Rezultati

### Logovi izvršavanja

Ovde su dati logovi izvršavanja paralelnog i sekvencijalnog programa za definisane test primere iz *run* fajla.

**1000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358977**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.009684**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358956**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.002179**

**Test PASSED**

**10000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358979**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.031867**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358947**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.007998**

**Test PASSED**

**100000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264358933**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.318410**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264359058**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.079724**

**Test PASSED**

**1000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265258805**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 3.183659**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265259308**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.796891**

**Test PASSED**

**10000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265348835**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 31.834916**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265349295**

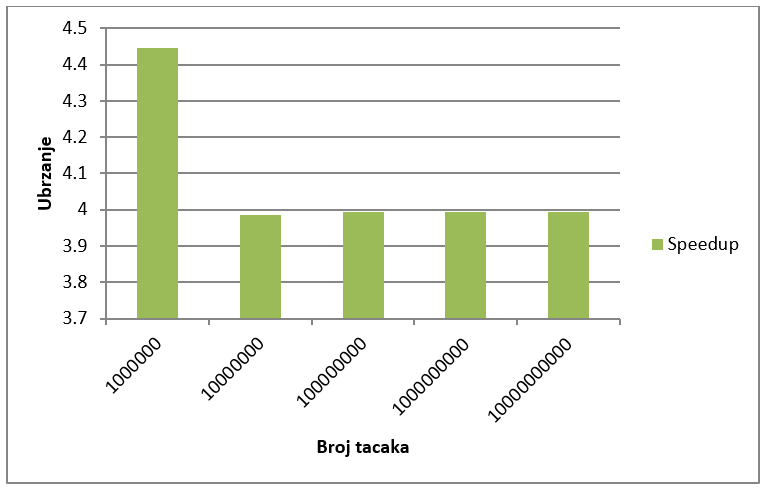
**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 7.968678**

**Test PASSED**

**Listing 1. Sekvencijalna i paralelna izvršavanja Pi Calc**

### Grafici ubrzanja



**Slika 1. Grafik zavisnosti ubrzanja za N = 8 niti**

### Diskusija dobijenih rezultata

Za prvi test primer se dobija malo veće ubrzanje od 4.5 od ostalih test primera dok za ostale ubrzanje je paralelni program brži od sekvencijalnog oko 4 puta.

# 

# Problem 2 – Pi Calc sa podelom posla

## Tekst problema

Prethodni program paralelizovati korišćenjem direktiva za podelu posla (worksharing direktive). Obratiti pažnju na raspodelu opterećenja po nitima i testirati program za različite načine raspoređivanja posla. Program testirati sa parametrima koji su dati u datoteci run. [1, N]

### Diskusija

Postoji jedna for petlje koja može da se paralelizuje i to se radi direktivama za podelu posla.

### Način paralelizacije

Koristi se direktiva za podelu posla sa static raspodelom jer se posao fiksne veličine i ovo rešenje nudi najveći balans sa najmanje režijskih troškova.

## Rezultati

### Logovi izvršavanja

Ovde su dati logovi izvršavanja paralelnog i sekvencijalnog programa za definisane test primere iz *run* fajla.

**1000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358977**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.003200**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358973**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.003217**

**Test PASSED**

**10000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358979**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.031857**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358983**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.009062**

**Test PASSED**

**100000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264358933**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.318659**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264358988**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.093912**

**Test PASSED**

**1000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265258805**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 3.184916**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265258932**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.898941**

**Test PASSED**

**10000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265348835**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 31.848515**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265348821**

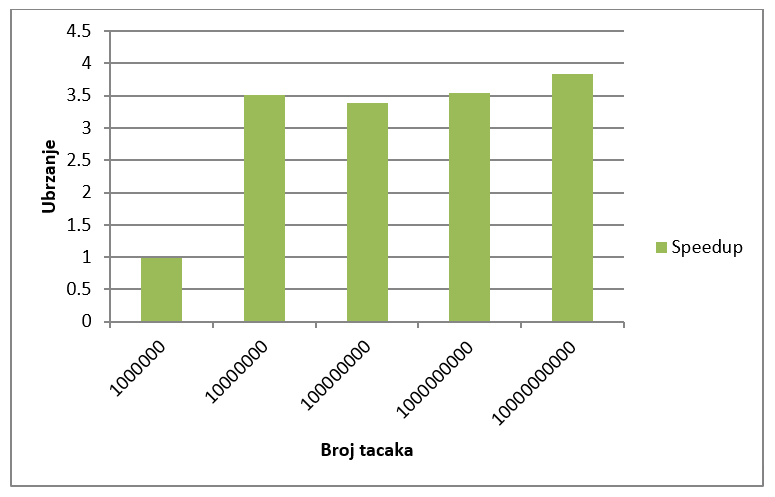
**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 8.309825**

**Test PASSED**

**Listing 1. Sekvencijalna i paralena izvršavanja Pi Calc**

### Grafici ubrzanja



**Slika 1. Grafik zavisnosti ubrzanja za N = 8 niti**

### Diskusija dobijenih rezultata

Za manji broj tačaka (do 1000000) se ne isplati raditi paralelizaciju jer se ne dobija nikakvo ubrzanje, dok za veći broj tačaka ubrzanje moze biti 3 do 4 puta veće u odnosu na sekvencijalno izvršavanje. Kako je problem fiksane veličine *static* raspodela posla se pokazala kao najbolja zbog adekvatnog balansiranja poslova, a pritom sa minimalnim režijskim troškovima za razliku od *dynamic* raspodele koja se pokazala kao primetno sporija.

# Problem 3 – Pi Calc koriscenjem poslova

## Tekst problema

Rešiti prethodni problem korišćenjem koncepta poslova (tasks). Obratiti pažnju na eventualnu potrebu za sinhronizacijom. Rešenje testirati i prilagoditi tako da granularnost poslova bude optimalna. Program testirati sa parametrima koji su dati u datoteci run. [1, N]

### Diskusija

Jedina petlja koja vrši obradu se se može optimizovati tako da se zameni petljom koja sekvencijalno stvara poslove koji će obaviti računanje.

### Način paralelizacije

Bitno je rasporediti izračunavanja po poslovima adekvatno. Uveli smo promenjivu gran kojom smo regulisali granularnost tj. definisali smo koliko sabiralaca će obrađivati jedna nit međutim se pokazalo da je najbolja paralelizacija kada poslova ima koliko i niti koje mogu da izvršavaju paralelno jer sa manjim granularnostima su veći režijski troškovi.

## Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1.

### Logovi izvršavanja

Ovde su dati logovi izvršavanja paralelnog i sekvencijalnog programa za definisane test primere iz *run* fajla.

**1000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358977**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.005412**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000 terms**

**Our estimate of pi = 3.14159165358973**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.003721**

**Test PASSED**

**10000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358979**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.035853**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000 terms**

**Our estimate of pi = 3.14159255358983**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.008008**

**Test PASSED**

**100000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264358933**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.318307**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 100000000 terms**

**Our estimate of pi = 3.14159264358988**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.079761**

**Test PASSED**

**1000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265258805**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 3.183303**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 1000000000 terms**

**Our estimate of pi = 3.14159265258932**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 0.797661**

**Test PASSED**

**10000000000 points**

**SEQUENTIAL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = -1.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265348835**

**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 31.833884**

**PARALLEL**

**Before for loop, factor = 0.000000.**

**After for loop, factor = 0.000000.**

**With n = 10000000000 terms**

**Our estimate of pi = 3.14159265348821**

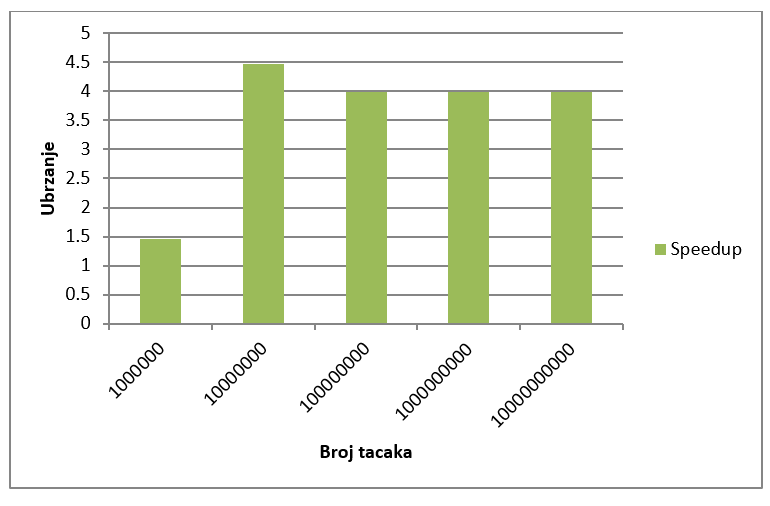
**Ref estimate of pi = 3.14159265358979**

**Elapsed time: 7.972028**

**Test PASSED**

**Listing 1. Sekvencijalna i paralelna izvršavanja Pi Calc**

### Grafici ubrzanja



**Slika 1. Grafik zavisnosti ubrzanja za N = 8 niti**

### Diskusija dobijenih rezultata

Za manji broj tačaka (do 1000000) se dobija veoma malo ubrzanje (oko 1.5), dok za veći broj tačaka ubrzanje može biti 3.5 do 4.5 puta veće u odnosu na sekvencijalno izvršavanje. Za raspoređivanje u kojim jedan task obuhvata manji broj izračunavanja smo dobili da je izuzetno sporo izračunavanje, međutim kada smo podesili da svaki posao izvršava skoro podjednak deo poslova dobili smo značajno ubrzanje.

# Problem 4 – Needleman-Wunsch

## Tekst problema

Paralelizovati program koji vrši poravnavanje bioloških sekvenci korišćenjem Needleman-Wunsch algoritma. Algoritam predstavlja primenu koncepta dinamičkog programiranja za globalno poravnavanje dve sekvence nukleotida ili aminokiselina. Program se nalazi u datoteci needle.c u arhivi koja je priložena uz ovaj dokument. Obratiti pažnju na raspodelu opterećenja po nitima i testirati program za različite načine raspoređivanja posla. Program testirati sa parametrima koji su dati u datoteci run. [1, N]

### Diskusija

Ovaj problem ima više petlji koje treba paralelizovati. Petlje koje sluze za inicijalizaciju se mogu slobodno paralelizovati osim petlje koja poziva funkciju *rand()* koja nije pouzdana za višenitno izvršavanje. Samo unutrasnja petlja dvostruke petlje obilaska se može paralelizovati za oba obilaska.

### Način paralelizacije

Za petlje koje inicijalizuju vrednosti koristi se obicna podela poslova a u slučaju da je ugnježdena petlja onda koristimo *collapse* direktivu. Za glavne petlje koristimo podelu poslova isključivo za unutrašnju petlju. Kako se iteracije kreću dijagonalno onda svaka iteracija spoljašnje petlje zavisi direktno od prethodne iteracije dok su iteracije unutrašnje petlje međusobno nezavisne.

## Rezultati

### Logovi izvršavanja

Ovde su dati logovi izvršavanja paralelnog i sekvencijalnog programa za definisane test primere iz *run* fajla.

**2048, 10**

**STARTING SEQUENTIAL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (sequetial): 0.081149**

**STARTING PARALLEL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (parallel): 0.035569**

**6144, 10**

**STARTING SEQUENTIAL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (sequetial): 0.969279**

**STARTING PARALLEL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (parallel): 0.461546**

**16384, 10**

**STARTING SEQUENTIAL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (sequetial): 8.915067**

**STARTING PARALLEL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (parallel): 3.676323**

**22528, 10**

**STARTING SEQUENTIAL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

**Processing bottom-right matrix**

**Elapsed time (sequetial): 16.647010**

**STARTING PARALLEL**

**Start Needleman-Wunsch**

**Processing top-left matrix**

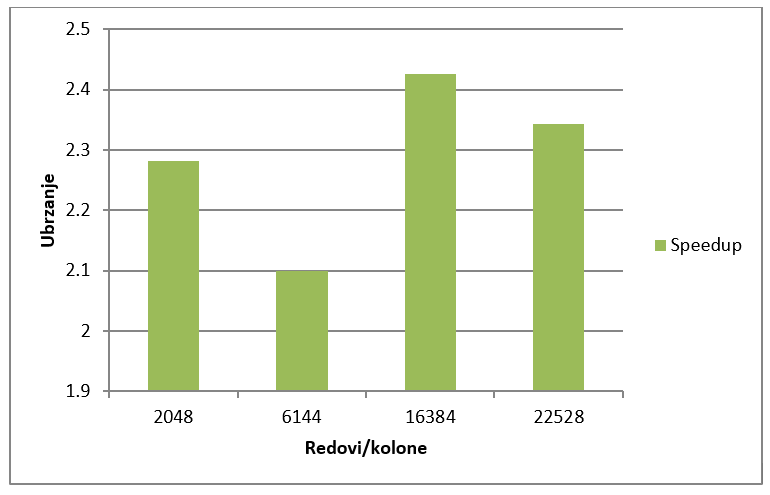
**Processing bottom-right matrix**

**Elapsed time (parallel): 7.103133**

**Listing 1. Sekvencijalno izvršavanje Needleman-Wunsch**

### Grafici ubrzanja

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



**Slika 1. Grafik zavisnosti ubrzanja za N = 8 niti**

### Diskusija dobijenih rezultata

Za sve test primere ubrzanje paralelnog programa iznosi izmedju 2 i 2.5 u odnosu na sekvencijalni program. Ovaj zadatak je testiran sa statičkim i dinamičkim raspoređivanjem posla. Statički rezlulatati daju znatno bolje rezultate zbog režijskih troškova dinamičke podele posla.

# Problem 5 – Nbody

## Tekst problema

Paralelizovati program koji simulira problem interakcije čvrstih tela u dvodimenzionalnom prostoru (nbody problem). Tela interaguju putem gravitacione sile na osnovu sopstvene mase, pozicije u prostoru i trenutne brzine. Program se nalazi u direktorijumu nbody u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih je od interesa datoteka nbody.c. Analizirati dati kod i obratiti pažnju na način izračunavanja sila i energija. Ukoliko je potrebno međusobno isključenje prilikom paralelizacije programa, koristiti dostupne OpenMP konstrukte. Obratiti pažnju na efikasnost međusobnog isključenja niti i po potrebi ga svesti na što je moguće manju meru uvođenjem pomoćnih struktura podataka. Verifikaciju paralelizovanog rešenja vršiti nad dobijenim energijama i poslednjem stanju sistema. Način pokretanja programa se nalazi u datoteci run. [1, N]

### Diskusija

Petlja za računaje kinetičke energije u Compute\_energy funkciji se slobodno paralelizuje. Ugnježdena (unutrašnja) petlja za računanje potencijalne energije se paralelizuje.

### Način paralelizacije

Petlja za računanje kinetičke energije se optimizuje korišćenjem aditivne redukcije nad ukupno kinetičkom energijom sa statičkom raspodelom poslova koja je i najbolja za problem fiksne veličine. Petlja za računanje potencijalne energije se optimizuje ali samo unutrašnja.

Petlju u Compute\_force funkciji smo paralelizovali podelom poslova sa redukcijom međutim inicijalno smo pokušali da direktno pristupamo forces nizu i povećavamo sumu, ali je to zahtevalo korišćenje atomic direktive koja je znatno usporavala izvršavanje zbog međusobnog čekanja niti. Kada smo pristup forces nizu izbacili iz petlje i uklonili atomic direktivu ubrzanje je bilo višestruko.

## Rezultati

### Logovi izvršavanja

Ovde su dati logovi izvršavanja paralelnog i sekvencijalnog programa za definisane test primere iz *run* fajla.

**100**

**SEQUENTIAL**

**PE = -6.985593e+36, KE = 2.250000e+35, Total Energy = -6.760593e+36**

**PE = -7.035612e+36, KE = 1.304554e+36, Total Energy = -5.731058e+36**

**Elapsed time = 3.727818e-02 seconds**

**PARALLEL**

**PE = -6.985593e+36, KE = 2.250000e+35, Total Energy = -6.760593e+36**

**PE = -7.035612e+36, KE = 1.304554e+36, Total Energy = -5.731058e+36**

**Elapsed time = 1.503344e-01 seconds**

**Test PASSED**

**500**

**SEQUENTIAL**

**PE = -4.831939e+37, KE = 1.125000e+36, Total Energy = -4.719439e+37**

**PE = -4.754056e+37, KE = 1.360414e+36, Total Energy = -4.618014e+37**

**Elapsed time = 8.025111e-01 seconds**

**PARALLEL**

**PE = -4.831939e+37, KE = 1.125000e+36, Total Energy = -4.719439e+37**

**PE = -4.754056e+37, KE = 1.360414e+36, Total Energy = -4.618014e+37**

**Elapsed time = 8.618548e-01 seconds**

**Test PASSED**

**5000**

**SEQUENTIAL**

**PE = -6.751832e+38, KE = 1.125000e+37, Total Energy = -6.639332e+38**

**PE = -6.649074e+38, KE = 2.481116e+36, Total Energy = -6.624263e+38**

**Elapsed time = 8.002957e+01 seconds**

**PARELLEL**

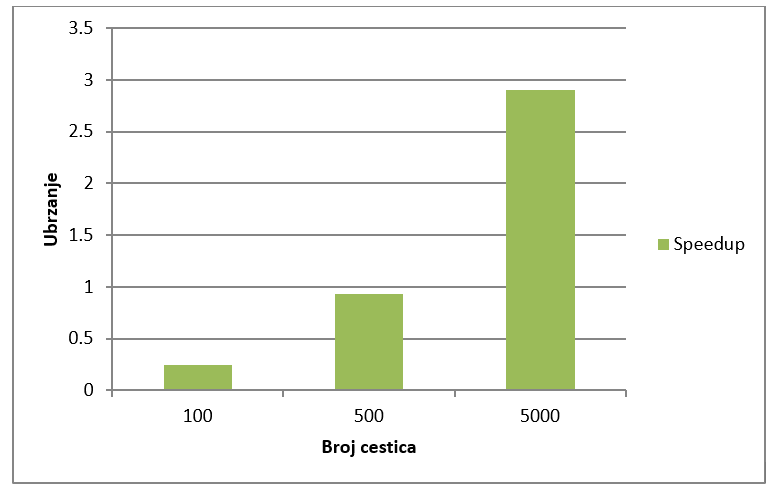
**PE = -6.751832e+38, KE = 1.125000e+37, Total Energy = -6.639332e+38**

**PE = -6.649074e+38, KE = 2.481116e+36, Total Energy = -6.624263e+38**

**Elapsed time = 2.758442e+01 seconds**

**Test PASSED**

### Grafici ubrzanja



**Slika 1. Grafik zavisnosti ubrzanja za N = 8 niti (primer)**

### Diskusija dobijenih rezultata

Za mali broj čestica (100) se korišćenjem paralelizacije dobija suprotan efekat od žljenog odnosno usporenje. Sa većim brojem (500) čestica se paralelni program izjednačava sa sekvencijalnim. Tek sa brojem čestica 5000 se primećuje znatno ubrzanje oko 2.5 do 3 puta brzine sekvencijalnog izvršavanja.