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 | Bogdan Bebić 2017/0011  Uroš Krstić 2017/0228 |

Beograd, novembar 2020.

Sadržaj

[Sadržaj 2](#_Toc56515627)

[1. Problem 1 – Računanje broja π (pi) 3](#_Toc56515628)

[1.1. Tekst problema 3](#_Toc56515629)

[1.2. Delovi koje treba paralelizovati 3](#_Toc56515630)

[1.2.1. Diskusija 3](#_Toc56515631)

[1.2.2. Način paralelizacije 3](#_Toc56515632)

[1.3. Rezultati 3](#_Toc56515633)

[1.3.1. Logovi izvršavanja 3](#_Toc56515634)

[1.3.2. Grafici ubrzanja 4](#_Toc56515635)

[1.3.3. Diskusija dobijenih rezultata 4](#_Toc56515636)

[2. Problem 2 – Računanje broja π (pi) 5](#_Toc56515637)

[2.1. Tekst problema 5](#_Toc56515638)

[2.2. Delovi koje treba paralelizovati 5](#_Toc56515639)

[2.2.1. Diskusija 5](#_Toc56515640)

[2.2.2. Način paralelizacije 5](#_Toc56515641)

[2.3. Rezultati 5](#_Toc56515642)

[2.3.1. Logovi izvršavanja 5](#_Toc56515643)

[2.3.2. Grafici ubrzanja 6](#_Toc56515644)

[2.3.3. Diskusija dobijenih rezultata 6](#_Toc56515645)

[3. Problem 3 – Računanje broja π (pi) 7](#_Toc56515646)

[3.1. Tekst problema 7](#_Toc56515647)

[3.2. Delovi koje treba paralelizovati 7](#_Toc56515648)

[3.2.1. Diskusija 7](#_Toc56515649)

[3.2.2. Način paralelizacije 7](#_Toc56515650)

[3.3. Rezultati 7](#_Toc56515651)

[3.3.1. Logovi izvršavanja 7](#_Toc56515652)

[3.3.2. Grafici ubrzanja 8](#_Toc56515653)

[3.3.3. Diskusija dobijenih rezultata 8](#_Toc56515654)

[4. Problem 4 – *Needleman-Wunsch* algoritam 9](#_Toc56515655)

[4.1. Tekst problema 9](#_Toc56515656)

[4.2. Delovi koje treba paralelizovati 9](#_Toc56515657)

[4.2.1. Diskusija 9](#_Toc56515658)

[4.2.2. Način paralelizacije 9](#_Toc56515659)

[4.3. Rezultati 9](#_Toc56515660)

[4.3.1. Logovi izvršavanja 9](#_Toc56515661)

[4.3.2. Grafici ubrzanja 10](#_Toc56515662)

[4.3.3. Diskusija dobijenih rezultata 10](#_Toc56515663)

[5. Problem 5 – Interakcija čvrstih tela (*n-body*) 11](#_Toc56515664)

[5.1. Tekst problema 11](#_Toc56515665)

[5.2. Delovi koje treba paralelizovati 11](#_Toc56515666)

[5.2.1. Diskusija 11](#_Toc56515667)

[5.2.2. Način paralelizacije 11](#_Toc56515668)

[5.3. Rezultati 11](#_Toc56515669)

[5.3.1. Logovi izvršavanja 11](#_Toc56515670)

[5.3.2. Grafici ubrzanja 12](#_Toc56515671)

[5.3.3. Diskusija dobijenih rezultata 12](#_Toc56515672)

1. Problem 1 – Računanje broja π (pi)
   1. Tekst problema

Paralelizovati program koji izračunava vrednost broja PI korišćenjem formule:

.

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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Jedino mesto u programu koje se može paralelizovati je glavna for petlja koja radi sumu faktora koji čine vrednost broja pi.

* + 1. Način paralelizacije

Paralelizacije je rađena ručno – bez direktiva za podelu posla (*worksharing* direktiva). Ovo je urađeno podelom posla prema id-u niti, svakoj niti je dodeljen određeni chunk posla – ovo je implementacija poznatog SPMD pattern-a paralelnog programiranja. Razmatrana je i podela posla koja je cirkularna, ali ovo jedino može da unese dodatne režijske troškove oko promašaja u procesorkom kešu. U ovom prostom programu za računanje broja pi se to verovatno ne dešava – ali svakako se performanse time ne mogu poboljšati.

* 1. Rezultati

Paralelizacijom ovog programa na više hardverskih niti se dobija ubrzanje koje je prikazano na grafiku (Slika 1) – ovo ubrzanje se vidi u slučaju većeg posla.

* + 1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti.

$ ./z1 1000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000 terms

   Our estimate of pi = 3.14159165358973

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.003196s

Parallel elapsed time: 0.002080s

Test PASSED

$ ./z1 10000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000 terms

   Our estimate of pi = 3.14159255358983

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.040515s

Parallel elapsed time: 0.009042s

Test PASSED

$ ./z1 100000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 100000000 terms

   Our estimate of pi = 3.14159264358988

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.326829s

Parallel elapsed time: 0.080812s

Test PASSED

$ ./z1 1000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000000 terms

   Our estimate of pi = 3.14159265258932

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 3.192847s

Parallel elapsed time: 0.797162s

Test PASSED

$ ./z1 10000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000000 terms

   Our estimate of pi = 3.14159265348821

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 31.837818s

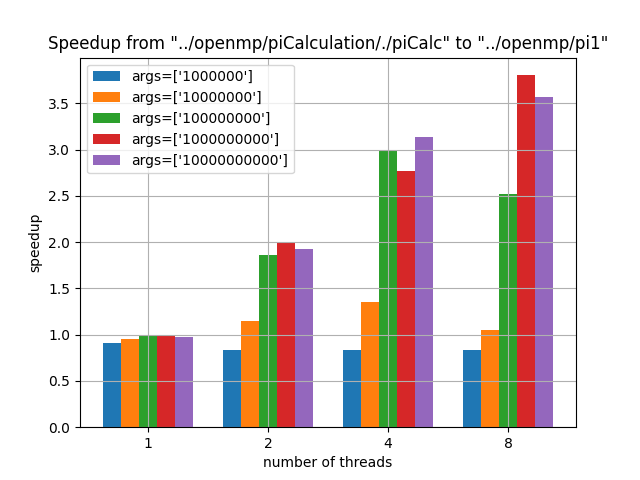
Parallel elapsed time: 7.959584s

Test PASSED

Listing 1. Logovi izvršavanja računanja pi

* + 1. Grafici ubrzanja

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



Slika 1. Grafik zavisnosti ubrzanja u odnosu na broj niti za različite argumente pokretanja

* + 1. Diskusija dobijenih rezultata

Paralelizacija na više niti je donela ubrzanje za posao koji je dovoljno veliki – ovo ubrzanje se najbolje vidi kada imamo više od 1000000 iteracije petlje. Program je embarrassingly parallel i svaka nit dobija određeni chunk iteracija za računanje. Posao svake niti je iste veličine, pa je statička raspodela najbolji način paralelizacije, jer ne unosi dodatne režijske troškove pozivanja schedule operacije. Za pokretanje na jednoj niti se vidi vrlo malo usporenje – ovo je očekivano – to usporenje je uzrokovano režijskim troškovima inicijalizacije OpenMP okruženja.

1. Problem 2 – Računanje broja π (pi)
   1. Tekst problema

Paralelizovati program koji izračunava vrednost broja PI korišćenjem formule:

.

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 koristiti direktive 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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Jedino mesto u programu koje se može paralelizovati je glavna for petlja koja radi sumu faktora koji čine vrednost broja pi.

* + 1. Način paralelizacije

Paralelizacije je rađena putem direktiva za podelu posla (*worksharing* direktiva). Ovo je urađeno ugrađenim konstruktom OpenMP-a (#pragma omp parallel for), svakoj niti je dodeljen određeni chunk posla – ovo je implementacija poznatog SPMD pattern-a paralelnog programiranja. Razmatrana je i dinamčka podela posla nitima, ali ovo jedino može da unese dodatne režijske troškove oko pozivanja scheduler-a.

* 1. Rezultati

Paralelizacijom ovog programa na više hardverskih niti se dobija ubrzanje koje je prikazano na grafiku (Slika 2) – ovo ubrzanje se vidi u slučaju većeg posla.

* + 1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti.

$ ./z2 1000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000 terms

   Our estimate of pi = 3.14159165358973

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.008275s

Parallel elapsed time: 0.005338s

Test PASSED

$ ./z2 10000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000 terms

   Our estimate of pi = 3.14159255358983

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.039251s

Parallel elapsed time: 0.009018s

Test PASSED

$ ./z2 100000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 100000000 terms

   Our estimate of pi = 3.14159264358988

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.324678s

Parallel elapsed time: 0.080689s

Test PASSED

$ ./z2 1000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000000 terms

   Our estimate of pi = 3.14159265258932

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 3.182975s

Parallel elapsed time: 0.797392s

Test PASSED

$ ./z2 10000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000000 terms

   Our estimate of pi = 3.14159265348821

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 31.835786s

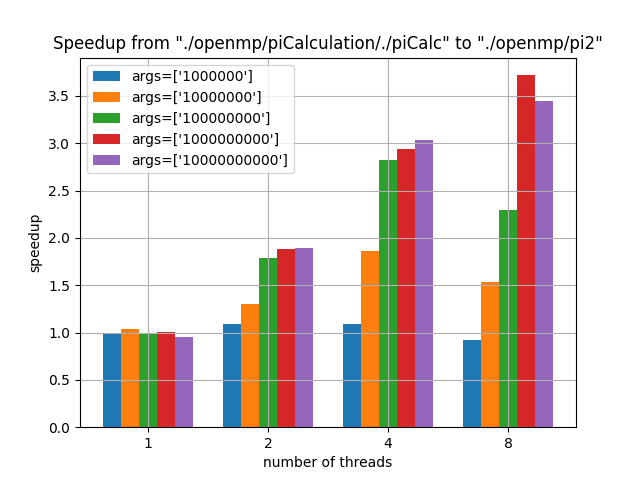
Parallel elapsed time: 7.961063s

Test PASSED

Listing 2. Logovi izvršavanja računanja pi

* + 1. Grafici ubrzanja

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



Slika 2. Grafik zavisnosti ubrzanja u odnosu na broj niti za različite argumente pokretanja

* + 1. Diskusija dobijenih rezultata

Paralelizacija na više niti je donela ubrzanje za posao koji je dovoljno veliki – ovo ubrzanje se najbolje vidi kada imamo više od 1000000 iteracije petlje. Program je embarrassingly parallel i svaka nit dobija određeni chunk iteracija za računanje. Posao svake niti je iste veličine, pa je statička raspodela najbolji način paralelizacije, jer ne unosi dodatne režijske troškove pozivanja schedule operacije. Za pokretanje na jednoj niti se vidi vrlo malo usporenje – ovo je očekivano – to usporenje je uzrokovano režijskim troškovima inicijalizacije OpenMP okruženja.

1. Problem 3 – Računanje broja π (pi)
   1. Tekst problema

Paralelizovati program koji izračunava vrednost broja PI korišćenjem formule:

.

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 koristiti koncept 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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Jedino mesto u programu koje se može paralelizovati je glavna for petlja koja radi sumu faktora koji čine vrednost broja pi.

* + 1. Način paralelizacije

Paralelizacije je rađena putem koncepta poslova (*tasks*). Ovo je urađeno ugrađenim konstruktom OpenMP-a (#pragma omp task), svakoj task-u je dodeljena po jedan chunk iteracija, potom se ti task-ovi raspodeljuju na hardverske niti putem OpenMP okruženja. Razmatrano je i podela posla task-ovima tako da svaki task dobije jednu iteraciju, ali je primećeno da ovo unosi velike režijske troškove za raspoelu task-ova na niti, a da svaki task ima trivijalno mali posao koji radi.

* 1. Rezultati

Paralelizacijom ovog programa na više task-ova koji se dodeljuju hardverskim nitima se dobija ubrzanje koje je prikazano na grafiku (Slika 3) – ovo ubrzanje se vidi u slučaju većeg posla.

* + 1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti.

$ ./z3 1000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000 terms

   Our estimate of pi = 3.14159165358973

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.014481s

Parallel elapsed time: 0.002097s

Test PASSED

$ ./z3 10000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000 terms

   Our estimate of pi = 3.14159255358983

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.042082s

Parallel elapsed time: 0.008904s

Test PASSED

$ ./z3 100000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 100000000 terms

   Our estimate of pi = 3.14159264358988

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 0.326934s

Parallel elapsed time: 0.080753s

Test PASSED

$ ./z3 1000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 1000000000 terms

   Our estimate of pi = 3.14159265258932

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 3.192513s

Parallel elapsed time: 0.797299s

Test PASSED

$ ./z3 10000000000

---------------------Sequential execution---------------------

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

----------------------Parallel execution----------------------

Before for loop, factor = 0.000000.

After for loop, factor = -1.000000.

With n = 10000000000 terms

   Our estimate of pi = 3.14159265348821

   Ref estimate of pi = 3.14159265358979

Sequential elapsed time: 31.839965s

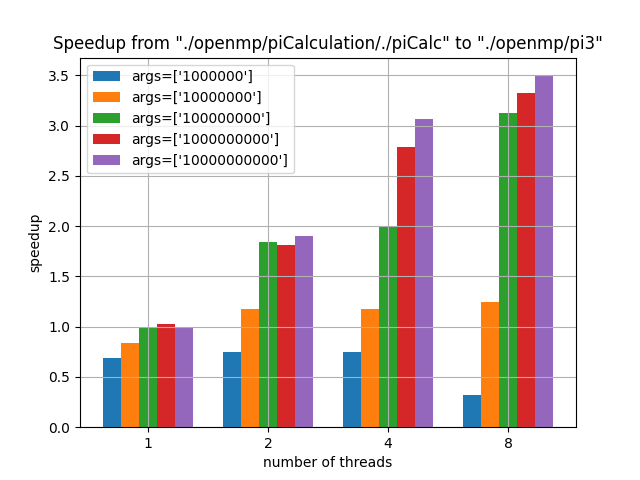
Parallel elapsed time: 7.968897s

Test PASSED

Listing 3. Logovi izvršavanja računanja pi

* + 1. Grafici ubrzanja

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



Slika 3. Grafik zavisnosti ubrzanja u odnosu na broj niti za različite argumente pokretanja

* + 1. Diskusija dobijenih rezultata

Paralelizacija na više task-ova je donela ubrzanje za posao koji je dovoljno veliki – ovo ubrzanje se najbolje vidi kada imamo više od 1000000 iteracije petlje. Program je embarrassingly parallel i svaka nit dobija određeni chunk iteracija za računanje – time što svaka nit dobija nekoliko task-ova. Posao svakog task-a je iste veličine, ali svaki task se dinamički raspodeljuje na hardverske niti, pa imamo dodatne režijske troškove pri pozivanju scheduler-a – zbog ovoga ovakav problem nije optimalno rešavati konstruktom task-ova. Za pokretanje na jednoj niti se vidi usporenje koje je očekivano zbog režijskih troškova inicijalizacije OpenMP okruženja i raspodele task-ova na istu nit.

1. Problem 4 – *Needleman-Wunsch* algoritam
   1. 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 (više o algoritmu na adresi: <https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm>). 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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Paralelizacija je moguća u svim for petljama koje vrše inicijalizaciju – to je urađeno for konstruktom OpenMP okruženja. Usled zavisnosti po podacima preko granica iteracija for petlji koje rade obradu matrice (for petlje su do-across tipa), nemoguće je paralelizovati spoljašnje petlje obrade, slično tome ni obrade dve trougaone matrice koje čine matricu koja se obrađuje se ne mogu raditi u paraleli. Uočeno je da unutrašnje petlje obrade trouganih delova matrice rade iteriranje po dijagonalama matrice, a da svaki element na dijagonali ima nezavisnu obradu od ostalih. Ove dve unutrašnje petlje su zbog toga paralelizovane for konstruktom. Kako TRACEBACK deo programa radi sekvencijalnu pretragu sa zavisnostima od prošle iteracije, ovaj deo programa mora ostati sekvencijalan zbog korektnosti izvršavanja.

* + 1. Način paralelizacije

Paralelizacija je vršena for konstruktima OpenMP okruženja.

* 1. Rezultati

Paralelizacijom ovog programa se dobija ubrzanje koje je prikazano na grafiku (Slika 4) – ovo ubrzanje se najbolje vidi u slučaju većeg posla, iako se može primetiti i na manjoj količini posla.

* + 1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti.

$ ./z4 2048 10

---------------------Sequential execution---------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

----------------------Parallel execution----------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

Sequential elapsed time: 0.110103s

Parallel elapsed time: 0.034839s

Test PASSED

$ ./z4 6144 10

---------------------Sequential execution---------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

----------------------Parallel execution----------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

Sequential elapsed time: 1.001968s

Parallel elapsed time: 0.522392s

Test PASSED

$ ./z4 16384 10

---------------------Sequential execution---------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

----------------------Parallel execution----------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

Sequential elapsed time: 9.237692s

Parallel elapsed time: 3.623918s

Test PASSED

$ ./z4 22528 10

---------------------Sequential execution---------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

----------------------Parallel execution----------------------

Start Needleman-Wunsch

Processing top-left matrix

Processing bottom-right matrix

Sequential elapsed time: 16.767249s

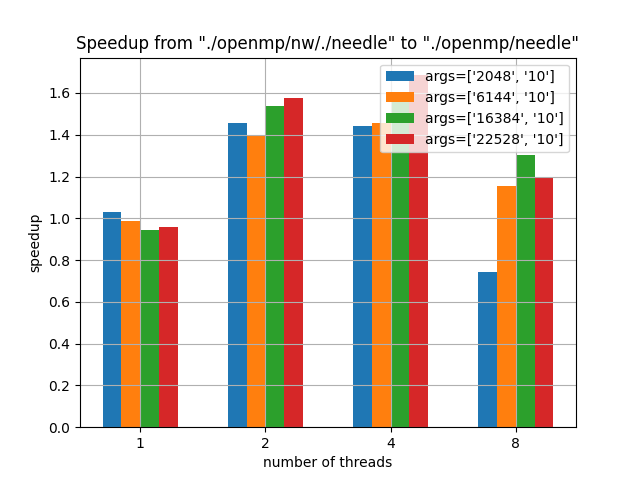
Parallel elapsed time: 7.104467s

Test PASSED

Listing 4. Logovi izvršavanja Needleman-Wunsch algoritma

* + 1. Grafici ubrzanja

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



Slika 4. Grafik zavisnosti ubrzanja u odnosu na broj niti za različite argumente pokretanja

* + 1. Diskusija dobijenih rezultata

Problem sam po sebi nije preterano paralelan pošto koristi dinamičko programiranje koje unosi dosta zavisnoti po podacima preko granica iteracija svih petlji. Vidljivo je da je sa povećanjem posla veće ubrzanje, a primećuje se da za 2 i 4 niti se dobijaju najbolje performanse. Kako ova izvršavanja nisu testirana u izolovanom okruženju (vreme je mereno na običnom PC-u), moguće je da i za 8 niti ubrzanje bude bolje. Iz ovakvih podataka se može zaključiti da za 8 niti ima previše režijskog posla za fork-join, ovo je i očekivano pošto su paralelizovane unutrašnje petlje. Pokušana je i dinamička raspodela posla po nitima, ali u većini slučajeva ovo samo donosi usporenje u izvršavanju zbog dodatnog posla raspoređivanja niti.

1. Problem 5 – Interakcija čvrstih tela (*n-body*)
   1. Tekst problema

Paralelizovati program koji simulira problem interakcije čvrstih tela u dvodimenzionalnom prostoru (*n-body* 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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

TODO: U okviru ove sekcije će biti opisani uočeni delovi koda koje je moguće paralelizovati. Biće diskutovati delovi nad kojima je izvršena paralelizacija. Ukoliko je neki deo moguće paralelizovati, ali to nije učinjeno, navesti razloge.

* + 1. Način paralelizacije

TODO: Ovde treba opisati način paralelizacije i razloge koji su vodili ka odluci. Ukoliko je isprobano nekoliko načina, ovde ih diskutovati.

* 1. Rezultati

TODO: U okviru ove sekcije su izloženi rezultati paralelizacije problema 1.

* + 1. Logovi izvršavanja

Ovde su dati logovi izvršavanja za definisane test primere i različit broj niti.

$ ./z5 100 500 0.01 500 g

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

Elapsed time = 8.872795e-02 seconds

Sequential elapsed time: 0.030873s

Parallel elapsed time: 0.088732s

Test PASSED

$ ./z5 500 500 0.01 500 g

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

Elapsed time = 6.411481e-01 seconds

Sequential elapsed time: 0.672129s

Parallel elapsed time: 0.641153s

Test PASSED

$ ./z5 5000 500 0.01 500 g

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

Elapsed time = 2.794771e+01 seconds

Sequential elapsed time: 54.167515s

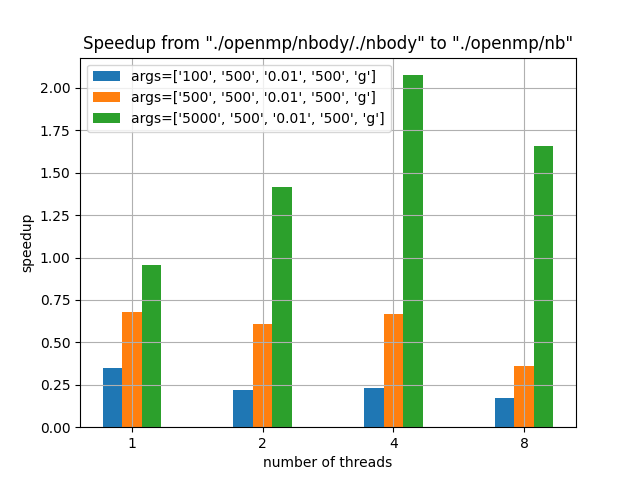
Parallel elapsed time: 27.947755s

Test PASSED

Listing 5. Logovi izvršavanja n-body problema

* + 1. Grafici ubrzanja

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



Slika 5. Grafik zavisnosti ubrzanja u odnosu na broj niti za različite argumente pokretanja

* + 1. Diskusija dobijenih rezultata

TODO: Dobijene rezultate treba izdiskutovati i objasniti sa nekoliko rečenica u odnosu na dobijena ubrzanja (ili usporenja), način paralelizacije, karakteristike problema i sl.