Univerzitet u Beogradu - Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



Domaći zadatak 2 – MPI

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni asistent: | Studenti: |
| doc. dr Marko Mišić | Lazar Cvetković 2016/0127  Strahinja Stefanović 2016/0130 |

Beograd, mart 2020.

Sadržaj

[Sadržaj 2](#_Toc36550726)

[1. Problem 1 – Julia Set (ručna raspodela) 3](#_Toc36550727)

[1.1. Tekst problema 3](#_Toc36550728)

[1.2. Delovi koje treba paralelizovati 3](#_Toc36550729)

[1.2.1. Diskusija 3](#_Toc36550730)

[1.2.2. Način paralelizacije 3](#_Toc36550731)

[1.3. Rezultati 3](#_Toc36550732)

[1.3.1. Logovi izvršavanja 3](#_Toc36550733)

[1.3.2. Grafici ubrzanja 9](#_Toc36550734)

[1.3.3. Diskusija dobijenih rezultata 9](#_Toc36550735)

[2. Problem 2 – Julia Set (worksharing) 10](#_Toc36550736)

[2.1. Tekst problema 10](#_Toc36550737)

[2.2. Delovi koje treba paralelizovati 10](#_Toc36550738)

[2.2.1. Diskusija 10](#_Toc36550739)

[2.2.2. Način paralelizacije 10](#_Toc36550740)

[2.3. Rezultati 10](#_Toc36550741)

[2.3.1. Logovi izvršavanja 10](#_Toc36550742)

[2.3.2. Grafici ubrzanja 16](#_Toc36550743)

[2.3.3. Diskusija dobijenih rezultata 16](#_Toc36550744)

[3. Problem 3 – Julia Set (tasks) 17](#_Toc36550745)

[3.1. Tekst problema 17](#_Toc36550746)

[3.2. Delovi koje treba paralelizovati 17](#_Toc36550747)

[3.2.1. Diskusija 17](#_Toc36550748)

[3.2.2. Način paralelizacije 17](#_Toc36550749)

[3.3. Rezultati 17](#_Toc36550750)

[3.3.1. Logovi izvršavanja 17](#_Toc36550751)

[3.3.2. Grafici ubrzanja 23](#_Toc36550752)

[3.3.3. Diskusija dobijenih rezultata 23](#_Toc36550753)

[4. Problem 4 – Izoštravanje slike 24](#_Toc36550754)

[4.1. Tekst problema 24](#_Toc36550755)

[4.2. Delovi koje treba paralelizovati 24](#_Toc36550756)

[4.2.1. Diskusija 24](#_Toc36550757)

[4.2.2. Način paralelizacije 24](#_Toc36550758)

[4.3. Rezultati 24](#_Toc36550759)

[4.3.1. Logovi izvršavanja 24](#_Toc36550760)

[4.3.2. Grafici ubrzanja 28](#_Toc36550761)

[4.3.3. Diskusija dobijenih rezultata 28](#_Toc36550762)

[5. Problem 5 – MRI Gridding 29](#_Toc36550763)

[5.1. Tekst problema 29](#_Toc36550764)

[5.2. Delovi koje treba paralelizovati 29](#_Toc36550765)

[5.2.1. Diskusija 29](#_Toc36550766)

[5.2.2. Način paralelizacije 29](#_Toc36550767)

[5.3. Rezultati 29](#_Toc36550768)

[5.3.1. Logovi izvršavanja 29](#_Toc36550769)

[5.3.2. Grafici ubrzanja 32](#_Toc36550770)

[5.3.3. Diskusija dobijenih rezultata 32](#_Toc36550771)

1. Problem 1 – Julia Set (podela posla)
   1. Tekst problema

Paralelizovati program koji formira sliku tačaka koje pripadaju Julia skupu tačaka (https://en.wikipedia.org/wiki/Julia\_set). Neka se posmatra skup tačaka *(x, y)* u na pravougaonom domenu *x, y* ∈ [-1,5, 1.5] i neka važi *z = x+yi*. Julia skup je skup tačaka za koji iteracija *z = z2 + c* ne divergira za određene zadate početne uslove. U zadatom programu početni uslov odgovara *c=- 0.8+0.156i*. Ukoliko u bilo kom trenutku važi *1000 < |z|,* smatra se da tačka *z* ne pripada Julia skupu. Program formira sliku u *Targa* (*.tga*) formatu koja se može otvoriti u nekom od namenskih pregledača slika. Program se nalazi u datoteci **julia.c** u arhivi koja je priložena uz ovaj dokument, dok se primeri izlaznih datoteka nalaze u direktorijumu **output**. 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

Moguće je paralelizovati samo glavnu funkciju koja pokreće izračunavanje svih tačaka skupa **julia\_set**, odnosno dvostruku for petlju unutar nje. Funkciju **julia** koja izračunava tačnu vrednost tačke nije moguće paralelizovati zato što postoji zavisnost između susednih iteracija petlje koju ona izvršava.

* + 1. Način paralelizacije

Funkcija **julia\_set** je paralelizovana sa #pragma omp parallel navođenjem koji podaci su privatni, a koji deljeni, a onda je unutar nje izvršena ručna raspodela posla između niti, tako da svaka nit dobije chunkSize deo posla.

* 1. Rezultati
     1. Logovi izvršavanja

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

Input values: 500 500 200

Number of threads: 1

Sequential execution time: 0.037751

Parallel execution time: 0.041152

Test PASSED

Input values: 500 500 500

Number of threads: 1

Sequential execution time: 0.046585

Parallel execution time: 0.052502

Test PASSED

Input values: 500 500 1000

Number of threads: 1

Sequential execution time: 0.048130

Parallel execution time: 0.054510

Test PASSED

Input values: 1000 1000 200

Number of threads: 1

Sequential execution time: 0.151097

Parallel execution time: 0.157974

Test PASSED

Input values: 1000 1000 500

Number of threads: 1

Sequential execution time: 0.186530

Parallel execution time: 0.194000

Test PASSED

Input values: 1000 1000 1000

Number of threads: 1

Sequential execution time: 0.192753

Parallel execution time: 0.201030

Test PASSED

Input values: 2000 1000 200

Number of threads: 1

Sequential execution time: 0.302736

Parallel execution time: 0.312024

Test PASSED

Input values: 2000 1000 500

Number of threads: 1

Sequential execution time: 0.375105

Parallel execution time: 0.382097

Test PASSED

Input values: 2000 1000 1000

Number of threads: 1

Sequential execution time: 0.386589

Parallel execution time: 0.395180

Test PASSED

Input values: 500 500 200

Number of threads: 2

Sequential execution time: 0.037894

Parallel execution time: 0.027957

Test PASSED

Input values: 500 500 500

Number of threads: 2

Sequential execution time: 0.046571

Parallel execution time: 0.028890

Test PASSED

Input values: 500 500 1000

Number of threads: 2

Sequential execution time: 0.048142

Parallel execution time: 0.027483

Test PASSED

Input values: 1000 1000 200

Number of threads: 2

Sequential execution time: 0.151081

Parallel execution time: 0.080597

Test PASSED

Input values: 1000 1000 500

Number of threads: 2

Sequential execution time: 0.187334

Parallel execution time: 0.100474

Test PASSED

Input values: 1000 1000 1000

Number of threads: 2

Sequential execution time: 0.192770

Parallel execution time: 0.099377

Test PASSED

Input values: 2000 1000 200

Number of threads: 2

Sequential execution time: 0.302706

Parallel execution time: 0.160277

Test PASSED

Input values: 2000 1000 500

Number of threads: 2

Sequential execution time: 0.374043

Parallel execution time: 0.196176

Test PASSED

Input values: 2000 1000 1000

Number of threads: 2

Sequential execution time: 0.392636

Parallel execution time: 0.207682

Test PASSED

Input values: 500 500 200

Number of threads: 4

Sequential execution time: 0.037788

Parallel execution time: 0.023378

Test PASSED

Input values: 500 500 500

Number of threads: 4

Sequential execution time: 0.047239

Parallel execution time: 0.026861

Test PASSED

Input values: 500 500 1000

Number of threads: 4

Sequential execution time: 0.048333

Parallel execution time: 0.025617

Test PASSED

Input values: 1000 1000 200

Number of threads: 4

Sequential execution time: 0.151116

Parallel execution time: 0.077034

Test PASSED

Input values: 1000 1000 500

Number of threads: 4

Sequential execution time: 0.191763

Parallel execution time: 0.089074

Test PASSED

Input values: 1000 1000 1000

Number of threads: 4

Sequential execution time: 0.194235

Parallel execution time: 0.094649

Test PASSED

Input values: 2000 1000 200

Number of threads: 4

Sequential execution time: 0.302729

Parallel execution time: 0.154640

Test PASSED

Input values: 2000 1000 500

Number of threads: 4

Sequential execution time: 0.375930

Parallel execution time: 0.175974

Test PASSED

Input values: 2000 1000 1000

Number of threads: 4

Sequential execution time: 0.389033

Parallel execution time: 0.195098

Test PASSED

Input values: 500 500 200

Number of threads: 8

Sequential execution time: 0.046081

Parallel execution time: 0.069714

Test PASSED

Input values: 500 500 500

Number of threads: 8

Sequential execution time: 0.046870

Parallel execution time: 0.020852

Test PASSED

Input values: 500 500 1000

Number of threads: 8

Sequential execution time: 0.054669

Parallel execution time: 0.034979

Test PASSED

Input values: 1000 1000 200

Number of threads: 8

Sequential execution time: 0.152271

Parallel execution time: 0.126921

Test PASSED

Input values: 1000 1000 500

Number of threads: 8

Sequential execution time: 0.188042

Parallel execution time: 0.164486

Test PASSED

Input values: 1000 1000 1000

Number of threads: 8

Sequential execution time: 0.203561

Parallel execution time: 0.158690

Test PASSED

Input values: 2000 1000 200

Number of threads: 8

Sequential execution time: 0.308491

Parallel execution time: 0.263148

Test PASSED

Input values: 2000 1000 500

Number of threads: 8

Sequential execution time: 0.386331

Parallel execution time: 0.202962

Test PASSED

Input values: 2000 1000 1000

Number of threads: 8

Sequential execution time: 0.389125

Parallel execution time: 0.243285

Test PASSED

Listing 1. Izlaz programa

* + 1. Grafici ubrzanja

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

Slika 1. Grafik zavisnosti ubrzanja

* + 1. Diskusija dobijenih rezultata

Primetno je da MPI sa jednim procesom radi sporije, zbog režijskih troškova, dok se postiže zadovoljavajuće ubrzanje pri 2 i 4 procesa. Kada je broj procesa 8, tada očigledno dolazi do prevelikog zagušenja/zasićenja resursa mašine.

1. Problem 2 – Julia Set (manager - worker)
   1. 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]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Moguće je paralelizovati samo glavnu funkciju koja pokreće izračunavanje svih tačaka skupa **julia\_set**, odnosno dvostruku for petlju unutar nje. Funkciju **julia** koja izračunava tačnu vrednost tačke nije moguće paralelizovati zato što postoji zavisnost između susednih iteracija petlje koju ona izvršava.

* + 1. Način paralelizacije

Korišćena je OpenMP **for** direktiva koja vrši paralelizaciju petlji uz direktivu **schedule**(static,1) koja uvodi cikličnu blokovsku raspodelu iteracija.

* 1. Rezultati
     1. Logovi izvršavanja

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

Input values: 500 500 200

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.039732

Parallel execution time: 0.038191

Test PASSED

Input values: 500 500 500

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.047094

Parallel execution time: 0.050788

Test PASSED

Input values: 500 500 1000

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.048543

Parallel execution time: 0.052643

Test PASSED

Input values: 1000 1000 200

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.151518

Parallel execution time: 0.165196

Test PASSED

Input values: 1000 1000 500

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.190191

Parallel execution time: 0.216154

Test PASSED

Input values: 1000 1000 1000

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.193003

Parallel execution time: 0.235339

Test PASSED

Input values: 2000 1000 200

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.303615

Parallel execution time: 0.309439

Test PASSED

Input values: 2000 1000 500

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.374630

Parallel execution time: 0.377203

Test PASSED

Input values: 2000 1000 1000

Number of threads: 2 (1 slave(s))

Sequential execution time: 0.397137

Parallel execution time: 0.390690

Test PASSED

Input values: 500 500 200

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.041163

Parallel execution time: 0.022671

Test PASSED

Input values: 500 500 500

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.046775

Parallel execution time: 0.025520

Test PASSED

Input values: 500 500 1000

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.048447

Parallel execution time: 0.026341

Test PASSED

Input values: 1000 1000 200

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.151816

Parallel execution time: 0.080789

Test PASSED

Input values: 1000 1000 500

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.195126

Parallel execution time: 0.097672

Test PASSED

Input values: 1000 1000 1000

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.193288

Parallel execution time: 0.105237

Test PASSED

Input values: 2000 1000 200

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.303993

Parallel execution time: 0.178079

Test PASSED

Input values: 2000 1000 500

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.375966

Parallel execution time: 0.215465

Test PASSED

Input values: 2000 1000 1000

Number of threads: 3 (2 slave(s))

Sequential execution time: 0.387468

Parallel execution time: 0.240128

Test PASSED

Input values: 500 500 200

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.039493

Parallel execution time: 0.012064

Test PASSED

Input values: 500 500 500

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.048518

Parallel execution time: 0.017369

Test PASSED

Input values: 500 500 1000

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.051927

Parallel execution time: 0.018529

Test PASSED

Input values: 1000 1000 200

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.153650

Parallel execution time: 0.045462

Test PASSED

Input values: 1000 1000 500

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.186805

Parallel execution time: 0.052683

Test PASSED

Input values: 1000 1000 1000

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.193924

Parallel execution time: 0.051962

Test PASSED

Input values: 2000 1000 200

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.303315

Parallel execution time: 0.084746

Test PASSED

Input values: 2000 1000 500

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.375097

Parallel execution time: 0.114497

Test PASSED

Input values: 2000 1000 1000

Number of threads: 5 (4 slave(s))

Sequential execution time: 0.387436

Parallel execution time: 0.109942

Test PASSED

Input values: 500 500 200

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.038145

Parallel execution time: 0.006882

Test PASSED

Input values: 500 500 500

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.046957

Parallel execution time: 0.009891

Test PASSED

Input values: 500 500 1000

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.048482

Parallel execution time: 0.008602

Test PASSED

Input values: 1000 1000 200

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.152162

Parallel execution time: 0.029565

Test PASSED

Input values: 1000 1000 500

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.187066

Parallel execution time: 0.033438

Test PASSED

Input values: 1000 1000 1000

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.194020

Parallel execution time: 0.033587

Test PASSED

Input values: 2000 1000 200

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.304903

Parallel execution time: 0.053050

Test PASSED

Input values: 2000 1000 500

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.376476

Parallel execution time: 0.083756

Test PASSED

Input values: 2000 1000 1000

Number of threads: 9 (8 slave(s))

Sequential execution time: 0.388858

Parallel execution time: 0.069647

Test PASSED

Listing 2. Izlaz programa

* + 1. Grafici ubrzanja

Slika 2. Grafik zavisnosti ubrzanja

* + 1. Diskusija dobijenih rezultata

Paralelizacija gde se koristi 1 master i 1 slave je sporija zbog režijskih troškova MPI biblioteke. Kako se povećava broj slave procesa, program se izvršava brže i brže, a što se vidi na grafiku.

1. Problem 3 – Izoštravanje slike

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

* 1. Tekst problema

Paralelizovati program koji izoštrava zadatu sliku u *Portable Graymap Format* (PGM) formatu. PGM format se može otvoriti u nekom od namenskih pregledača slika ili *online* na adresi http://paulcuth.me.uk/netpbm-viewer/. Program se nalazi u direktorijumu **sharpen** u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke **sharpen**.**c**, **dosharpen**.**c** i **filter.c**. 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

Izvršena je paralelizacija četiri ugnježdene petlje koje izračunavaju konvoluciju, kao i sledeće dvostruke petlje koja taj dobijeni rezultat koristi za izoštravanje slike.

* + 1. Način paralelizacije

Korišćena je direktiva #pragma omp parallel for uz korišćenje odredbe **collapse(2)** kako bi se sažimanjem savršeno ugneždenih pravougaonih petlji postiglo još veće ubrzanje.

* 1. Rezultati
     1. Logovi izvršavanja

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

Input

Listing 3. Ispis programa

* + 1. Grafici ubrzanja

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

Slika 3. Grafik zavisnosti ubrzanja

* + 1. Diskusija dobijenih rezultata

Primetno je značajno ubrzanje, ali ono je u mnogome i posledica dodavanja funkcije **makeFilterMatrix** koja postupkom memoizacije samo jednom računa filtersku matricu, za razliku od sekvencijalnog izvršavanja koja u svakoj iteraciji to radi. Takođe, može se primetiti da ne postoji neki veliki dobit u performansama ukoliko se broj niti poveća sa 4 na 8.

1. Problem 4 – MRI Gridding

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 4.

* 1. Tekst problema

Paralelizovati program koji vrši mapiranje neuniformnih podataka u 3D prostoru na regularnu mrežu u 3D prostoru. Svaka tačka iz neuniformnog 3D prostora doprinosi susednim tačkama u regularnoj mreži u skladu sa *Kaiser-Bessel* funkcijom za određivanje rastojanja. Program se nalazi u direktorijumu **mri-gridding** u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke **main**.**c** i **CPU\_kernels.c**. Analizirati dati kod i obratiti pažnju na način generisanja vrednosti tačaka u regularnoj mreži. 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. Ulazni test primeri se nalaze u direktorijumu **data**. Verifikaciju paralelizovanog rešenja vršiti nad nizovima **gridData** i **sampleDensity** iz glavnog programa. Način pokretanja programa se nalazi u datoteci **run**. [1, N]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Paralelizovana je funkcija **gridding\_Gold** koju ima i najviše smisla paralelizovati pošto su u njoj izvršava sav potreban račun.

* + 1. Način paralelizacije

Korišćena je direktiva #pragma omp parallel for uz odredbu **schedule**(dynamic, 5000) koja vrši raspodelu posla u pakete veličine 5000, a niti te pakete uzimaju u FIFO redosledu. Deljeni podaci koje je potrebno zaštiti sinhorinizacijonim primitivama su **gridData** i **sampleDensity** i tu svrhu korišćena je direktiva #pragma omp atomic koja daje najbolje performanse u poređenju sa bravama i kritičnim sekcijama.

* 1. Rezultati
     1. Logovi izvršavanja

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

Number of threads: 1

Sequential execution time: 4.447496

Parallel execution time: 4.503017

TEST PASSED - gridData

TEST PASSED - sampleDensity

Number of threads: 2

Sequential execution time: 4.477351

Parallel execution time: 2.477347

TEST PASSED - gridData

TEST PASSED - sampleDensity

Number of threads: 4

Sequential execution time: 4.455474

Parallel execution time: 1.481890

TEST PASSED - gridData

TEST PASSED - sampleDensity

Number of threads: 8

Sequential execution time: 4.483704

Parallel execution time: 1.270220

TEST PASSED - gridData

TEST PASSED - sampleDensity

Listing 4. Ispis programa

* + 1. Grafici ubrzanja

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

Slika 5. Grafik zavisnosti ubrzanja

* + 1. Diskusija dobijenih rezultata

Režijski troškovi MPI biblioteke dolaze do izražaja pri postavci od jednog procesa, inače se postiže očekivano i značajno ubrzanje i do četiri puta u odnosu na sekvencijalno izvršavanje.