UNIVERZITET U BEOGRADU - ELEKTROTEHNIČKI FAKULTET

MULTIPROCESORKI SISTEMI (13S114MUPS, 13E114MUPS)

DOMAĆI ZADATAK 1 – OPENMP

Izveštaj o urađenom domaćem zadatku

Predmetni asistent:

doc. dr Marko Mišić

Kandidati:

Marija Kostić 2015/0096

Stefan Milanović 2015/0361

Beograd, novembar 2018.

SADRŽAJ

1. PROBLEM 1 - SGEMM

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 1.

1.1. Tekst problema

Paralelizovati program koji vrši jednostavno generalizovano množenje matrica u jednostrukoj preciznosti *Single precision floating General Matrix Multiply* (SGEMM). SGEMM operacije je definisana sledećom formom:

$$C \leftarrow \alpha \cdot A \cdot B + \beta \cdot C$$

Program se nalazi u datoteci sgemm.cc 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

Kod koji rešava ovaj problem se može razbiti na tri velike celine – najpre se vrši učitavanje ulaznih matrica iz dva tekstualna fajla, vrši se obrada ulaznih podataka u vidu kreiranja izlazne matrice koja predstavlja proizvod unetih matrica, i na samom kraju se vrši ispis izlazne matrice u tekstualni fajl.

Prvu i treću celinu nije moguće paralelizovati – ovi delovi koda predstavljaju rad sa I/O sistemom koji se mora obaviti isključivo unutar jedne niti. U teoriji bi bilo moguće uz pomoć više niti obraditi jedan ulazni fajl tako što bi se fajl najpre memorijski mapirao, a nakon toga bi se vršilo raspoređivanje delova mapiranog fajla različitim nitima u cilju paralelne obrade. Ovo rešenje daleko komplikuje kod, a zbog neophodnog *overhead*-a i zauzeća memorije može dovesti i do usporenja.

Jedini deo koda nad kojim se može vršiti paralelizacija jeste deo koda koji radi obradu, tj. funkcija *void basicSgemm(args)*. U skladu sa postavkom zadatka, paralelizacija će se vršiti na tri načina koji će zasebno biti diskutovati i prezentovani na graficima i tabelama.

1.2.2. Način paralelizacije

Zadatak 1

Rešenje ovog zadatka nalazi se u datoteci *dz1z1.cpp*. U ovom zadatku neophodno je ručno rasporediti iteracije petlje koja se paralelizuje.

Ovo je urađeno tako što je glavna *for* petlja koja se nalazi u *basicSgemm()* funkciji podeljena na više delova, pri čemu svaki deo obrađuje jedna nit. Uvedena je promenljiva *chunk* koja predstavlja veličinu dela glavne *for* petlje i ova vrednost je ista za sve niti (osim eventualno za poslednju nit), pa se iz tog razloga ona može izračunati samo jednom. Ovo je omogućeno *single* regionom na početku *parallel* regiona programa koja obuhvata glavnu *for* petlju. Pošto se ovaj region nalazi na samom početku *parallel* regiona, obezbeđeno je da je vrednost *chunk* definisana za sve niti (pošto je promenljiva *shared*) u trenutku kada one dolaze do glavne *for* petlje koja se paralelizuje. Kod iz *single* regiona nije bilo moguće staviti van *parallel regiona*, jer u tom slučaju poziv funkcije *omp_get_num_threads()* vraća 1, a ne predviđen broj niti.

Nakon ovoga, definišu se vrednosti *start* i *end* koje su različite za svaku nit i predstavljaju granice obrade glavne *for* petlje (jedna nit obrađuje vrednosti od 0 do *chunk – 1*, druga od *chunk* do (*2 * chunk) – 1*, itd.).

Jedino što se sada menja jeste samo telo glavne *for* petlje, koje sad iterira od *mm = start* sve dokle važi *mm < end*, čime se postiže ručna paralelizacija petlje.

Zadatak 2

Rešenje ovog zadatka nalazi se u datoteci *dz1z2.cpp*. U ovom zadatku neophodno je paralelizovati program koristeći *worksharing* direktive.

U ovom zadatku je kod dosta jednostavniji i malo se menja u odnosu na sekvencijalnu implementaciju, zato što *omp* biblioteka vrši raspoređivanje petlje na više niti. Uočena je savršeno ugneždena pravougana petlja koja se sastoji od 2 petlje, pa je njih moguće paralelizovati *collapse* odredbom u okviru *parallel for* regiona. Ovo je ujedno sve što je neophodno odraditi kako bi se izvršila paralelizacija u ovom zadatku. Za razliku od prethodnog zadatka, ovde se sam kod petlji ne menja uopšte.

Zadatak 3

Rešenje ovog zadatka nalazi se u datoteci *dz1z3.cpp*. U ovom zadatku neophodno je paralelizovati program koristeći koncept zadataka (engl. *task-ova*).

Ovaj zadatak takođe ima jednostavniji kod u odnosu na prvi zadatak. Neophodno je staviti glavnu *for* petlju u jedan paralelni region, a zatim izdvojiti deo unutrašnjeg koda kao *task* koji će se izvršiti kasnije. Samim tim se ovaj zadatak može rešiti na 4 načina: Moguće je izvršiti *coarse grain* ili *fine grain* paralelizaciju tako što task obuhvata sve što se nalazi unutar glavne *for* petlje, ili tako što *task* obuhvata samo kod unutar druge *for* petlje. Na ovaj način se posao raspoređuje tako da jedan task predstavlja računanje vrednosti jednog reda, odnosno jedne ćelije krajnje matrice. Pored ovoga, moguće je postaviti samo jednu nit koja će kreirati *task*-ove koje će ostale niti izvršavati, dok je druga mogućnost da sve niti kreiraju *task*-ove, a onda ih i izvršavaju.

Svi ovi načini su isprobani i pokazalo se da najbolje performanse pruža varijanta *coarse grain* paralelizacije pri čemu sve *task*-ove pravi jedna nit. Nakon pokretanja test primera za sve varijante više puta pokazalo se da je jako mala razlika u performansama kada se obrađuje mali ulazni skup podataka. Za veći ulazni skup podataka dobijeno je da su *coarse grain* implementacije u svim merenjima bile brže od *fine grain* implementacija. Nakon posmatranja samo *coarse grain* implementacija i njihovih performansi nakon obrade velikog ulaznog skupa podataka, uočilo se da *coarse grain* implementacija kod koje samo jedna nit kreira *task*-ove konstantno daje bolje rezultate od *coarse grain* implementacije kod koje sve niti kreiraju *task*-ove.

Rezultati koji su dobijeni koristeći ovu varijantu predstavljeni su u sekciji **Rezultati** i prikazani su na graficima performansi.

1.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1 koristeći rešenja sva tri zadatka.

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.

Listing 1. Zadatak 1 – 1 nit

```
Opening file:data/small/input/matrix1.txt
Matrix dimension: 128x96
Opening file:data/small/input/matrix2t.txt
Matrix dimension: 160x96
Opening file:result_small_par.txt for write.
Matrix dimension: 128x160
```

**********DZ1Z1**********

Elapsed time - SEQ: 0.00146536.

Elapsed time - PAR(1): 0.00146888.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z1**********

Elapsed time - SEQ: 11.8154.

Elapsed time - PAR(1): 11.8128.

TEST PASSED

****

Listing 2. Zadatak 1 – 2 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z1**********

Elapsed time - SEQ: 0.00139774.

Elapsed time - PAR(2): 0.000756554.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z1**********

Elapsed time - SEQ: 11.8395.

Elapsed time - PAR(2): 5.93361.

TEST PASSED

```
**********************************************
```

Listing 3. Zadatak 1 – 4 niti

```
Opening file:data/small/input/matrix1.txt
Matrix dimension: 128x96
Opening file:data/small/input/matrix2t.txt
Matrix dimension: 160x96
Opening file:result_small_par.txt for write.
Matrix dimension: 128x160
**********DZ1Z1**********
Elapsed time - SEQ: 0.00149391.
Elapsed time - PAR(4): 0.00046414.
TEST PASSED
**********************************************
Opening file:data/medium/input/matrix1.txt
Matrix dimension: 1024x992
Opening file:data/medium/input/matrix2t.txt
Matrix dimension: 1056x992
Opening file:result_medium_par.txt for write.
Matrix dimension: 1024x1056
**********DZ1Z1**********
Elapsed time - SEQ: 11.836.
Elapsed time - PAR(4): 2.9717.
TEST PASSED
**********************************************
```

Listing 4. Zadatak 1 – 8 niti

```
Opening file:data/small/input/matrix1.txt
Matrix dimension: 128x96
Opening file:data/small/input/matrix2t.txt
Matrix dimension: 160x96
Opening file:result_small_par.txt for write.
Matrix dimension: 128x160
**********DZ1Z1**********
Elapsed time - SEQ: 0.00147804.
Elapsed time - PAR(8): 0.00085655.
TEST PASSED
**********************************************
Opening file:data/medium/input/matrix1.txt
Matrix dimension: 1024x992
```

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z1**********

Elapsed time - SEQ: 11.7016.

Elapsed time - PAR(8): 3.26623.

TEST PASSED

****

Listing 5. Zadatak 2 – 1 nit

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z2**********

Elapsed time - SEQ: 0.00144419.

Elapsed time - PAR(1): 0.00146191.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z2**********

Elapsed time - SEQ: 11.8102.

Elapsed time - PAR(1): 11.8144.

TEST PASSED

****

Listing 6. Zadatak 2 – 2 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z2**********

Elapsed time - SEQ: 0.00143754.

Elapsed time - PAR(2): 0.000787277.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z2**********

Elapsed time - SEQ: 11.8193.

Elapsed time - PAR(2): 5.91367.

TEST PASSED

****

Listing 7. Zadatak 2 – 4 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z2**********

Elapsed time - SEQ: 0.00146809.

Elapsed time - PAR(4): 0.000457916.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z2**********

Elapsed time - SEQ: 11.878.

Elapsed time - PAR(4): 2.96931.

TEST PASSED

****

Listing 8. Zadatak 2 – 8 niti

```
Opening file:data/small/input/matrix1.txt
Matrix dimension: 128x96
Opening file:data/small/input/matrix2t.txt
Matrix dimension: 160x96
Opening file:result_small_par.txt for write.
Matrix dimension: 128x160
**********DZ1Z2**********
Elapsed time - SEQ: 0.0014584.
Elapsed time - PAR(8): 0.00161207.
TEST PASSED
**********
Opening file:data/medium/input/matrix1.txt
Matrix dimension: 1024x992
Opening file:data/medium/input/matrix2t.txt
Matrix dimension: 1056x992
Opening file:result_medium_par.txt for write.
Matrix dimension: 1024x1056
**********DZ1Z2**********
Elapsed time - SEQ: 11.8274.
Elapsed time - PAR(8): 2.95951.
TEST PASSED
**********
```

Listing 9. Zadatak 3 – 1 nit

```
Opening file:data/small/input/matrix1.txt
Matrix dimension: 128x96
Opening file:data/small/input/matrix2t.txt
Matrix dimension: 160x96
Opening file:result_small_par.txt for write.
Matrix dimension: 128x160
**********DZ1Z3**********
Elapsed time - SEQ: 0.00147348.
Elapsed time - PAR(1): 0.00145707.
TEST PASSED
**********
Opening file:data/medium/input/matrix1.txt
```

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z3**********

Elapsed time - SEQ: 11.7824.

Elapsed time - PAR(1): 11.799.

TEST PASSED

****

Listing 10. Zadatak 3 – 2 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z3**********

Elapsed time - SEQ: 0.00147057.

Elapsed time - PAR(2): 0.000871783.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z3**********

Elapsed time - SEQ: 11.7924.

Elapsed time - PAR(2): 5.90932.

TEST PASSED

****

Listing 11. Zadatak 3 – 4 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z3**********

Elapsed time - SEQ: 0.00146563.

Elapsed time - PAR(4): 0.000495821.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z3**********

Elapsed time - SEQ: 11.8135.

Elapsed time - PAR(4): 2.96233.

TEST PASSED

****

Listing 12. Zadatak 3 – 8 niti

Opening file:data/small/input/matrix1.txt

Matrix dimension: 128x96

Opening file:data/small/input/matrix2t.txt

Matrix dimension: 160x96

Opening file:result_small_par.txt for write.

Matrix dimension: 128x160

**********DZ1Z3**********

Elapsed time - SEQ: 0.0015428.

Elapsed time - PAR(8): 0.0019373.

TEST PASSED

****

Opening file:data/medium/input/matrix1.txt

Matrix dimension: 1024x992

Opening file:data/medium/input/matrix2t.txt

Matrix dimension: 1056x992

Opening file:result_medium_par.txt for write.

Matrix dimension: 1024x1056

**********DZ1Z3**********

Elapsed time - SEQ: 11.7888.

Elapsed time - PAR(8): 2.96397.

TEST PASSED

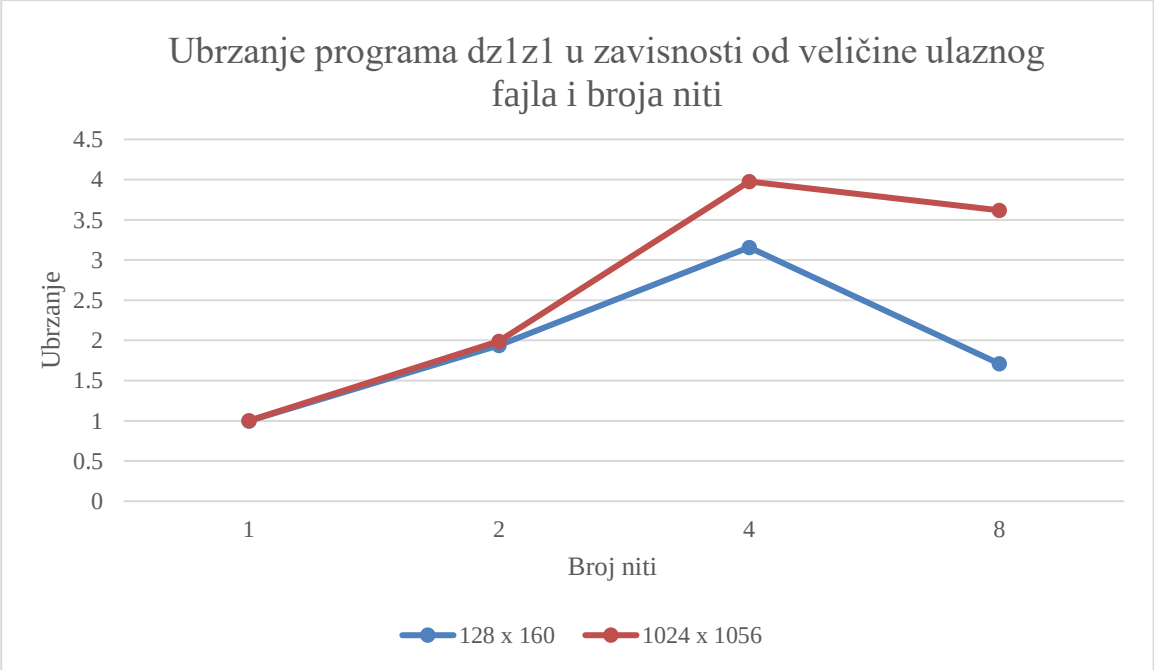
****

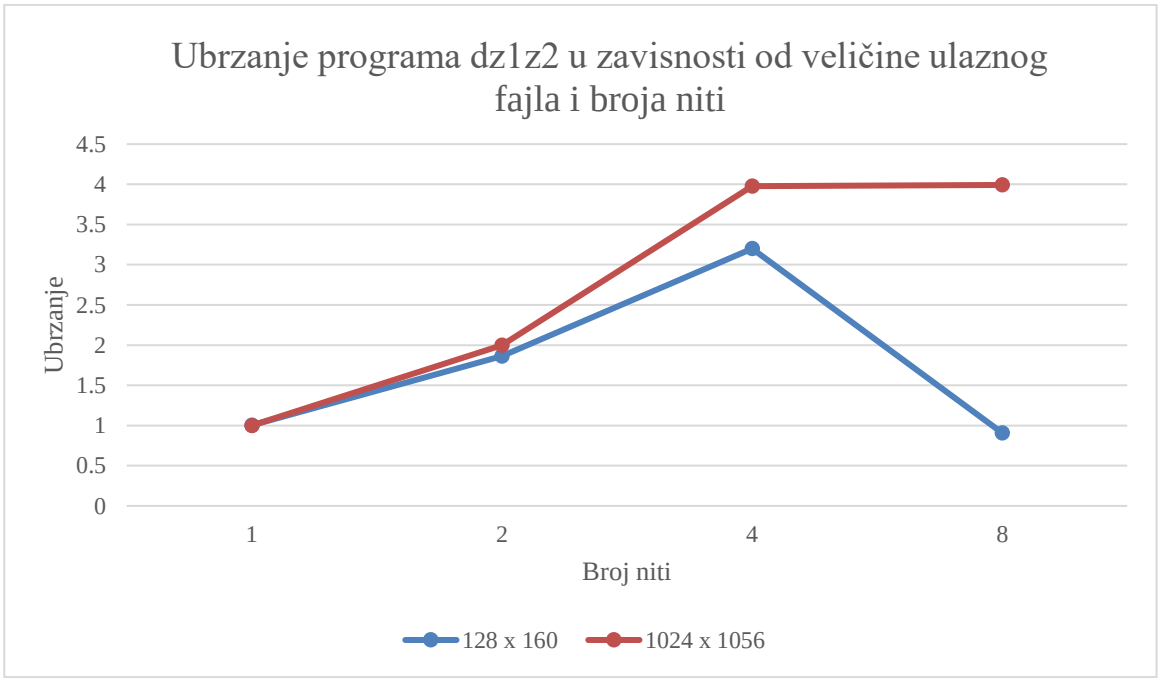
1.3.2. Tabela trajanja programa

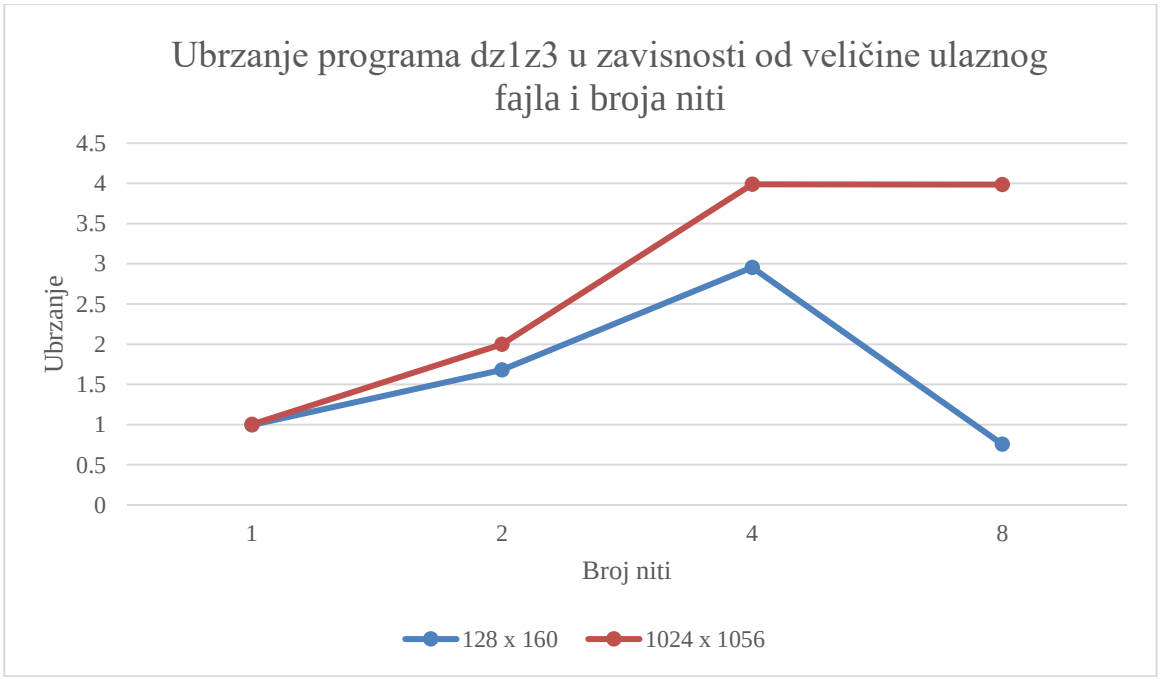
	small input		medium input	
	vreme	**ubrzanje**	**vreme**	**ubrzanje**
Sekvencijalno izvršavanje	0,00146536		11,81540000	
Zadatak 1 – 1 nit	0,00146888	0,99760362	11,81280000	1,00022010
Zadatak 1 – 2 niti	0,00075655	1,93688752	5,93361000	1,99126670
Zadatak 1 – 4 niti	0,00046414	3,15715086	2,97170000	3,97597335
Zadatak 1 – 8 niti	0,00085655	1,71076995	3,26623000	3,61744274
Zadatak 2 – 1 nit	0,00146191	1,00235993	11,81440000	1,00008464
Zadatak 2 – 2 niti	0,00078728	1,86130168	5,91367000	1,99798095
Zadatak 2 – 4 niti	0,00045792	3,20006289	2,96931000	3,97917361
Zadatak 2 – 8 niti	0,00161207	0,90899279	2,95951000	3,99235008
Zadatak 3 – 1 nit	0,00147070	0,99636908	11,79900000	1,00138995
Zadatak 3 – 2 niti	0,00087178	1,68087701	5,90932000	1,99945171
Zadatak 3 – 4 niti	0,00049582	2,95542141	2,96233000	3,98854955
Zadatak 3 – 8 niti	0,00193730	0,75639292	2,96397000	3,98634264

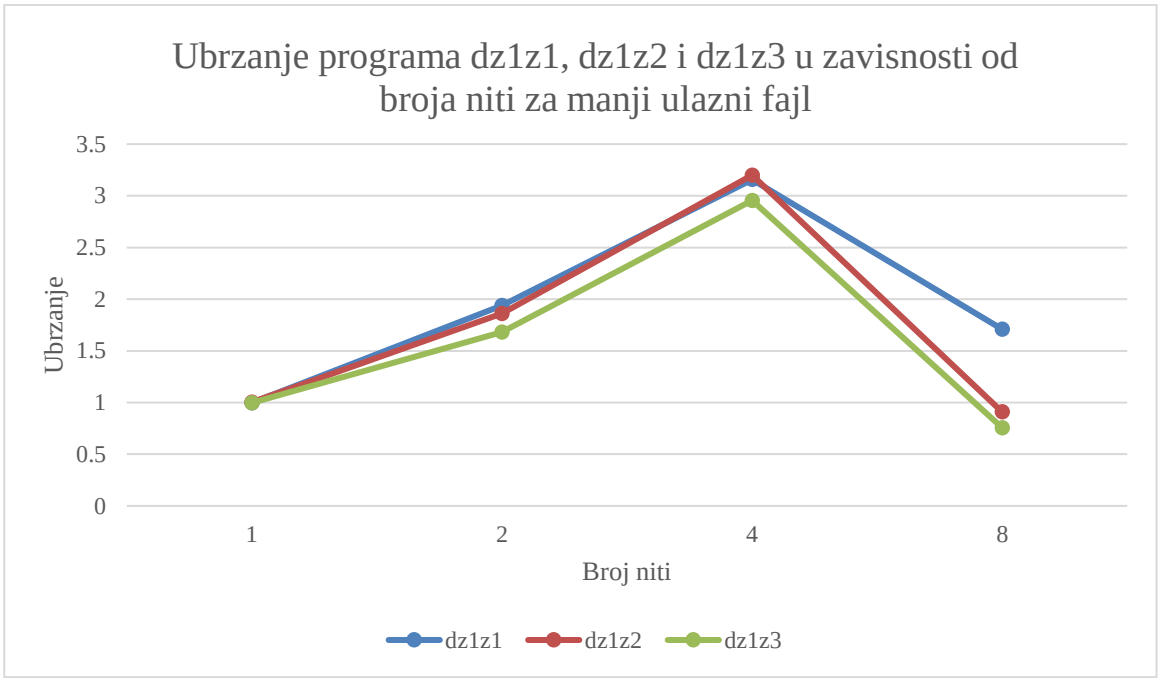
1.3.3. Grafici ubrzanja

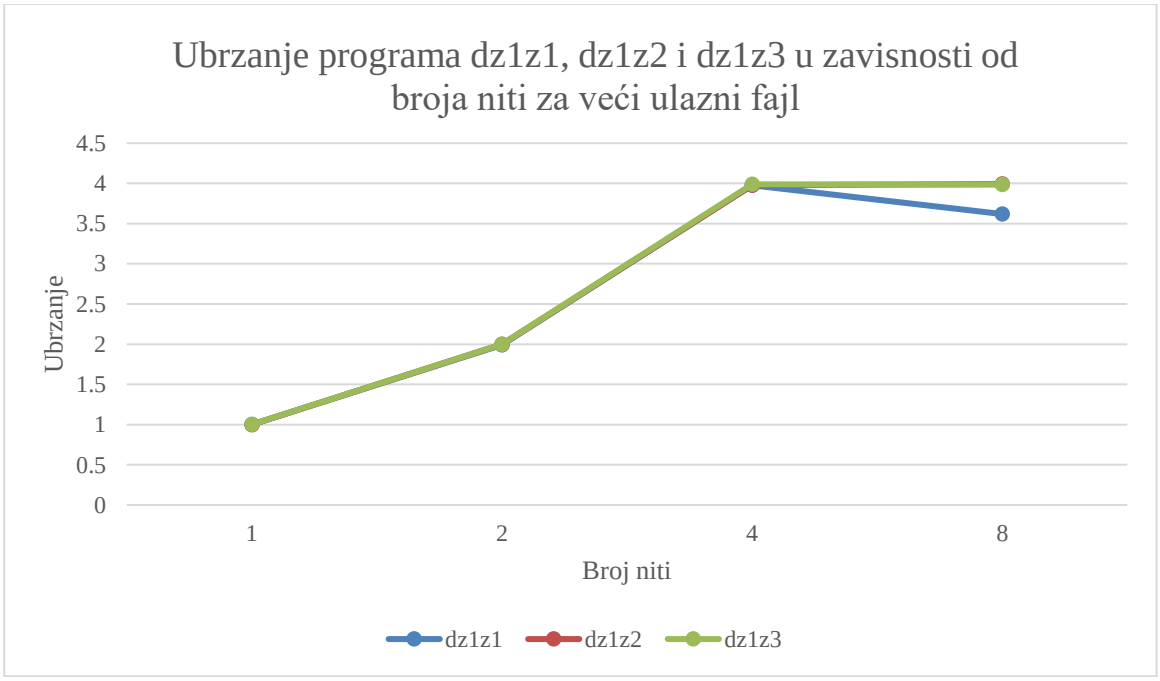
U okviru ove sekcije su dati grafici ubrzanja paralelnih implementacija u odnosu na sekvencijalnu implementaciju. Ubrzanje na svim narednim graficima predstavljeno je kao odnos trajanja sekvencijalne implementacije programa i trajanja paralelizovane implementacije.




Ubrzanje programa dz1z2 u zavisnosti od veličine ulaznog fajla i broja niti
Ubrzanje
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
4
8
Broj niti
128 x 160
1024 x 1056


Ubrzanje programa dz1z3 u zavisnosti od veličine ulaznog fajla i broja niti
Ubrzanje
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
4
8
Broj niti
128 x 160
1024 x 1056


Ubrzanje programa dz1z1, dz1z2 i dz1z3 u zavisnosti od broja niti za manji ulazni fajl
Ubrzanje
3.5
3
2.5
2
1.5
1
0.5
0
1
2
4
8
Broj niti
dz1z1
dz1z2
dz1z3


Ubrzanje programa dz1z1, dz1z2 i dz1z3 u zavisnosti od broja niti za veći ulazni fajl
Ubrzanje
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
4
8
Broj niti
dz1z1
dz1z2
dz1z3

1.3.4. Diskusija dobijenih rezultata

Rezultati se mogu posmatrati tako da se zasebno gledaju izmerene performanse kada se na ulaz sistema dovedu matrice malih dimenzije i kada se na ulaz sistema dovedu matrice većih sistema.

Što se manjeg ulaznog fajla tiče, iz dobijenih rezultata se može videti da se najveće ubrzanje prilikom paralelizacije dobija za 4 niti. Za 8 niti performanse paralelizovanog koda drastično opadaju, a u nekim slučajevima se dobija i usporenje u odnosu na sekvencijalnu implementaciju. Ovakvi rezultati imaju smisla zato što su *overhead*-i paralelizacije, kreiranja i pokretanja tolikog broja niti dosta veliki i nisu opravdani nad ovako malim skupom ulaznih podataka.

Zato se kao rezultat poređenja performansi sekvencijalne i paralelne implementacije kod većeg fajla dobija malo drugačiji graf. Kao i kod manjeg fajla, vidi se poboljšanje performansi za 2 i 4 niti, dok korišćenje 8 niti daje jako slične rezultate kao i korišćenje 4 niti. Dakle, približno se isto ubrzanje dobija bilo da koristimo 4 niti ili 8 niti za ulazni fajl ove veličine. U slučaju ručne paralelizacije dobijeno je ubrzanje koje je za oko *0.5* manje od ubrzanja koje se dobilo koristeći kod koji je paralelizovan preko *worksharing* direktiva i preko *task*-ova.

Kao zaključak, za obe veličine ulaznih fajlova korišćenje 8 niti umesto 4 nije dovelo do značajnog poboljšanja performansi. U slučaju korišćenja većih ulaznih fajlova dobijene su slične performanse kao za 4 niti, dok je u slučaju manjeg ulaznog fajlova korišćenje 8 niti zapravo dovelo do usporenja.

2. PROBLEM 2 - JACOBI

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

2.1. Tekst problema

Paralelizovati program koji vrši jednostavno generalizovano množenje matrica u jednostrukoj Paralelizovati program koji rešava sistem linearnih jednačina $A * x = b$ Jakobijevim metodom. Kod koji treba paralelizovati se nalazi u datoteci *jacobi.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].

2.2. Delovi koje treba paralelizovati

2.2.1. Diskusija

U ovom zadatku moguće je paralelizovati prve dve *for* petlje koje služe za inicijalizaciju, a nakon toga i sav kod koji se nalazi unutar glavne *for* petlje koja inkrementira vrednost promenljive *it* od 0 do $m - 1$ i u kojoj se vrši obrada. Ovu petlju nije moguće paralelizovati zato što postoji zavisnost po podacima – u narednoj iteraciji ove petlje neophodno je u nizu *x* imati vrednosti koje se računaju unutar same petlje, a te izračunate vrednosti se i koriste u poslednjem delu koda kada se računa vrednost promenljive *r*.

Samim tim, moguće je paralelizovati kod za inicijalizaciju i kod unutar glavne *for* petlje.

Kod na samom početku koji vrši alokaciju i dealokaciju prostora, kao i ispise, nema potrebe paralelizovati.

2.2.2. Način paralelizacije

Najpre je moguće malo modifikovati priloženi sekvencijalni kod kako bi se smanjio broj nepotrebnih iteracija petlji. Primećuje se da prve dve petlje obe iteriraju do *n* i i vrše inicijalizaciju odgovarajućih nizova na 0 (uz izutetak poslednjeg elementa niza *b*). Ove dve petlje se mogu spojiti u jednu, i tada je moguće izvršiti paralelizaciju koja će dati bolje rezultate zato što i same niti imaju više posla (čime su opravdani režijski troškovi njihovog kreiranja i pokretanja).

Na sličan način se mogu i spojiti petlje označene komentarima kao *Jacobi update* i *Difference*. Prva *for* petlja od navedenih računa vrednosti za sve elemente niza *xnew* i popunjava ih, pri čemu

nema nikakvih zavisnosti jer se iz niza *x* samo vrši čitanje, pa je moguće da više niti rade operaciju čitanja bez potrebe za sinhronizacijom. Petlja označena komentarom *Difference* samo računa vrednost promenljive *d* po zavisnosti od elemenata u nizovima *x* i *xnew*, pa se ovo može raditi kako se *xnew* računa (ovo omogućava spajanje ove dve petlje). Pošto će više niti raditi ovaj posao, svaka može pamtiti svoju vrednost za *d* i na kraju se može izvršiti redukcija sabiranjem ovih međurezultata (ovo omogućava paralelizaciju petlji).

Petlja koja radi prepisivanje niza *xnew* u niz *x* se može paralelizovati radi ubrzanja kod velike veličine niza.

Naredna petlja označena sa *Residual* takođe predstavlja petlju koja se može paralelizovati uz korišćenje redukcije za promenljivu *r*. Na kraju paralelne *for* petlje vrši se sabiranje svih međurezultata iz svih niti kako bi se dobila konačna vrednost za *r*.

Isprobano je rešenje koje vrši paralelizaciju programa koristeći *task*-ove, ali se ovo rešenje pokazalo dosta sporijim od paralelizacije pomoću *worksharing* direktiva, pa je kao rezultat domaćeg zadatka predat kod koji ne koristi *task*-ove za paralelizaciju.

2.3. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 2 koristeći rešenje koja paralelizuje kod pomoću *worksharing* direktiva.

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.

Listing 1. Zadatak 4 – 1 nit

```
**********DZ1Z4**********
Elapsed time - SEQ: 0.0140979.
Elapsed time - PAR(1): 0.0108403.
TEST PASSED
**********
**********DZ1Z4**********
Elapsed time - SEQ: 0.0322217.
Elapsed time - PAR(1): 0.0349269.
TEST PASSED
**********
```

**********DZ1Z4**********

Elapsed time - SEQ: 0.980146.

Elapsed time - PAR(1): 0.885137.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 4.14646.

Elapsed time - PAR(1): 3.84104.

TEST PASSED

****

Listing 2. Zadatak 4 – 2 niti

**********DZ1Z4**********

Elapsed time - SEQ: 0.0135903.

Elapsed time - PAR(2): 0.0100558.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.0308372.

Elapsed time - PAR(2): 0.0277511.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.943135.

Elapsed time - PAR(2): 0.579109.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 4.71614.

Elapsed time - PAR(2): 3.38458.

TEST PASSED

****

Listing 3. Zadatak 4 – 4 niti

**********DZ1Z4**********

Elapsed time - SEQ: 0.0162113.

Elapsed time - PAR(4): 0.0099634.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.0304958.

Elapsed time - PAR(4): 0.0227405.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.987612.

Elapsed time - PAR(4): 0.431045.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 4.15647.

Elapsed time - PAR(4): 1.71066.

TEST PASSED

****

Listing 4. Zadatak 4 – 8 niti

**********DZ1Z4**********

Elapsed time - SEQ: 0.00765223.

Elapsed time - PAR(8): 0.014162.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.0305258.

Elapsed time - PAR(8): 0.032857.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 0.942663.

Elapsed time - PAR(8): 0.421068.

TEST PASSED

****

**********DZ1Z4**********

Elapsed time - SEQ: 5.00235.

Elapsed time - PAR(8): 1.9775.

TEST PASSED

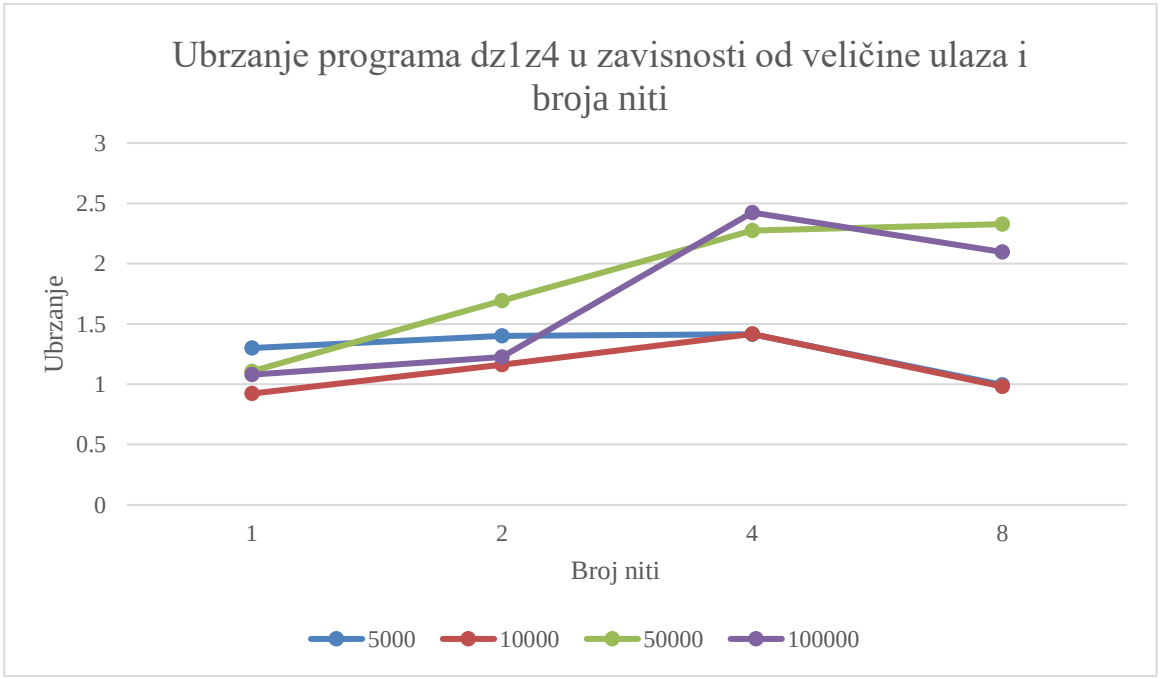
****

2.3.2. Tabela trajanja programa

	5000		10000		50000		100000	
	vreme	**ubrzanje**	**vreme**	**ubrzanje**	**vreme**	**ubrzanje**	**vreme**	**ubrzanje**
Sekvencijaln izvršavanje	0,01409 790		0,03222 170		0,98014 600		4,14646 000	
Zadatak 4 - 1 nit	0,01084 030	1,30050 829	0,03492 690	0,92254 680	0,88513 700	1,10733 819	3,84104 000	1,07951 492
Zadatak 4 - 2 niti	0,01005 580	1,40196 702	0,02775 110	1,16109 632	0,57910 900	1,69250 694	3,38458 000	1,22510 326
Zadatak 4 - 4 niti	0,00996 340	1,41496 879	0,02274 050	1,41693 015	0,43104 500	2,27388 324	1,71066 000	2,42389 487
Zadatak 4 - 8 niti	0,01416 200	0,99547 380	0,03285 700	0,98066 470	0,42106 800	2,32776 179	1,97750 000	2,09681 922

2.3.3. Grafici ubrzanja

U okviru ove sekcije su dati grafici ubrzanja paralelnih implementacija u odnosu na sekvencijalnu implementaciju. Ubrzanje na svim narednim graficima predstavljeno je kao odnos trajanja sekvencijalne implementacije programa i trajanja paralelizovane implementacije.



2.3.4. Diskusija dobijenih rezultata

Za ovaj zadatak se dobija donekle brže izvršavanje za jednu nit kod paralelne implementacije zato što je i sam sekvencijalni kod optimizovan u ovom rešenju – dvaput je izvršeno sažimanje petlji čime se smanjuje ukupan broj iteracija na nivou čitavog programa.

Kao što je i očekivano, paralelno rešenje problema je dosta sporije kod malih fajlova, pri čemu se usporanje povećava uz povećanje broja niti. Ovo je iz razloga što je *overhead* kreiranja i pokretanja niti dosta veliki, a veličina ulaza je dosta mala, pa se ne opravdava korišćenje paralelizacije. Za velike ulazne fajlove kod kojih sekvencijalna obrada dugo traje, paralelna implementacija pokazuje značajna ubrzanja (priloženo u prethodnim sekcijama poglavlja **Rezultati** ovog problema).

Takođe je isprobano i rešenje koje vrši paralelizaciju programa koristeći *task*-ove (ovo je navedeno u poglavlju *2.2.2.*), ali se ovo rešenje pokazalo dosta sporijim od paralelizacije pomoću *worksharing* direktiva. Pokretanje programa paralelizovanog pomoću *task*-ova je dalo korektne rezultate, ali je odnos dobijenih vremena između ovakvog načina paralelizacije i paralelizacije pomoću *worksharing* direktiva:

- *2x* sporiji – za osam niti
- *10x* sporiji – za jednu nit

3. PROBLEM 3 - KMEANS

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

3.1. Tekst problema

Paralelizovati program koji vrši k-means klasterizaciju podataka. Klasterizacija metodom k-srednjih vrednosti (eng. k-means clustering) je metod koji particioniše n objekata u k klastera u kojem svaki objekat pripada klasteru sa najbližom srednjom vrednošću. Objekat se sastoji od niza vrednosti - osobina (eng. features). Podelom objekata u potklastere, algoritam predstavlja sve objekte pomoću njihovi srednjih vrednosti (tzv. centroida potklastera). Inicijalni centroid za svaki potklaster se bira ili nasumično ili pomoću odgovarajuće heuristike. U svakoj iteratiji, algoritam pridružuje svaki objekat najbližem centroidu na osnovu definisane metrike. Novi centroidi za sledeću iteraciju se izračunavaju usrednjavanjem svih objekata unutar potklastera. Algoritam se izvršava sve dok se makar jedan objekat pomera iz jednog u drugi potklaster. Program se nalazi u direktorijumu kmeans u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke kmeans.c, cluster.c i kmeans_clustering.c. Analizirati dati kod i obratiti pažnju na generisanje novih centroida u svakoj iteraciji unutar datoteke kmeans_clustering.c. Ukoliko je potrebno međusobno isključenje prilikom paralelizacije programa, koristiti dostupne OpenMP konstrukte. Obratiti pažnju na efikasnost međusobnog isključenja niti i svesti ga na što je moguće manju meru uvođenjem pomoćnih struktura podataka. Ulazni test primeri se nalaze u direktorijumu data, a način pokretanja programa u datoteci run. [1, N]

3.2. Delovi koje treba paralelizovati

3.2.1. Diskusija

Kod koji sekvencijalno rešava ovaj problem razdvojen je u tri fajla – *kmeans.c, cluster.c,* i *kmeans_clustering.*c. Sama *main* funkcija koja se nalazi u *kmeans.c* fajlu samo vrši učitavanje svih neophodnih argumenata iz komandne linije i iz odgovarajućih fajlova i alokaciju prostora, a zatim kroz određeni broj iteracija (trenutno postavljeno na 1) poziva funkciju *cluster* iz *cluster.c* fajla. Ovu petlju nije moguće paralelizovati zato što postoji zavisnost po podacima za računanje centara klastera. Pored toga, kod iz *main* funkcije samo vrši ispis i dealokaciju prostora.

Funkcija *cluster* iz *cluster.c* fajla je samo *wrapper* funkcija koja alocira niz *membership* koji se koristi u *kmeans_clustering* funkciji, generiše *seed* random generatora, i generiše rezultat tako što

poziva funkciju *kmeans_clustering*. Povratna vrednost funkcije *kmeans_clustering* se zatim vraća u *main* funkciju preko promene argumenta *cluster_centres* koji je prosleđen po referenci. Dakle, ovaj kod se ne paralelizuje.

U ovom zadataku moguće je izvršiti paralelizaciju koda koji se u sekvencijalnoj implementaciji nalazi u fajlu *kmeans_clustering.c*. Ovo uključuje paralelizaciju funkcije *float** *kmeans_clustering(args).* Funkcije *int find_nearest_point(args), float euclid_dist_2(args)* ne treba paralelizovati jer se pozivaju iz već paralelizovanog koda, a ugneždavanje paralelizma nije omogućeno. Način na koji je izvršena paralelizacija za navedenu funkciju opisan je u narednom poglavlju.

3.2.2. Način paralelizacije

Za razliku od problema 4, u ovom problemu nije moguće modifikovati sekvencijalni kod kako bi se smanjio broj ukupnih iteracija svih petlji, već se parelelizacija vrši nad petljama kakve jesu. Ali u ovom problemu jeste moguće modifikovati sekvencijalni kod kako bi uopšte bila moguća primena paralelizacije, i ovo se radi za petlje kod kojih se vrši inicijalizacija nizova *clusters* i *new_centers*. Ovo je moguće uraditi zato što oba niza predstavljaju niz pokazivača kod kojih samo nulti ulaz ukazuje na ceo niz sa podacima (ukazuju na niz koji predstavlja linearizovanu matricu), a zatim svi ostali elementi nizova samo ukazuju na odgovarajuće delove linearizovanog niza. Sitnom promenom koda za računanje vrednosti za element *clusters[i]* (*1 <= i <* n) omogućava se paralelizacija ovog koda za inicijalizaciju. Kod za inicijalizaciju niza *membership* nije neophodno menjati i on se već može paralelizovati.

Kod za generisanje centara klastera nije moguće paralelizovati u slučaju kada vrednost promenljive *n* zaista nije random, već se u okviru petlje inkrementira. Zbog ovog uslova dobija se da nultom delu niza *clusters* odgovara vrednost *n = 0*, prvom delu odgovara vrednost *n = 1*, itd. Samim tim, ovaj kod se ne može paralelizovati. Kod bi bilo moguće paralelizovati kada bi se otkomentarisala linija na kojoj se nalazi naredba *n = (int)rand() % npoints;*.

Nakon ovoga neophodno je izvršiti paralelizaciju glavne petlje za obradu, koja je u ovom slučaju *do-while* petlja. Ona se sama po sebi ne može paralelizovati, ali je moguće paralelizovati kod koji se nalazi unutar ove petlje, a da onda samo jedna nit proverava da li je izračunata vrednost *delta* (koja se na kraju dobija tako što se redukuju rezultati iz svih niti) veća od vrednosti *threshold*, i ukoliko jeste ponovo poziva kod za obradu. Sama obrada obuhvata dve *for* petlje koje se mogu

paralelizovati pomoću *worksharing* direktiva, pri čemu se za prvu *for* petlju uz paralelizaciju koristi i redukcija vrednosti promenljive *delta*, dok je za drugu petlju neophodno izvršiti samo paralelizaciju.

U prvoj petlji postoji potencijalni *race condition*. Naime, moguće je da dve različite niti kao povratnu vrednost iz funkcije *find_nearest_point()* dobiju istu vrednost i to smeste u lokalnu promenljivu *index*. Pošto se pomoću ove promenljive gađa tačno određeni red niza *new_centers*, a više niti mogu istovremeno imati istu vrednost ove promenljive, neophodno je ovu sekciju zaštititi. U ovom zadatku je to odrađeno koristeći brave, pa se na početku *kmeans_clustering()* funkcije postavlja neophodna inicijalizacija, na kraj funkcije neophodna destrukcija, a zatim se pristup delu rezultujućeg niza koji je obuhvaćen promenljivom *index* štiti. Ovo usporava paralelizaciju ali je neophodno za korektno izvršavanje programa za bilo kakvo preplitanje niti.

Ovo je ujedno sve što je neophodno uraditi kako bi se paralelizovao program.

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.

Listing 1. Zadatak 5 – 1 nit

```
I/O completed
===================================SEQ=======================================

number of Clusters 5
number of Attributes 34

Time for process: 0.000168
===================================PAR=======================================

number of threads: 1
number of Clusters 5
number of Attributes 34

Time for process: 0.000167
TEST PASSED
```

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 2.130001

==========PAR==========

number of threads: 1

number of Clusters 5

number of Attributes 34

Time for process: 2.129688

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.452037

==========PAR==========

number of threads: 1

number of Clusters 5

number of Attributes 34

Time for process: 0.451945

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 1.914637

==========PAR==========

number of threads: 1

number of Clusters 5

number of Attributes 34

Time for process: 1.899101

TEST PASSED

****

Listing 2. Zadatak 5 – 2 niti

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.000247

==========PAR==========

number of threads: 2

number of Clusters 5

number of Attributes 34

Time for process: 0.000245

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 2.135121

==========PAR==========

number of threads: 2

number of Clusters 5

number of Attributes 34

Time for process: 2.133272

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.452108

==========PAR==========

number of threads: 2

number of Clusters 5

number of Attributes 34

Time for process: 0.451645

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 1.909433

==========PAR==========

number of threads: 2

number of Clusters 5

number of Attributes 34

Time for process: 1.932522

TEST PASSED

****

Listing 3. Zadatak 5 – 4 niti

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.000350

==========PAR==========

number of threads: 4

number of Clusters 5

number of Attributes 34

Time for process: 0.000348

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 2.133314

==========PAR==========

number of threads: 4

number of Clusters 5

number of Attributes 34

Time for process: 2.133864

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.451870

==========PAR==========

number of threads: 4

number of Clusters 5

number of Attributes 34

Time for process: 0.451897

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 1.912580

==========PAR==========

number of threads: 4

number of Clusters 5

number of Attributes 34

Time for process: 1.898070

TEST PASSED

****

Listing 4. Zadatak 5 – 8 niti

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.000183

==========PAR==========

number of threads: 8

number of Clusters 5

number of Attributes 34

Time for process: 0.000181

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 2.130714

==========PAR==========

number of threads: 8

number of Clusters 5

number of Attributes 34

Time for process: 2.130793

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 0.451810

==========PAR==========

number of threads: 8

number of Clusters 5

number of Attributes 34

Time for process: 0.451846

TEST PASSED

I/O completed

==========SEQ==========

number of Clusters 5

number of Attributes 34

Time for process: 1.910907

==========PAR==========

number of threads: 8

number of Clusters 5

number of Attributes 34

Time for process: 1.895204

TEST PASSED

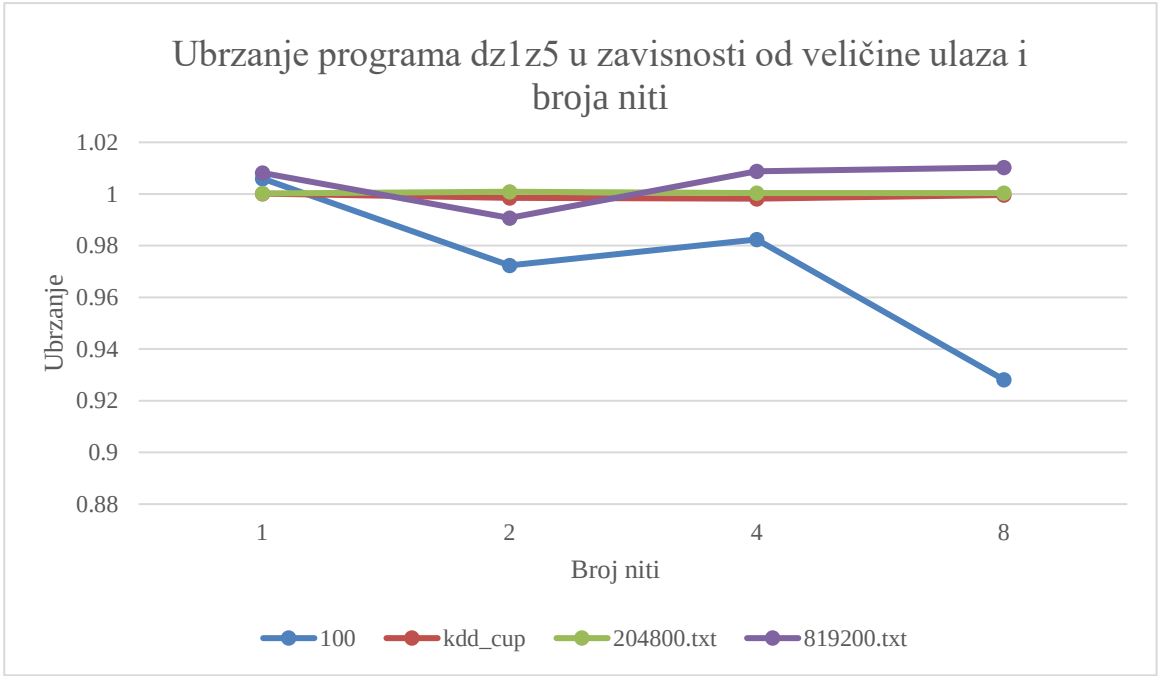
****

3.3.2. Tabela trajanja programa

	100		kdd_cup		204800.txt		819200.txt	
	vreme	ubrzanje	vreme	ubrzanje	vreme	ubrzanje	vreme	ubrzanje
Sekvencijalno izvršavanje	0,000168		2,130001		0,452037		1,914637	
Zadatak 5 - 1 nit	0,000167	1,005988024	2,129688	1,000146917	0,451945	1,000203636	1,899101	1,008180707
Zadatak 5 - 2 nit	0,000245	0,685714286	2,133272	0,998466675	0,451645	1,000863779	1,932522	0,990745353
Zadatak 5 - 4 nit	0,000348	0,482758621	2,133864	0,998189669	0,451897	1,000307398	1,898027	1,008727183
Zadatak 5 - 8 nit	0,000181	0,928176796	2,130793	0,999628307	0,451846	1,000420027	1,895204	1,010253838

3.3.3. Grafici ubrzanja

U okviru ove sekcije su dati grafici ubrzanja paralelnih implementacija u odnosu na sekvencijalnu implementaciju. Ubrzanje na svim narednim graficima predstavljeno je kao odnos trajanja sekvencijalne implementacije programa i trajanja paralelizovane implementacije.



3.3.4. Diskusija dobijenih rezultata

U ovom zadatku nije bilo moguće postići veliku paralelizaciju za date ulazne primere. Inicijalizacioni kod sadrži mali broj instrukcija unutar petlji pa je moguće da, nakon raspodele posla, niti brzo završe svoj deo i onda čekaju ostale na implicitnim barijerama. Prisustvo sinhronizacije pomoću brava potencijalno može usporiti sistem, a ukoliko se *do-while* petlja izvršava veliki broj

puta tada će biti dosta režijskih troškova oko uništavanja, kreiranja, i pokretanja niti zato što se paralelni region otvara unutar same petlje.

Na datom grafiku u sekciji *3.3.3.* primećuje se veliki pad performansi za 8 niti kod najmanjeg ulaznog fajla, što ima smisla zbog velikog *overhead*-a prilikom paralelizacije. Za ostale testove dobijeno je vrlo malo ubrzanje.

Generalno posmatrajući, ubrzanje paralelne implementacije u odnosu na sekvencijalnu je izuzetno malo (u najboljem slučaju je za date ulazne primere paralelna implementacija bolja od sekvencijalne kada se koristi 8 niti i kada se posmatra najveći ulazni fajl), i verovatno je neophodno uporediti performanse obe varijante programa sa daleko većom količinom ulaznih podataka.