Univerzitet u Beogradu - Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



Domaći zadatak 2 – MPI

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni asistent: | Kandidati: |
| doc. dr Marko Mišić | Marija Kostić 2015/0096  Stefan Milanović 2015/0361 |

Beograd, decembar 2018.

Sadržaj

[Sadržaj 2](#_Toc533287480)

[1. Problem 1 - SGEMM 3](#_Toc533287481)

[1.1. Tekst problema 3](#_Toc533287482)

[1.2. Delovi koje treba paralelizovati 3](#_Toc533287483)

[1.2.1. Diskusija 3](#_Toc533287484)

[1.2.2. Način paralelizacije 3](#_Toc533287485)

[1.3. Rezultati 5](#_Toc533287486)

[1.3.1. Logovi izvršavanja 5](#_Toc533287487)

[1.3.2. Tabela trajanja programa 13](#_Toc533287488)

[1.3.3. Grafici ubrzanja 13](#_Toc533287489)

[1.3.4. Diskusija dobijenih rezultata 16](#_Toc533287490)

[2. Problem 2 - Jacobi 17](#_Toc533287491)

[2.1. Tekst problema 17](#_Toc533287492)

[2.2. Delovi koje treba paralelizovati 17](#_Toc533287493)

[2.2.1. Diskusija 17](#_Toc533287494)

[2.2.2. Način paralelizacije 18](#_Toc533287495)

[2.3. Rezultati 19](#_Toc533287496)

[2.3.1. Logovi izvršavanja 19](#_Toc533287497)

[2.3.2. Tabela trajanja programa 21](#_Toc533287498)

[2.3.3. Grafici ubrzanja 22](#_Toc533287499)

[2.3.4. Diskusija dobijenih rezultata 22](#_Toc533287500)

[3. Problem 3 - Kmeans 24](#_Toc533287501)

[3.1. Tekst problema 24](#_Toc533287502)

[3.2. Delovi koje treba paralelizovati 24](#_Toc533287503)

[3.2.1. Diskusija 24](#_Toc533287504)

[3.2.2. Način paralelizacije 25](#_Toc533287505)

[3.3. Rezultati 27](#_Toc533287506)

[3.3.1. Logovi izvršavanja 27](#_Toc533287507)

[3.3.2. Tabela trajanja programa 35](#_Toc533287508)

[3.3.3. Grafici ubrzanja 35](#_Toc533287509)

[3.3.4. Diskusija dobijenih rezultata 35](#_Toc533287510)

1. Problem 1 - SGEMM

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 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 ← 𝛼 ∙ 𝐴 ∙ 𝐵 + 𝛽 ∙ 𝐶

Program se nalazi u datoteci sgemm.c u arhivi koja je priložena uz ovaj dokument. Proces sa rangom 0 treba da učita ulazne podatke, raspodeli posao ostalim procesima, na kraju prikupi dobijene rezultate i ravnopravno učestvuje u obradi. Za razmenu podataka, koristiti rutine za kolektivnu komunikaciju. Program testirati sa parametrima koji su dati u datoteci run. [1, N]

* 1. Delovi koje treba paralelizovati
     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.

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 dva načina – prvi način deli posao ravnopravno između procesa (uključujući i master proces, koji vrši učitavanje i prikupljanje rezultata, a takođe vrši i obradu), dok drugi način podrazumeva korišćenje *manager-worker* modela, pri čemu master predstavlja proces gospodara (menadžera).

* + 1. Način paralelizacije

**Zadatak 1**

Rešenje ovog zadatka nalazi se u datoteci *dz2z1.cpp*. U ovom zadatku svaki proces obrađuje *chunk* redova izlazne matrice koristeći deo matrice A koji njemu pripada i kopiju matrice B koja mu je neophodna za samu obradu podataka.

Ovo je postignuto tako što svi procesi pozivaju funkciju *basicSgemm\_par().* Unutar ove funkcije svaki proces najpre odredi svoj rank unutar komunikatora i broj procesa u komunikatoru i nakon toga pozivaju *broadcast* funkciju za sve neophodne podatke. Ove podatke jedino poseduje master proces koji ih šalje svim ostalim procesima. U ovom trenutku izvršena je raspodela samo neophodnih prostih promenljivih i sada se alociraju tri bafera (ovo radi svaki proces, uključujući i master proces). Bafer A sadrži odgovarajuće redove originalne matrice A koju ovaj proces treba da obradi, bafer B sadrži kopiju matrice B, i bafer C služi za lokalno čuvanje rezultata koji će kasnije biti poslat master procesu na sakupljanje. Matrica A se od mastera šalje svim procesima koristeći *Scatter* funkciju, a nakon ove raspodele svaki proces odradi svoj deo obrade i zatim pozivom funkcije *Gather* vraća deo rezultata masteru koji to smešta u odgovarajući deo matrice C koja predstavlja rezultat paralelne obrade.

Bitno je primetiti da je matrica A linearizovana *po kolonama.* Kako bi se omogućilo slanje samo određenih redova ove matrice svim procesima korišćen je korisnički definisan vektorski tip *row\_matA*. Ovaj tip uzima *k* blokova dužine *1* sa *stride-om m* između blokova čime se postiže dohvatanje reda matrice A. Ukoliko se uzme *chunk* za dužinu bloka umesto 1, dobija se dohvatanje *chunk* redova matrice A, i upravo ovo se koristi za raspodelu matrice na delove procesima.

**Zadatak 2**

Rešenje ovog zadatka nalazi se u datoteci *dz2z2.cpp*. U ovom zadatku neophodno je koristiti *manager-*worker model – master čita neophodne podatke i generiše poslove koje će radnici obrađivati. Po prijemu podataka radnik vrši obradu, vraća rezultat gospodaru i šalje signal da je spreman da primi sledeći posao. Sada master ne vrši obradu kao u prethodnom zadatku, već samo radi podelu posla i sakupljanje rezultata sve dok problem nije u celosti rešen.

Main funkcija izgleda identično kao u prethodnom zadatku. Svi pozivaju funkciju *basicSgemm\_par()* unutar koje će biti izvršena raspodela posla. Opet se prvo radi *broadcast* svih neophodnih prostih promenljivih, kao i matrice B koja je neophodna svim procesima za izračunavanje rezultata. Za razliku od prethodnog zadatka, sada se ne šalju odmah delovi matrice A ostalim procesima, već se matrica A šalje red po red od mastera ka radnicima i to samo kada je radnik spreman da krene nov ciklus obrade. Takođe u ovom zadatku se sam kod procesa sa rankom master i kod drugih procesa dosta razlikuje.

Master sada prati koliko je taskova poslato radnicima i koliko rezultata je primljeno, a zatim u odgovarajućim petljama radi slanje podataka za obradu i primanje rezultata. Jedan task predstavlja izračunavanje jednog reda matrice C (detaljnija diskusija o ovome će se naći u poglavlju *1.3.4.*). Kada se šalje task radniku, najpre se šalje redni broj reda matrice A koji će radnik obraditi, a nakon toga i sam red matrice. Ovo se radi pomoću poziva asinhrone *Isend* funkcije Kada se prima rezultat od radnika (ovo radi samo master), on poziva sinhronu *Recv* funkciju i to sa parametrom *MPI\_ANY\_SOURCE* što omogućava prijem od bilo kog radnika koji je završio. Nakon ovog prijema povećava se brojač primljenih rezultata od svih radnika, a rezultat koji radnik šalje se smešta na odgovarajuće mesto u matrici C. Ukoliko ima još posla, master radniku koji je upravo završio šalje novi red matrice A na obradu. Ukoliko više nema posla, master šalje *end* poruku radnicima.

Što se koda procesa radnika tiče, on ima odgovarajuće bafere za matricu A i C i jedan flag *needed*, koji je inicijalno postavljen na *true* i označava to da li radnik uopšte treba da čeka novi task ili treba da završi rad*.* Radnik se vrti u ovoj petlji dokle god je *needed* ispunjeno i, analogno slanju koje se vrši u masteru, prvo prima redni broj reda matrice A a zatim i sam red, vrši obradu koja je identična kao u prvom zadatku, i zatim masteru šalje rezultat. Kada se šalje rezultat, prvo se šalje redni broj reda (kako bi master znao gde da smesti rezultat radnika), a zatim i sam rezultat (koji je radnik kod sebe lokalno čuvao u baferu C).

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1 koristeći rešenja prethodne dve implementacije.

* + 1. Logovi izvršavanja

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

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.00781145.

Elapsed time - PAR(1): 0.00804341.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 12.9546

Elapsed time - PAR(1): 11.9514.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.0084999.

Elapsed time - PAR(2): 0.0.00456744.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 12.106.

Elapsed time - PAR(2): 3.442.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.126872.

Elapsed time - PAR(4): 0.0044802.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 11.427.

Elapsed time - PAR(4): 1.91111.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.0127321.

Elapsed time - PAR(8): 0.0062112.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 15.7805.

Elapsed time - PAR(8): 2.0381.

TEST PASSED

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Listing 5. Zadatak 2 – 1 nit

Inadequate number of processes

--------------------------------------------------------------------------

MPI\_ABORT was invoked on rank 0 in communicator MPI\_COMM\_WORLD

with errorcode -10.

NOTE: invoking MPI\_ABORT causes Open MPI to kill all MPI processes.

You may or may not see output from other processes, depending on

exactly when Open MPI kills them.

--------------------------------------------------------------------------

Inadequate number of processes

--------------------------------------------------------------------------

MPI\_ABORT was invoked on rank 0 in communicator MPI\_COMM\_WORLD

with errorcode -10.

NOTE: invoking MPI\_ABORT causes Open MPI to kill all MPI processes.

You may or may not see output from other processes, depending on

exactly when Open MPI kills them.

--------------------------------------------------------------------------

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.0125579.

Elapsed time - PAR(2): 0.00879433.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 12.0136.

Elapsed time - PAR(2): 3.66735.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.00975984.

Elapsed time - PAR(4): 0.00312711.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 13.3526.

Elapsed time - PAR(4): 1.40961.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 0.0126909.

Elapsed time - PAR(8): 0.00578494.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Elapsed time - SEQ: 16.5002.

Elapsed time - PAR(8): 1.37175.

TEST PASSED

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. Tabela trajanja programa

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **small input** | | | | **medium input** | | |
|  | **Sekvencijalno vreme** | **Paralelno vreme** | **ubrzanje** | **Sekvencijalno vreme** | | **Parelelno vreme** | **ubrzanje** |
| **Zadatak 1 – 1 nit** |  |  |  |  | |  |  |
| **Zadatak 1 – 2 niti** |  |  |  |  | |  |  |
| **Zadatak 1 – 4 niti** |  |  |  |  | |  |  |
| **Zadatak 1 – 8 niti** |  |  |  |  | |  |  |
| **Zadatak 2 – 1 nit** |  |  |  |  | |  |  |
| **Zadatak 2 – 2 niti** |  |  |  |  | |  |  |
| **Zadatak 2 – 4 niti** |  |  |  |  | |  |  |
| **Zadatak 2 – 8 niti** |  |  |  |  | |  |  |

* + 1. 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.

* + 1. 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.

// ovde diskusija o dz1z2 ako je task 1 celija, 1 red, ili vise redova

1. Problem 2 - Jacobi

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 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. Ukoliko je moguće, koristiti rutine za neblokirajuću komunikaciju za razmenu poruka. Program testirati sa parametrima koji su dati u datoteci run. [1, N]

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Za razliku od paralelne implementacije koristeći OpenMP, ovde je neophodno više izmeniti način obrade kako bi se omogućila najveća moguća paralelizacija. Moguće je izbeći korišćenje niza *b* u potpunosti *–* ovaj niz se nikad ne menja (iz njega se samo čita) i od početka programa ima vrednosti 0 za sve elemente niza osim za poslednji, koji sadrži vrednost *n+1*. Samim tim moguće je ovaj niz ukloniti, a zatim koristiti vrednost *n+1* umesto *b[i]* samo u procesu koji obrađuje poslednji element poslednjeg dela niza *x*. Ovim se izbegava *scatter* niza *b* od mastera ka ostalim procesima ili njegova inicijalizacija u svim procesima.

Pored ovoga, moguće je izbeći dosta komunikacije oko prepisivanja niza *xnew* u niz *x* tako što se kod paralelizuje tako da proces koji nije master svoj rezultat šalje samo kada se sve iteracije petlje *for (it = 0; it < m; i++)* završe. To se omogućava tako što deo obrade koji generiše vrednost promenljive *r* lokalno (dakle, u okviru jednog procesa) koristi svoj niz *xnew* (u kodu je ovo *xnew-buffer*) umesto niza *x* (što je urađeno u sekvencijalnoj verziji gde se prepisivanje iz niza *xnew* u niz *x* radi pre računanja vrednosti r). Nakon ovog koraka, proces lokalno vrši prepisivanje iz svog niza *xnew* u svoj niz *x* (u kodu su ovo baferi). Ovo omogućava odlaganje komunikacije između drugih procesa i mastera do samog kraja obrade.

Izbegnuta je inicijalizacija i raspodela početnog niza *x* zato što je on inicijalno popunjen nulama. Nema potrebe zagušiti mrežu rađenjem *scatter*-a ovog niza, već je moguće da svaki proces inicijalizuje početnu vrednost svog bafera niza *x* nulama, a zatim radi odgovarajuću obradu.

Takođe, unutar funkcija koje vrše obradu, kod je izmenjen tako da svaki proces obrađuje svoj niz *x\_buffer* koji je dužine *chunk* (*chunk* se dobija kao rezultat deljenja vrednosti *n* (dužine originalnog niza *x*) sa brojem procesa koji rade obradu). Pritom se prvo vrši slanje graničnih elemenata niza (ukoliko je neophodno), zatim se vrši obrada za *i=1..chunk -2,* pa se na kraju čeka na dobijanje graničnih elemenata od okolnih procesa (objašnjeno u narednom poglavlju) i onda se vrši obrada za vrednosti *i=0* i *i=chunk-1*.

* + 1. Način paralelizacije

Paralelizacija ovog problema funkcioniše na sličan način kao i u zadatku 1 – naime, i master radi obradu zajedno sa ostalim procesima i rezultat čuva u okviru svog lokalnog bafera x\_*buffer*. Za računanje novih vrednosti tog niza koristi se pomoćni niz koji je takođe lokalan za sve procese *xnew\_buffer,* a za računanje vrednosti *d* i *r* svaki proces poseduje svoje promenljive *d\_local* i *r\_local*. Ove dve promenljive se redukuju koristeći *Ireduce* zato što procesi mogu nastaviti ostatak obrade i ne moraju da čekaju da i ostali procesi završe odgovarajuću računicu. Master kada pozove ovu funkciju takođe poziva i *Wait* funkciju koristeći *MPIRequest* objekat koji mu je vraćen pozivom *Ireduce* funkcije kako bi sačekao da svi procesi završe računanje svojih *d\_local* i ­*r\_local* vrednosti, i zatim radi ispis redukovanih vrednosti.

Ovo predstavlja jedan deo obrade. Za sve procese obrada se radi *m* puta i sastoji se od sledećih koraka: najpre se radi *jacobi update* i računanje vrednosti *d\_local,* zatim se radi računanje vrednosti *r\_local,* a nakon toga se vrši prepisivanje niza *xnew\_buffer* u *xnew* (ovo je izmenjeno u odnosu na sekvencijalnu implementaciju tako da se ne radi plitko kopiranje, već se samo obrću vrednosti pokazivača na ova dva niza). Posle ovoga radi se redukcija vrednosti *r\_local* i *d\_local* po potrebi. Ova obrada se ponavlja *m* puta, pri čemu vrednosti *m* i *n* poseduju svi procesi jer su to vrednosti prosleđene kao argumenti komandne linije.

Ovim je već postignut najveći deo paralelizacije – svaki proces obrađuje svoj deo niza i na kraju se rezultat šalje masteru na sakupljanje, ispis, i proveru tačnosti. Jedini problem koji se ovde javlja jeste da postoji zavisnost po podacima prilikom računanja vrednosti *xnew\_buffer[i]* i *r\_local* za graničnu vrednosti brojača *i,* a to su 0 i *chunk-1.* Zavisnost po podacima se javlja zato što je za računanje *i=0* i *i=chunk-1* elementa neophodno dobiti vrednost niza *x* za *i=chunk-1* onog procesa čiji je rang za 1 manji od trenutnog ranga (ukoliko postoji) i vrednost niza *x* za *i=0* onog procesa čiji je rang za 1 veći od trenutnog ranga (ukoliko postoji), respektivno.

Kao rešenje ovog problema se u funkcijama koje rade *jacobi update*, računanje *d\_local* i *r\_local* vrednosti najpre vrši slanje odgovarajućih graničnih elemenata svojim procesima susedima pomoću funkcije *Isend* (nulti element se šalje procesu sa rangom niži za 1, a *chunk-1* element se šalje procesu sa rangom viši za 1). Pored ovog slanja, takođe se i šalju zahtevi za primanje odgovarajućeg podatka od suseda pomoću *Irecv* funkcije. Obe ove funkcije su asinhrone zato što svaki proces može da nastavi da računa rezultate za sve elemente kod kojih ne postoji zavisnost po podacima, a to su elementi za vrednost brojača *i=1..chunk-2.* Ovim se dobija ubrzanje paralelizacije jer ne postoji nepotrebno blokirajuće čekanje na neophodne vrednosti koje poseduju susedi, već se radi korisna obrada sve dok ti podaci nisu neophodni. Kada se izvrši obrada svih elemenata osim graničnih, tada se mora čekati da se završi prijem od suseda pozivom funkcije *MPI\_Wait*, a onda se i granični elementi obrađuju pošto je neophodna informacija od suseda pristigla.

Time se postiže najveći mogući nivo paralelizacije koristeći rutine za neblokirajuću komunikaciju.

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 2 koristeći rutine za neblokirajuću komunikaciju za razmenu poruka.

* + 1. Logovi izvršavanja

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

Listing 1. Zadatak 4 – 1 nit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.0875039.**

**Elapsed time - PAR(1): 0.0618948.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.262239.**

**Elapsed time - PAR(1): 0.227064.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 6.78218.**

**Elapsed time - PAR(1): 6.40606.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 26.7743.**

**Elapsed time - PAR(1): 26.1202.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Listing 2. Zadatak 4 – 2 niti

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.0855327.**

**Elapsed time - PAR(2): 0.0367261.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.255073.**

**Elapsed time - PAR(2): 0.134064.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 6.64812.**

**Elapsed time - PAR(2): 3.42375.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 28.503.**

**Elapsed time - PAR(2): 15.6521.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Listing 3. Zadatak 4 – 4 niti

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.112622.**

**Elapsed time - PAR(4): 0.0248323.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.28755.**

**Elapsed time - PAR(4): 0.0928026.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 7.18861.**

**Elapsed time - PAR(4): 2.03025.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 29.3591.**

**Elapsed time - PAR(4): 8.69126.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Listing 4. Zadatak 4 – 8 niti

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.25688.**

**Elapsed time - PAR(8): 0.0748413.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 0.421779.**

**Elapsed time - PAR(8): 0.0811448.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 11.1522.**

**Elapsed time - PAR(8): 1.89955.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DZ2Z3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Elapsed time - SEQ: 46.1065.**

**Elapsed time - PAR(8): 7.73964.**

**TEST PASSED**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + 1. Tabela trajanja programa

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **5000** | | **10000** | | **50000** | | **100000** | |
|  | **vreme** | **ubrzanje** | **vreme** | **ubrzanje** | **vreme** | **ubrzanje** | **vreme** | **ubrzanje** |
| **Sekvencijaln izvršavanje** | 0,01409790 |  | 0,03222170 |  | 0,98014600 |  | 4,14646000 |  |
| **Zadatak 4 - 1 nit** | 0,01084030 | 1,30050829 | 0,03492690 | 0,92254680 | 0,88513700 | 1,10733819 | 3,84104000 | 1,07951492 |
| **Zadatak 4 - 2 niti** | 0,01005580 | 1,40196702 | 0,02775110 | 1,16109632 | 0,57910900 | 1,69250694 | 3,38458000 | 1,22510326 |
| **Zadatak 4 - 4 niti** | 0,00996340 | 1,41496879 | 0,02274050 | 1,41693015 | 0,43104500 | 2,27388324 | 1,71066000 | 2,42389487 |
| **Zadatak 4 - 8 niti** | 0,01416200 | 0,99547380 | 0,03285700 | 0,98066470 | 0,42106800 | 2,32776179 | 1,97750000 | 2,09681922 |

* + 1. 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.

* + 1. 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

1. Problem 3 - Kmeans

U okviru ovog poglavlja je dat kratak izveštaj u vezi rešenja zadatog problema 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 različite mogućnosti i nivoe na kojima se može obaviti paralelizacija koda, kao i na deo koda za generisanje novih centroida u svakoj iteraciji unutar datoteke kmeans\_clustering.c. Ulazni test primeri se nalaze u direktorijumu data, a način pokretanja programa u datoteci run. [1, N]

* 1. Delovi koje treba paralelizovati
     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. Pored toga, kod iz *main* funkcije samo vrši ispis i dealokaciju prostora. U paralelnoj verziji, većinu ovog posla radi proces sa rangom master. On će pokrenuti sekvencijalnu implementaciju programa i zapamtiti vreme izvršavanja, a zatim će odgovarajuće vrednosti pomoću funkcije *broadcast* proslediti i ostalim procesima koji rade obradu. Na kraju, proces master vrši ispis dobijenih rezultata i upoređuje validnost dobijenih vrednosti sa onim dobijenih iz sekvencijalne implementacije, a i ispisuje vremena izvršavanja obe implementacije.

Funkcija *cluster* iz *cluster.c* fajla je samo *wrapper* funkcija koja alocira niz *membership* koji se koristi u *kmeans\_clustering* funkciji, generiše *seed* randomgeneratora, 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. U slučaju paralelizacije koristeći MPI svi procesi će pozvati funkciju *cluster\_par*, samo će u njoj alocirati *membership* koji odgovara samo objektima koji su dodeljeni tom procesu, a zatim se unutar *kmeans\_clustering\_par* funkcije vrši komunikacija između mastera i ostalih procesa.

U ovom zadataku najveće ubrzanje dobija se paralelizacijom funkcije koja 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)* su pomoćne funkcije koje ne treba paralelizovati.Način na koji je izvršena paralelizacija za navedenu funkciju opisan je u narednom poglavlju.

* + 1. Način paralelizacije

Ideja paralelizacije ovog problema jeste da se svakom procesu dodeli određeni deo niza *clusters* na izračunavanje, i da se zatim u master procesu izvrši sakupljanje rezultata obrade, pri čemu bi i master radio obradu zajedno sa ostalim procesima. Pošto se određivanje niza *clusters* radi pomoću dve velike *for* petlje koje se ponavljaju sve dok važi *delta > threshold* cilj je njih paralelizovati na najoptimalniji mogući način. Implementacija koja ovo postiže detaljno je opisana u nastavku ovog poglavlja:

Slično kao i u prethodnom zadatku, sama *main* funkcija je osmišljena tako da master prikuplja sve argumente sa komandne linije, vrši čitanje ulaznih podataka iz dostavljenih fajlova, pokreće sekvencijalnu verziju i meri vreme izvršavanja obe implementacije. Nakon ovoga, on vrši raspodelu neophodnih promenljivih, i na kraju paralelne obrade sakuplja rezultat i vrši ispis i upoređivanje vremena između paralelne i sekvencijalne verzije i proverava tačnost dobijenog rezultata.

Na početku *main* funkcije svaki proces za sebe nalazi *rank* i *size* globalnog komunikatora, a od procesa mastera dobija vrednosti za *numObjects, numAttributes, nclusters i threshold* pozivom funkcije *Broadcast.* Pored ovoga, svi procesi koji nisu master imaju bafer koji sadrži kopiju matrice atributa. Nakon ove raspodele, svi procesi pozivaju funkciju *cluster\_par*, pri čemu se u sekvencijalnoj verziji rezultat vraćao putem parametra *cluster\_centres\_par.* U ovoj implementaciji ovaj parametar se posmatra jedino u master procesu, pa će samo master i u funkciji *cluster\_par* vraćati rezultat, dok se za ostale procese ta povratna vrednost ne posmatra i nigde ne koristi. Samo sakupljanje rezultata se radi unutar *kmeans\_clustering\_par* funkcije, tako da se ništa ne radi na kraju poziva *kmeans\_clustering\_par* i *cluster\_par*  funkcija osim vraćanja dobijenih vrednosti.

Funkciju *cluster\_par* pozivaju svi procesi. Svaki proces za sebe izračuna dve vrednosti – *chunkClusters* i *chunkPoints*. Pošto se obrada sastoji iz dve velike petlje od kojih jedna ide do *npoints*, a druga do *nclusters*, idealno je podeliti ovu petlju tako da svaki proces radi deo ove obrade, a to se upravo omogućava podelom određenih struktura na *chunkClusters* ili *chunkPoints* delova i zatim te delove dodeliti određenim procesima. Samim tim, unutar ove funkcije svaki proces generiše svoj niz *membership* (koji je sada veličine *chunkPoints*) koji pomaže u izradi rezultata u *kmeans\_clustering\_par* funkciji. Jedino proces master u ovoj funkciji radi vraćanje rezultata u glavni program.

Funkcija *kmeans\_clustering\_par* sadrži najveći deo paralelizovanog koda. Kao što je diskutovano u prethodnom izveštaju za OpenMP, problem u paralelizaciji ovog koda jeste što se mora izvršiti redukcija niza *new\_centers\_len* i matrice *new\_centers* u sredini same obrade. U toku prve petlje, upis u *new\_centers\_len* niz i *new\_centers* matricu zavisi od vrednosti *index* koja se dobija pozivom funkcije *find\_nearest\_point*. Ova funkcija uvodi dva ograničenja prilikom paralelizacije:

1. Pošto funkcija kao parametar prihvata ceo niz *clusters* kao argument, zahteva se prisustvo ažurne kopije celog ovog niza pri svakoj iteraciji velike *do-while* petlje. Kako bi se ovo rešilo, na kraju *do-while* petlje vrši se prikupljanje rezultata trenutne iteracije u masteru pomoću *MPI\_Gather* poziva, dok na početku petlje master ovu ažurnu kopiju *clusters* niza daje i ostalim procesima pomoću *MPI\_Bcast* poziva.
2. Pošto povratna vrednost ove funkcije (tj. *index*) može predstavljati bilo koji broj iz opsega *0..numObjects*, a jedan proces radi samo sa *chunkPoints* delom ulaznih podataka, ovo zahteva da svaki proces mora pamtiti svoje bafere *local\_new\_centers\_len* i *local\_new\_centers* koji su veličine *nclusters* i *nclusters \* nfeatures*, a ne *chunkClusters* i *chunkCsluters \* nfeatures.* Tada se pri redukciji ovih lokalnih bafera dobijaju korektne vrednosti za sve objekte.

Takođe, unutar ove prve petlje svaki proces za sebe računa lokalnu vrednost parametra *delta* (u kodu nazvana *localDelta*) koju je neophodno redukovati pre same provere uslova *do-while* petlje. Pošto svi procesi rade nezavisno, a svi treba da završe u istom trenutku, poziva se funkcija *MPI\_Allreduce* za parametar *delta*. Nakon ovog poziva svi procesi u promenljivoj *delta* imaju ažurnu vrednost ovog podatka.

Između prve i druge *for* petlje obrade radi se redukcija dobijenih rezultata u svim procesima. Najpre se radi redukcija local\_new\_centers\_len niza u niz new\_centers\_len, a zatim redukcija matrice local\_new\_centers u new\_centers. Pošto i dalje svaki proces treba da radi nad svojim delom podataka, odmah se nakon ove redukcije vrši raspodela odgovarajućeg dela podataka odgovarajućem procesu. Iz ovog razloga, koristi se poziv funkcije *MPI\_Reduce\_scatter* za oba bafera u svim procesima. Nakon ovoga, izvršena je redukcija međurezultata, i oni su raspodeljeni nazad u odgovarajuće procese.

Nakon ovoga vrši se paralelizacija druge velike *for* petlje obrade, pri čemu sada svaki proces kreira svoj rezultat u nizu *local\_clusters*. Pošto svaki proces kreira deo rezultata, ova petlja je podeljena i svaki proces vrši *chunkClusters* iteracija ove petlje. Nakon izvršene obrade, vrši se prikupljanje svih rezultata koji se nalaze u *local\_clusters* matrici u *clusters* matrici koja se nalazi u master procesu. Ukoliko je ovo poslednja iteracija, master u *clusters* matrici ima celokupan rezultat i on se vraća kao rezultat obrade pomoću *return clusters;* naredbe i ova matrica propagira do glavnog programa kao rešenje problema. Ukoliko ovo nije poslednja iteracija, ponovo se radi *broadcast* sada ovog međurezultata koji se nalazi u matrici *clusters* i obrada se ponovo izvršava sve dok je broj premeštanja objekata iz jednog klastera u drugi veći od zadate vrednosti parametra *threshold.*

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 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

numObjects=100, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 0.001155**

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

number of threads: 1

number of Clusters 8

number of Attributes 34

**Time for process: 0.001093**

**TEST PASSED**

I/O completed

numObjects=494016, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 24.416080**

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

number of threads: 1

number of Clusters 8

number of Attributes 34

**Time for process: 25.038107**

**TEST PASSED**

I/O completed

numObjects=204796, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 5.807440**

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

number of threads: 1

number of Clusters 8

number of Attributes 34

**Time for process: 5.907594**

**TEST PASSED**

I/O completed

numObjects=819196, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 41.560881**

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

number of threads: 1

number of Clusters 8

number of Attributes 34

**Time for process: 41.783075**

**TEST PASSED**

Listing 2. Zadatak 5 – 2 niti

I/O completed

numObjects=100, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 0.000939**

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

number of threads: 2

number of Clusters 8

number of Attributes 34

**Time for process: 0.000734**

**TEST PASSED**

I/O completed

numObjects=494016, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 25.310123**

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

number of threads: 2

number of Clusters 8

number of Attributes 34

**Time for process: 13.162480**

**TEST FAILED**

I/O completed

numObjects=204796, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 5.906832**

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

number of threads: 2

number of Clusters 8

number of Attributes 34

**Time for process: 3.022614**

**TEST PASSED**

I/O completed

numObjects=819196, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 44.133237**

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

number of threads: 2

number of Clusters 8

number of Attributes 34

**Time for process: 21.836951**

**TEST FAILED**

Listing 3. Zadatak 5 – 4 niti

I/O completed

numObjects=100, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 0.004375**

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

number of threads: 4

number of Clusters 8

number of Attributes 34

**Time for process: 0.002913**

**TEST PASSED**

I/O completed

numObjects=494016, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 26.797842**

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

number of threads: 4

number of Clusters 8

number of Attributes 34

**Time for process: 6.978688**

**TEST FAILED**

I/O completed

numObjects=204796, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 6.140115**

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

number of threads: 4

number of Clusters 8

number of Attributes 34

**Time for process: 1.601039**

**TEST PASSED**

I/O completed

numObjects=819196, numclusters=8

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

number of Clusters 8

number of Attributes 34

**Time for process: 46.224978**

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

number of threads: 4

number of Clusters 8

number of Attributes 34

**Time for process: 11.541173**

**TEST FAILED**

* + 1. Tabela trajanja programa

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **100** | | **kdd\_cup** | | **204800.txt** | | **819200.txt** | |
|  | **vreme** | **ubrzanje** | **vreme** | **ubrzanje** | **vreme** | **Ubrzanje** | **vreme** | **ubrzanje** |
| **Sekvencijalno izvršavanje** | 0.000168 |  | 2.129494 |  | 0.452037 |  | 1.908579 |  |
| **Zadatak 5 - 1 nit** | 0.000167 | 1.005988024 | 2.191252 | 0.971816112 | 0.451945 | 1.000203565 | 1.909017 | 0.999770563 |
| **Zadatak 5 - 2 nit** | 0.000313 | 0.536741214 | 1.185964 | 1.795580642 | 0.209367 | 2.159065182 | 0.979043 | 1.949433273 |
| **Zadatak 5 - 4 nit** | 0.000284 | 0.591549296 | 0.592483 | 3.594185825 | 0.129205 | 3.498602995 | 0.534844 | 3.568477911 |
| **Zadatak 5 - 8 nit** | 0.001773 | 0.094754653 | 0.564875 | 3.769849967 | 0.133778 | 3.379008507 | 0.463245 | 4.120020723 |

* + 1. 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.

* + 1. Diskusija dobijenih rezultata

U ovom zadatku dobijena je velika paralelizacija za date ulazne primere, pri čemu je ubrzanje direktno zavisilo od broja niti. Za razliku od prethodnih problema, ovde je povećanje broja niti dovelo do značajnog ubrzanja izvršavanja programa. Dobijeni rezultati ukazuju na to da je uspešno izvršena paralelizacija i da je pokretanje mehanizma kreiranja niti i njihovog puštanja u rad opravdano jer su dobici u brzini izvršavanja dosta veliki. Testovi ispisuju TEST\_FAILED zbog male vrednosti konstante *ACCURACY* koja je zadata u zadatku.

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. Dobija se usporenje od oko *10x*. Za ostale ulazne primere dobijeno je ubrzanje koje linearno raste u zavisnosti od broja niti.

Diskusija potencijalnih rešenja

U ovom problemu je takođe moguće modifikovati sekvencijalni kod koji vrši inicijalizaciju sistema. Neophodno je izmeniti taj kod kako bi uopšte bila moguća primena paralelizacije, i ovo bi se radilo 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;*.

Pokušana je paralelizacija ovog inicijalizacionog dela koda na opisani način, ali se testiranjem takvog programa pokazalo da zapravo nastaje pad performansi, pa je u predatom rešenju ostavljena sekvencijalna inicijalizacija.

Takođe, pokušano je i korišćenje brava umesto ručne redukcije u prvoj *for* petlji, ali je i ovaj način doveo do usporenja sistema.