Univerzitet u Beogradu – Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



Domaći zadatak 2 – MPI

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni saradnici: | Studenti: |
| doc. dr Marko Mišić  dipl. ing. Pavle Divović | Ana Radovanović 2019/0282  Jovana Jaćimović 2019/0593 |

Beograd, decembar 2022.

Sadržaj

[Sadržaj 2](#_Toc123056766)

[1. Problem 1 3](#_Toc123056767)

[1.1. Tekst problema 3](#_Toc123056768)

[1.2. Delovi koje treba paralelizovati 3](#_Toc123056769)

[1.2.1. Diskusija 3](#_Toc123056770)

[1.2.2. Način paralelizacije 3](#_Toc123056771)

[1.3. Rezultati 4](#_Toc123056772)

[1.3.1. Logovi izvršavanja 4](#_Toc123056773)

[1.3.2. Grafici ubrzanja 5](#_Toc123056774)

[1.3.3. Diskusija dobijenih rezultata 7](#_Toc123056775)

[2. Problem 2 8](#_Toc123056776)

[2.1. Tekst problema 8](#_Toc123056777)

[2.2. Delovi koje treba paralelizovati 8](#_Toc123056779)

[2.2.1. Diskusija 8](#_Toc123056780)

[2.2.2. Način paralelizacije 8](#_Toc123056781)

[2.3. Rezultati 9](#_Toc123056782)

[2.3.1. Logovi izvršavanja 9](#_Toc123056783)

[2.3.2. Grafici ubrzanja 11](#_Toc123056784)

[2.3.3. Diskusija dobijenih rezultata 12](#_Toc123056785)

[3. Problem 3 14](#_Toc123056786)

[3.1. Tekst problema 14](#_Toc123056787)

[3.2. Delovi koje treba paralelizovati 14](#_Toc123056795)

[3.2.1. Diskusija 14](#_Toc123056796)

[3.2.2. Način paralelizacije 15](#_Toc123056797)

[3.3. Rezultati 15](#_Toc123056798)

[3.3.1. Logovi izvršavanja 15](#_Toc123056799)

[3.3.2. Grafici ubrzanja 16](#_Toc123056800)

[3.3.3. Diskusija dobijenih rezultata 16](#_Toc123056801)

[4. Problem 3 18](#_Toc123056802)

[4.1. Tekst problema 18](#_Toc123056803)

[4.2. Delovi koje treba paralelizovati 18](#_Toc123056804)

[4.2.1. Diskusija 18](#_Toc123056805)

[4.2.2. Način paralelizacije 18](#_Toc123056806)

[4.3. Rezultati 19](#_Toc123056807)

[4.3.1. Logovi izvršavanja 19](#_Toc123056808)

[4.3.2. Grafici ubrzanja 19](#_Toc123056809)

[4.3.3. Diskusija dobijenih rezultata 20](#_Toc123056810)

1. Problem 1

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

* 1. Tekst problema

Paralelizovati program koji vrši određivanje ukupnog broja prostih brojeva u zadatom opsegu. Program se nalazi u datoteci prime.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. Delovi koje treba paralelizovati
     1. Diskusija

Paralelizacija se vrši u funkciji prime\_number, koja od računa ukupan broj prostih brojeva u zadatom opsegu. Funkcija sadrži dve for petlje, od kojih spoljašnja predstavlja prolazak kroz brojeve u tom opsegu, dok unutrašnja predstavlja prolazak kroz delioce jednog broja. Paralelizacija je urađena nad spoljašnjom for petljom.

* + 1. Način paralelizacije

Na početku paralelizacije inicijalizovan je MPI svet u main funkciji. Zatim je inicijalizovan master proces, koji u ovom slučaju ima rank 0 i koji je zadužen za raspoređivanje poslova ostalim procesima kao i za ispis dobijenih rezultata. U funkciji prime\_number dobijen je ukupan broj procesa koji će raditi na rešavanju problema, kao i rank svih procesa. Zatim master proces radi na ravnomernom raspoređivanju iteracija for petlje svim ostalim procesima. Master proces šalje početni i krajnji indeks niza svakom procesu pomoću grupne komunikacije kako je i naznačeno u zadatku. Svaki proces izvršava deo for petlje koji mu je dodeljen, uključujući i master proces. Na kraju se radi redukcija nad total promenljivom i tako redukovana promenljiva se vraća kao povratna vrednost funkcije prime\_number. Problem koji se uočava u ovakvoj paralelizaciji je taj što se niz deli na nekoliko sukcesivnih delova, tako da prvi proces obrađuje prvih n elemenata, dok poslednji proces obrađuje poslednjih n elemenata. Kako su prvi elementi niza mali brojevi, druga petlja će imati manje iteracija i samim tim će prvi proces dosta ranije završiti posao od poslednjeg, što dovodi do toga da raspodela poslova nije uniformna. Međutim ovakva raspodela je dobra jer obezbeđuje prostornu lokalnost, te se tako dobija nad performansama.

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1.

* + 1. Logovi izvršavanja

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

SEKVENCIJALNO IZVRŠAVANJE

./prime 1 131072 2

Time elapsed: 1.302937

./prime 5 500000 10

Time elapsed: 12.574481

./prime 1 65536 4

Time elapsed: 0.274239

Listing 1. Sekvencijalno izvršavanje problema 1

NUM\_THREADS=4

mpirun -np 4 ./dz2z1.exe 1 131072 2

Time elapsed: 0.563222

mpirun -np 4 ./dz2z1.exe 5 5000000 10

Time elapsed: 5.48196

mpirun -np 4 ./dz2z1.exe 1 65536 4

Time elapsed: 0.118333

Listing 2. Izvršavanje problema 1 za N=4 niti

NUM\_THREADS=2

mpirun -np 2 ./dz2z1.exe 1 131072 2

Time elapsed: 0.957773

mpirun -np 2 ./dz2z1.exe 5 5000000 10

Time elapsed: 9.260602

mpirun -np 2 ./dz2z1.exe 1 65536 4

Time elapsed: 0.201529

Listing 3. Izvršavanje problema 1 za N=2 niti

NUM\_THREADS=1

mpirun -np 1 ./dz2z1.exe 1 131072 2

Time elapsed: 1.303196

mpirun -np 1 ./dz2z1.exe 5 5000000 10

Time elapsed: 12.576027

mpirun -np 1 ./dz2z1.exe 1 65536 4

Time elapsed: 0.274098

Listing 4. Izvršavanje problema 1 za N=1 niti

* + 1. Grafici ubrzanja

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

Slika 1. Grafik zavisnosti ubrzanja izvršavanja problema 1 od ulaznih parametara za N = 4 niti

Slika 2. Grafik zavisnosti ubrzanja izvršavanja problema 1 od ulaznih parametara za N = 2 niti

Slika 3. Grafik zavisnosti ubrzanja izvršavanja problema 1 od ulaznih parametara za N = 1 niti

Slika 4. Grafik zavisnosti ubrzanja izvršavanja problema 1 od ulaznih parametara za 4, 2 i 1 nit

* + 1. Diskusija dobijenih rezultata

Kao što možemo videti iz priloženih logova i grafika, a i kao sto je očekivano, ubrzanje raste srazmerno sa porastom broja procesa. Najoptimalnije performanse dobijaju sa kada je broj procesa jednak 4 i tada je ubrzanje nekih 2.3, u zavisnosti od veličine podataka nad kojim se algoritam izvršava mogu se videti mala odstupanja. Sa smanjenjem broja procesa opada i ubrzanje, pa je ubrzanje koje dobijemo koristeći 2 procesa nešto manje i iznosi nekih 1.3

1. Problem 2

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

* 1. Tekst problema

Paralelizovati program koji vrši izračunavanje 3D Poasonove jednačine korišćenjem Feyman-Kac algoritma. Algoritam stohastički računa rešenje parcijalne diferencijalne jednačine krenuvši N puta iz različitih tačaka domena. Tačke se kreću po nasumičnim putanjama i prilikom izlaska iz granica domena kretanje se zaustavlja računajući dužinu puta do izlaska. Proces se ponavlja za svih N tačaka i konačno aproksimira rešenje jednačine. Program se nalazi u datoteci feyman.c u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci run.

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Paralelizacija se vrši u main-u, u kome se tačke generišu i računa dužina puta do izlaska. Kako je problem dosta složen, funkcija sadrži 4 for petelje. Odlučeno je da se paralelizacija radi nad prvom for petljom.

* + 1. Način paralelizacije

Na početku paralelizacije inicijalizovan je MPI svet u main funkciji. Zatim je inicijalizovan master proces, koji u ovom slučaju ima rank 0 i koji je zadužen za ispis dobijenih rezultata. U funkciji main dobijen je i ukupan broj procesa koji će raditi na rešavanju problema, kao i rank svih procesa.

For petlja je paralelizovana tako da se niz deli ciklično, tako da svaki proces u iteraciji uzme element niza čiji je indeks za broj procesa veći od indeksa iz prethodne iteracije, kako bi raspodela po procesima bila uniformna. Istina je da se ovakvom podelom vrši narušavanje lokalnosti podataka, ali ispostavlja se da se ovako dobijaju najbolji rezultati. Na kraju se radi redukcija nad promenljivama error i n\_inside , a zatim samo master računa apsolutnu grešku na osnovu ove dve promenljive.

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 2.

* + 1. Logovi izvršavanja

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

SEKVENCIJALNO IZVRŠAVANJE

time ./feyman 1000

real 21.243

time ./feyman 5000

real 105.367

time ./feyman 10000

real 211.274

time ./feyman 20000

real 422.919

Listing 5. Sekvencijalno izvršavanje problema 2

NUM\_THREADS=4

mpirun -np 4 ./dz2z2.exe 1000

Time elapsed: 5.411879

mpirun -np 4 ./dz2z2.exe 5000

Time elapsed: 29.018792

mpirun -np 4 ./dz2z2.exe 10000

Time elapsed: 59.193186

mpirun -np 4 ./dz2z2.exe 20000

Time elapsed: 122.083655

Listing 6. Izvršavanje problema 2 za N=4 niti

NUM\_THREADS=2

mpirun -np 2 ./dz2z2.exe 1000

Time elapsed: 11.187869

mpirun -np 2 ./dz2z2.exe 5000

Time elapsed: 61.025316

mpirun -np 2 ./dz2z2.exe 10000

Time elapsed: 114.848061

mpirun -np 2 ./dz2z2.exe 20000

Time elapsed: 221.481325

Listing 7. Izvršavanje problema 2 za N=2 niti

NUM\_THREADS=1

mpirun -np 1 ./dz2z2.exe 1000

Time elapsed: 20.961237

mpirun -np 1 ./dz2z2.exe 5000

Time elapsed: 104.896834

mpirun -np 1 ./dz2z2.exe 10000

Time elapsed: 212.84234

mpirun -np 1 ./dz2z2.exe 20000

Time elapsed: 438.982538

Listing 8. Izvršavanje problema 2 za N=1 niti

* + 1. Grafici ubrzanja

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

Slika 5. Grafik zavisnosti ubrzanja izvršavanja problema 2 od ulaznog parametara za N = 4 niti

Slika 6. Grafik zavisnosti ubrzanja izvršavanja problema 2 od ulaznog parametara za N = 2 niti

Slika 7. Grafik zavisnosti ubrzanja izvršavanja problema 2 od ulaznog parametara za N = 1 niti

Slika 8. Grafik zavisnosti ubrzanja izvršavanja problema 2 od ulaznog parametara za 4, 2 i 1 nit

* + 1. Diskusija dobijenih rezultata

Kao što možemo videti iz priloženih logova i grafika, a i kao sto je očekivano, ubrzanje raste srazmerno sa porastom broja procesa. Najoptimalnije performanse dobijaju sa kada je broj procesa jednak 4 i tada je ubrzanje između 3.5 i 4, u zavisnosti od veličine podataka nad kojim se algoritam izvršava mogu se videti mala odstupanja. Sa smanjenjem broja procesa opada i ubrzanje, pa je ubrzanje koje dobijemo koristeći 2 procesa nešto manje i iznosi nekih 1.5 i 2, dok ubrzanje sa jednom niti ne postoji.

1. Problem 3

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

* 1. Tekst problema

Paralelizovati jednostavan program koji se bavi molekularnom dinamikom. Kod predstavlja simulaciju molekularne dinamike argonovog atoma u ograničenom prozoru (prostoru) sa periodičnim graničnim uslovima. Atomi se inicijalno nalaze raspoređeni u pravilnu mrežu, a zatim se tokom simulacije dešavaju interakcije između njih. U svakom koraku simulacije u glavnoj petlji se dešava sledeće:

● Čestice (atomi) se pomeraju zavisno od njihovih brzina i brzine se parcijalno ažuriraju u pozivu funkcije domove.

● Sile koje se primenjuju na nove pozicije čestica se izračunavaju; takođe, akumuliraju se prosečna kinetička energija (virial) i potencijalna energija u pozivu funkcije forces.

● Sile se skaliraju, završava ažuriranje brzine i izračunavanje kinetičke energije u pozivu funkcije mkekin.

● Prosečna brzina čestice se računa i skaliraju temperature u pozivu funkcije velavg.

● Pune potencijalne i prosečne kinetičke energije (virial) se računaju i ispisuju u funkciji prnout.

Program se nalazi u datoteci direktorijumu MolDyn u arhivi koja je priložena uz ovaj dokument. Program se sastoji od više datoteka, od kojih su od interesa datoteke main.c i forces.c, jer se u njima provodi najviše vremena. Analizirati dati kod i obratiti pažnju na redukcione promenljive unutar datoteke forces.c. Ukoliko je potrebno međusobno isključenje prilikom paralelizacije programa, koristiti kritične sekcije ili atomske operacije.

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Kao što je u tekst zadatka naznačeno, potrebno je uraditi paralelizaciju nad funkcijom forces. Ova funkcija se sastoji od dve for petlje, gde spoljašnja for petlja prolazi kroz koordinate svakog pojedinačnog atoma, a unutrašnja prolazi kroz sve ostale atome i određuje sumu svih sila koje nastaju prilikom interakcije dva atoma. Na kraju unutrašnje for petlje dolazi do sumiranja svih sila i na osnovu sume se određuje dalje kretanje atoma. Paralelizacija je odrađena nad spoljašnjom for petljom. U main funkciji, master proces je zadužen za sve moguće ispise, kako se oni ne bi ponavljali n puta.

* + 1. Način paralelizacije

Na početku paralelizacije inicijalizovan je MPI svet u main funkciji. Zatim je inicijalizovan master proces, koji u ovom slučaju ima rank 0 i koji je zadužen ispis dobijenih rezultata. U funkciji forces dobijen je i ukupan broj procesa koji će raditi na rešavanju problema, kao i rank svih procesa.

For petlja je paralelizovana tako da se niz deli ciklično, tako da svaki proces u iteraciji uzme element niza čiji je indeks za size odnosno broj procesa veći od indeksa iz prethodne iteracije, kako bi raspodela po procesima bila uniformna. Istina je da se ovakvom podelom vrši narušavanje lokalnosti podataka, ali ispostavlja se da se ovako dobijaju najbolji rezultati. Na kraju se radi redukcija nad nizom f2. Master u niz f, koji predstavlja sumu svih sila za svaki atom pojedinačno, kopira dobijeni redukcioni niz f2. Zatim master šalje niz f svim ostalim procesima kolektivnom komunikacijom. Na kraju se rade i redukcije promenljivih vir i epot, koje master takođe šalje ostalim procesima.

* 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.

SEKVENCIJALNO IZVRŠAVANJE

./md

Time elapsed: 11.320826

Listing 9. Sekvencijalno izvršavanje problema 3

NUM\_THREADS=4

mpirun -np 4 ./md

Time elapsed: 2.841592

Listing 10. Izvršavanje problema 3 za N=4 niti

NUM\_THREADS=2

mpirun -np 2 ./md

Time elapsed: 5.45288

Listing 11. Izvršavanje problema 3 za N=2 niti

NUM\_THREADS=1

mpirun -np 1 ./md

Time elapsed: 11.228432

Listing 12. Izvršavanje problema 3 za N=1 niti

* + 1. Grafici ubrzanja

U okviru ove sekcije je dat grafik ubrzanja u odnosu na sekvencijalnu implementaciju.

Slika 9. Grafik zavisnosti ubrzanja izvršavanja problema 3 od broja niti

* + 1. Diskusija dobijenih rezultata

Kao što možemo videti iz priloženih logova i grafika, a i kao sto je očekivano, ubrzanje raste srazmerno sa porastom broja procesa. Najoptimalnije performanse dobijaju sa kada je broj procesa jednak 4 i tada je vreme izvršavanja paralelizovanog programa 4 puta manje od sekvencijalnog. U zavisnosti od veličine podataka nad kojim se algoritam izvršava mogu se videti mala odstupanja. Sa smanjenjem broja procesa opada i ubrzanje, pa je ubrzanje koje dobijemo koristeći 2 procesa nešto manje i iznosi nekih 2 puta.

1. Problem 3

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

* 1. Tekst problema

Prethodni program paralelizovati korišćenjem manager - worker modela. Proces gospodar (master) treba da učita neophodne podatke, generiše poslove, deli posao ostalim procesima i ispiše na kraju dobijeni rezultat. U svakom koraku obrade, proces gospodar šalje procesu radniku na obradu jednu jedinicu posla čiji veličinu treba pažljivo odabrati. Proces radnik prima podatke, vrši obradu, vraća rezultat, signalizira gospodaru kada je spreman da primi sledeći posao i ponavlja opisani postupak dok ne dobije signal da prekine sa radom. Veličinu jedne jedinice posla prilagoditi karakteristikama programa. Ukoliko je moguće, koristiti rutine za neblokirajuću komunikaciju za razmenu poruka. Način pokretanja programa se nalazi u datoteci run.

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Kao što je u tekst zadatka naznačeno, potrebno je uraditi paralelizaciju nad funkcijom forces. Ova funkcija se sastoji od dve for petlje, gde spoljašnja for petlja prolazi kroz koordinate svakog pojedinačnog atoma, a unutrašnja prolazi kroz sve ostale atome i određuje sumu svih sila koje nastaju prilikom interakcije dva atoma. Na kraju unutrašnje for petlje dolazi do sumiranja svih sila i na osnovu sume se određuje dalje kretanje atoma. Paralelizacija je odrađena nad spoljašnjom for petljom. U main funkciji, master proces je zadužen za sve moguće ispise, kako se oni ne bi ponavljali n puta.

* + 1. Način paralelizacije

Na početku paralelizacije inicijalizovan je MPI svet u main funkciji. Zatim je inicijalizovan master proces, koji u ovom slučaju ima rank 0 i koji je zadužen ispis dobijenih rezultata. U funkciji forces dobijen je i ukupan broj procesa koji će raditi na rešavanju problema, kao i rank svih procesa.

Kao što je u zadatku naznačeno, potrebno je uraditi paralelizaciju worker-manager modelom. Master je zadužen za učitavanje kao i ispis podataka. Takođe obaveza master je da dinamički šalje slave-ovima zadatake, odnosno u ovom slučaju indeks iteracije koje trebaju izvršiti. Proces slave prima podatkak, vrši obradu, vraća rezultate, čeka da mu master pošalje sledeći posao i ponavlja opisani postupak dok ne dobije signal da prekine sa radom. Na kraju, master šalje kolektivno komunikacijom dobijene rezultate.

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 4.

* + 1. Logovi izvršavanja

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

SEKVENCIJALNO IZVRŠAVANJE

./md

Time elapsed: 11.320826

Listing 13. Sekvencijalno izvršavanje problema 4

NUM\_THREADS=4

mpirun -np 4 ./md

Time elapsed: 11.102301

Listing 14. Izvršavanje problema 4 za N=4 niti

NUM\_THREADS=2

mpirun -np 2 ./md

Time elapsed: 21.919125

Listing 15. Izvršavanje problema 4 za N=2 niti

* + 1. Grafici ubrzanja

U okviru ove sekcije je dat grafik ubrzanja u odnosu na sekvencijalnu implementaciju.

Slika 10. Grafik zavisnosti ubrzanja izvršavanja problema 4 od broja niti

* + 1. Diskusija dobijenih rezultata

U ovom primeru možemo videti da nije dobijeno uspešno ubrzanje. Paralelizovan kod koji se izvršava na 4 procesora ne daje optimalnije rešenje od sekvencijalnog koda, dok se program koji izvršava 2 procesa izvršava čak duplo duže. Zaključak je da je rešenje neefikasno i da zasigurno sadrži propuste na koje nije uticano, te da sigurno postoji optimalnije rešenje. Još jedan od mogućih uzroka ove pojave je veliki overhead. Ukoliko se kao vid komunikacije izabere neblokirajuća komunikacija, performanse su neznatno bolje, što predstavlja još jedan propust.