Univerzitet u Beogradu - Elektrotehnički fakultet

Multiprocesorki sistemi (13S114MUPS, 13E114MUPS)



Domaći zadatak 1 – OPENMP

Izveštaj o urađenom domaćem zadatku

|  |  |
| --- | --- |
| Predmetni saradnici: | Studenti: |
| dipl. ing. Matija Dodović | Ljubica Majstorović 2020/0253  Pavle Šarenac 2020/0359 |

Beograd, novembar 2023.

Sadržaj

[Sadržaj 2](#_Toc150300935)

[1. Problem 1 – Izračunavanje aritmetičkih brojeva korišćenjem direktive za podelu posla (for direktive) 4](#_Toc150300936)

[1.1. Tekst problema 4](#_Toc150300937)

[1.2. Delovi koje treba paralelizovati 4](#_Toc150300938)

[1.2.1. Diskusija 4](#_Toc150300939)

[1.2.2. Način paralelizacije 6](#_Toc150300940)

[1.3. Rezultati 7](#_Toc150300941)

[1.3.1. Logovi izvršavanja 7](#_Toc150300942)

[1.3.2. Grafici ubrzanja 9](#_Toc150300943)

[1.3.3. Diskusija dobijenih rezultata 10](#_Toc150300944)

[2. Problem 2 – Izračunavanje aritmetičkih brojeva korišćenjem koncepta poslova (tasks) 12](#_Toc150300945)

[2.1. Tekst problema 12](#_Toc150300946)

[2.2. Delovi koje treba paralelizovati 12](#_Toc150300947)

[2.2.1. Diskusija 12](#_Toc150300948)

[2.2.2. Način paralelizacije 12](#_Toc150300949)

[2.3. Rezultati 12](#_Toc150300950)

[2.3.1. Logovi izvršavanja 12](#_Toc150300951)

[2.3.2. Grafici ubrzanja 12](#_Toc150300952)

[2.3.3. Diskusija dobijenih rezultata 12](#_Toc150300953)

[3. Problem 3 – Generisanje elemenata Halton Quasi Monte Carlo sekvence bez direktiva za podelu posla (worksharing direktiva) 13](#_Toc150300954)

[3.1. Tekst problema 13](#_Toc150300955)

[3.2. Delovi koje treba paralelizovati 13](#_Toc150300956)

[3.2.1. Diskusija 13](#_Toc150300957)

[3.2.2. Način paralelizacije 13](#_Toc150300958)

[3.3. Rezultati 13](#_Toc150300959)

[3.3.1. Logovi izvršavanja 13](#_Toc150300960)

[3.3.2. Grafici ubrzanja 13](#_Toc150300961)

[3.3.3. Diskusija dobijenih rezultata 13](#_Toc150300962)

[4. Problem 4 – Generisanje elemenata Halton Quasi Monte Carlo sekvence sa direktivama za podelu posla (worksharing direktivama) 14](#_Toc150300963)

[4.1. Tekst problema 14](#_Toc150300964)

[4.2. Delovi koje treba paralelizovati 14](#_Toc150300965)

[4.2.1. Diskusija 14](#_Toc150300966)

[4.2.2. Način paralelizacije 14](#_Toc150300967)

[4.3. Rezultati 14](#_Toc150300968)

[4.3.1. Logovi izvršavanja 14](#_Toc150300969)

[4.3.2. Grafici ubrzanja 14](#_Toc150300970)

[4.3.3. Diskusija dobijenih rezultata 14](#_Toc150300971)

[5. Problem 3 – Generisanje elemenata Halton Quasi Monte Carlo sekvence sa direktivama za podelu posla (worksharing direktivama) 15](#_Toc150300972)

[5.1. Tekst problema 15](#_Toc150300973)

[5.2. Delovi koje treba paralelizovati 15](#_Toc150300974)

[5.2.1. Diskusija 15](#_Toc150300975)

[5.2.2. Način paralelizacije 15](#_Toc150300976)

[5.3. Rezultati 15](#_Toc150300977)

[5.3.1. Logovi izvršavanja 15](#_Toc150300978)

[5.3.2. Grafici ubrzanja 15](#_Toc150300979)

[5.3.3. Diskusija dobijenih rezultata 15](#_Toc150300980)

1. Problem 1 – Izračunavanje aritmetičkih brojeva korišćenjem direktive za podelu posla (for direktive)
   1. Tekst problema

Paralelizovati program koji vrši izračunavanje aritmetičkih brojeva. Pozitivan ceo broj je aritmetički ako je prosek njegovih pozitivnih delilaca takođe ceo broj. Program se nalazi u datoteci aritmetic.c u arhivi koja je priložena uz ovaj dokument. Obratiti pažnju na ispravno deklarisanje svih promenljivih prilikom paralelizacije. Program testirati sa parametrima koji su dati u run skripti.

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Segment koda nad kojim je izvršena paralelizacija u ovom problemu je sledeći:

A computer screen shot of a code

Description automatically generated

Ova petlja je vrlo pogodna za paralelizaciju zato što ne postoji nikakva zavisnost po podacima između njenih iteracija, pa one mogu da se izvršavaju potpuno nezavisno.

Jedina nezgodna stvar je što petlja nije u zadovoljavajućoj formi koju for direktiva zahteva, pa je potrebno malo preurediti ovaj kod kako bi bila moguća takva paralelizacija.

Još jedan deo koda koji troši značajan deo procesorskog vremena:

A black screen with white text

Description automatically generated

Zaključili smo da nije dobro pokušati paralelizaciju ove petlje iz više razloga:

1. Ona se izvršava u telu funkcije divisor\_count\_and\_sum koja se poziva u svakoj iteraciji for petlje u okviru koje se poziva ova funkcija, pri čemu ova petlja može imati veoma veliki broj iteracija i zato bismo u tom slučaju isto toliko puta pravili paralelni region, što bi bili ogromni režijski troškovi i diskutabilno je da li bi se uopšte postiglo ubrzanje u odnosu na sekvencijalni kod u tom slučaju.
2. Postoji zavisnost po podacima između iteracija spoljašnje petlje zato što se „n“ koje je u uslovu spoljašnje petlje menja u unutrašnjoj petlji, pa zato ovako napisan algoritam nije pogodan za paralelizaciju. Da bi paralelizacija potencijalno bila moguća, bilo bi potrebno kompletno restrukturiranje ovog dela algoritma.
   * 1. Način paralelizacije

A computer screen shot of a program

Description automatically generated

Zaključili smo da nije moguće našu glavnu for petlju paralelizovati pomoću for direktive zato što forma petlje nije odgovarajuća pošto nije poznat broj iteracija unapred.

Međutim, smislili smo način da ipak upotrebimo for direktivu. Rešenje leži u činjenici da ako želimo da nađemo npr. prvih 10 000 aritmetičkih brojeva, mi zasigurno moramo da imamo najmanje 10 000 iteracija petlje. Ne znamo unapred za koliko će broj iteracija petlje biti veći od 10 000, ali znamo da ih mora biti najmanje 10 000. Ovo možemo iskoristiti onda tako što ćemu napisati for petlju koja će imati 10 000 iteracija, i u njoj je onda poznat broj iteracija unapred pa se može paralelizovati pomoću for direktive. Uslov prvobitne for petlje ćemo izmestiti u novu spoljašnju while petlju. Recimo da se u 10 000 iteracija for petlje našlo 8 000 aritmetičkih brojeva. To znači da for petlja u narednoj iteraciji while petlje mora da se izvrši još najmanje 2 000 puta jer je toliko aritmetičkih brojeva preostalo da se pronađe. Ovaj postupak ponavljamo sve dok se ne pronađe željeni broj aritmetičkih brojeva. Na ovaj način smo u svakoj iteraciji spoljašnje while petlje menjali broj iteracija for petlje tako da se uvek znao unapred taj broj, što nam je omogućilo paralelizaciju for direktivom.

Naravno, uočili smo da promenljive arithmetic\_count i composite\_count modifikuju sve niti i da se one samo inkrementiraju. Zato smo ove promenljive deklarisali kao redukcione kako bi svaka nit imala svoju kopiju ovih promenljivih i menjala nju, a zatim da se na kraju sve te lokalne kopije saberu i dodaju na inicijalne vrednosti promenljivih.

Što se tiče raspodele posla između niti, zaključili smo da posao koji niti izvršavaju postaje sve teži kako se odmiče ka kasnijim iteracijama naše paralelizovane for petlje zato što je onda vrednost parametra n koji se prosleđuje funkciji divisor\_count\_and\_sum sve veća, a samim tim će i najzahtevnija for petlja u toj funkciji imati više iteracija pošto se to isto n nalazi u uslovu te petlje. Zato je očigledno da nije pogodna podrazumevana schedule(static) raspodela pošto balans opterećenja nije idealan. Najlogičnije nam je bilo da je za ovu situaciju najbolja schedule(guided) raspodela zato što ona upravo početnim nitima dodeljuje veći broj iteracija (što potpuno ima smisla raditi pošto su ranije iteracije manje zahtevne od kasnijih), a onda se broj iteracija koji se dodeljuje kasnijim nitima eksponencijalno smanjuje. Na taj način se ostvaruje bolji balans opterećenja. Isprobavanjem static, dynamic i guided raspodela sa raznim parametrima raspodele smo primetili da su rezultati praktično isti, tako da smo se odlučili za podrazumevanu schedule(guided) raspodelu bez ikakvog parametra jer nema značajne razlike u performansama.

* 1. Rezultati

U okviru ove sekcije su izloženi rezultati paralelizacije problema 1.

* + 1. Logovi izvršavanja

Sequential implementation execution time: 0.002867s

Parallel implementation (one thread) execution time: 0.001502s

Test PASSED

Parallel implementation (two threads) execution time: 0.000306s

Test PASSED

Parallel implementation (four threads) execution time: 0.000207s

Test PASSED

Parallel implementation (eight threads) execution time: 0.000180s

Test PASSED

Listing 1. Izvršavanje komande ./arithmetic 10000

Sequential implementation execution time: 0.010243s

Parallel implementation (one thread) execution time: 0.008363s

Test PASSED

Parallel implementation (two threads) execution time: 0.004207s

Test PASSED

Parallel implementation (four threads) execution time: 0.002238s

Test PASSED

Parallel implementation (eight threads) execution time: 0.001501s

Test PASSED

Listing 1. Izvršavanje komande ./arithmetic 100000

Sequential implementation execution time: 0.172909s

Parallel implementation (one thread) execution time: 0.171841s

Test PASSED

Parallel implementation (two threads) execution time: 0.086988s

Test PASSED

Parallel implementation (four threads) execution time: 0.044931s

Test PASSED

Parallel implementation (eight threads) execution time: 0.028311s

Test PASSED

Listing 1. Izvršavanje komande ./arithmetic 1000000

Sequential implementation execution time: 4.199454s

Parallel implementation (one thread) execution time: 4.193153s

Test PASSED

Parallel implementation (two threads) execution time: 2.085609s

Test PASSED

Parallel implementation (four threads) execution time: 1.071954s

Test PASSED

Parallel implementation (eight threads) execution time: 0.571603s

Test PASSED

Listing 1. Izvršavanje komande ./arithmetic 10000000

* + 1. Grafici ubrzanja

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

Slika 1. Grafik zavisnosti ubrzanja naivnog algoritma od broja aritmetičkih brojeva koji se traže za N = 1 niti

Slika 1. Grafik zavisnosti ubrzanja naivnog algoritma od broja aritmetičkih brojeva koji se traže za N = 2 niti

Slika 1. Grafik zavisnosti ubrzanja naivnog algoritma od broja aritmetičkih brojeva koji se traže za N = 4 niti

Slika 1. Grafik zavisnosti ubrzanja naivnog algoritma od broja aritmetičkih brojeva koji se traže za N = 8 niti

* + 1. Diskusija dobijenih rezultata

Kako bi grafici i naši rezultati bili precizniji, zabeležili smo rezultate kako sekvencijalnog, tako i paralelnog izvršavanja programa za sve parametre i za svaki broj niti 5 puta i uzimali prosečna vremena.

Iz priloženih grafika je očigledno da za svaki ulazni parametar, tj. broj aritmetičkih brojeva koji se traže, dobijamo veće ubrzanje ukoliko povećamo broj niti koje rade u paraleli. Ovo je i očekivano upravo zato što su iteracije naše for petlje koju paralelizujemo potpuno nezavisne i nemamo gotovo nikakvu komunikaciju, a ni sinhronizaciju između niti pri paralelizaciji ove petlje. Jedina međusobna interakcija između niti jeste na kraju paralelnog regiona kada se redukcione promenljive od svih niti saberu, ali osim toga, nikakve interakcije između niti nema. Zato se slobodno možemo usuditi da nazovemo ovaj problem *„embarrasingly parallel”.*

Međutim, vidi se da, iako se ubrzanje povećava kako povećavamo broj niti od jedan do osam za sve ulazne parametre, stepen povećanja ubrzanja se blago smanjuje – sa povećanjem sa jedne na dve niti, ubrzanje za parametar 10 000 se upetostručilo, dok se za ostale parametre udvostručilo. Nakon toga vidimo da sa povećanjem sa četiri na osam niti, ubrzanje za parametar 10 000 se povećalo sa 13.93 na 15.4 puta, dok se za ostale ulazne parametre ono povećalo ali manje od 2 puta kao što je to bio slučaj pri prelascima sa jedne na dve, i sa dve na četiri niti. Ovo nam govori da definitivno iako povećanje broja niti poboljšava performanse, ta ubrzanja nisu neograničena. Uverili smo se u ovo tako što smo izvršili naš program čak i sa šesnaest niti (pošto računar koji smo koristili ima šesnaest jezgara pa je moguć paralelizam sa šesnaest niti), i zaista, ubrzanje se vrlo malo povećalo u odnosu na ubrzanje sa osam niti.

Interesantan trend koji se takođe odmah primećuje za svaki broj niti u programu jeste da je ubrzanje pri ulaznom parametru od 10 000 drastično veće od ubrzanja za sve ostale ulazne parametre od 100 000, 1 000 000 i 10 000 000. Ubrzanja za ova preostala tri ulazna parametra su poprilično slična za svaki broj niti.

Još jedna vrlo interesantna pojava koja se može primetiti jeste da pri izvršenju paralelne implementacije programa sa jednom niti imamo ubrzanje koje je skoro duplo za ulazni parametar 10 000, za parametar 100 000 je ubrzanje dosta manje, oko 1.1, dok za parametre 1 000 000 i 10 000 000 praktično nema ubrzanja jer je ono nešto malo preko 1.0. Zaključili smo da je ovo slučaj zato što smo for petlju iz sekvencijalne implementacije izmenili u paralelnoj implementaciji tako da se zna broj iteracija petlje unapred svaki put kada se ona izvršava. Smatramo da je to onda omogućilo gcc prevodiocu koji smo koristili za prevođenje našeg C programa da izvrši određene optimizacije pri prevođenju te petlje, pa zato dobijamo ubrzanje čak i kada se paralelni program izvršava sa jednom niti. Sa grafika se može zaključiti da ove optimizacije više utiču na ubrzanje programa kada je broj iteracija petlje manji, tj. za manje vrednosti ulaznog parametra.

1. Problem 2 – Izračunavanje aritmetičkih brojeva korišćenjem koncepta poslova (tasks)
   1. Tekst problema

Rešiti prethodni problem korišćenjem koncepta poslova (tasks). Obratiti pažnju na eventualnu potrebu za sinhronizacijom i granularnost poslova. Program testirati sa parametrima koji su dati u run skripti.

* 1. Delovi koje treba paralelizovati
     1. Diskusija

Važi sve isto kao u problemu 1 jer se radi zaista o istom problemu, samo će sada biti urađen na drugačiji način.

* + 1. Način paralelizacije

A screenshot of a computer program

Description automatically generated

U ovom problemu je paralelizacija urađena pomoću koncepta taskova, pošto je to eksplicitno zahtevano.

* 1. Rezultati
     1. Logovi izvršavanja
     2. Grafici ubrzanja
     3. Diskusija dobijenih rezultata

1. Problem 3 – Generisanje elemenata Halton Quasi Monte Carlo sekvence bez direktiva za podelu posla (worksharing direktiva)
   1. Tekst problema
   2. Delovi koje treba paralelizovati
      1. Diskusija
      2. Način paralelizacije
   3. Rezultati
      1. Logovi izvršavanja
      2. Grafici ubrzanja
      3. Diskusija dobijenih rezultata
2. Problem 4 – Generisanje elemenata Halton Quasi Monte Carlo sekvence sa direktivama za podelu posla (worksharing direktivama)
   1. Tekst problema
   2. Delovi koje treba paralelizovati
      1. Diskusija
      2. Način paralelizacije
   3. Rezultati
      1. Logovi izvršavanja
      2. Grafici ubrzanja
      3. Diskusija dobijenih rezultata
3. Problem 5 – Generisanje elemenata Halton Quasi Monte Carlo sekvence sa direktivama za podelu posla (worksharing direktivama)
   1. Tekst problema
   2. Delovi koje treba paralelizovati
      1. Diskusija
      2. Način paralelizacije
   3. Rezultati
      1. Logovi izvršavanja
      2. Grafici ubrzanja
      3. Diskusija dobijenih rezultata