Multiprocesorski sistemi

Domaći zadatak 2
Školska godina 2025/2026
MPI – komunikacija, izvedeni tipovi, grupe i komunikatori
(10 poena)

Uvod

Cilj zadatka je da studente obuči da samostalno podese **`MPI`** okruženje i razvijaju osnovne **`MPI`** programe korišćenjem rutina za pojedinačnu i kolektivnu komunikaciju, izvedenih tipova podataka, grupa i komunikatora.

Domaći zadaci se rade **samostalno** ili **u paru**. Rešenja domaćih zadataka i izveštaj svaki student predaje **samostalno** u svoj svn repozitorijum.

Podešavanje okruženja

Podešavanja okruženja izvršiti prema uputstvima koja se nalaze u dokumentu za laboratorijsku vežbu 2 - MPI. Obratiti pažnju na razlike koje postoje kod podešavanja za prevođenje na 32-bitnim i 64-bitnim računarskim sistemima. Alternativno, koristiti OpenMPI na računaru **`rtidev5.etf.rs`**.

Izveštaj

Uz predati domaći zadatak (izvorne kodove) treba napisati i priložiti kratak izveštaj o izvršenoj paralelizaciji i dobijenim ubrzanjima u odnosu na sekvencijalnu verziju koda. Za svaki rešeni zadatak treba kratko opisati uočena mesta koja je moguće paralelizovati i način paralelizacije. Takođe, potrebno je dati logove izvršenog koda za sve test primere koji se izvršavaju i nalaze se u **`run`** datoteci i nacrtati grafike ubrzanja u odnosu na sekvencijalnu verziju. Na graficima je potrebno dati i rezultate poređenja različitih načina paralelizacije za isti broj niti, ukoliko postoje takvi zahtevi u okviru teksta zadatka. Šablon za pisanje izveštaja se nalazi u okviru sekcije za domaće zadatke predmetnog sajta.

Zadaci

Svaki od programa treba napisati tako da može biti izvršen sa bilo kojim od broja procesa iz opsega navedenog iza postavke zadatka. **`N`** označava maksimalan mogući broj procesa u trenutno dostupnom okruženju. Za programe koji će biti izvršavani na samo jednom računaru, pretpostaviti da važi **`N=16`**. Svaki program treba da vrši proveru da li je broj procesa tekućeg izvršavanja odgovarajući postavci zadatka. U slučaju da to nije zadovoljeno, prekinuti izvršavanje.

Kod zadataka gde je to zahtevano, korisnik zadaje samo dimenzije problema/nizova/matrica, a sve potrebne ulazne podatke generisati u operativnoj memoriji uz pomoć generatora pseudoslučajnih brojeva iz biblioteke jezika C. Generisani brojevi treba da budu odgovarajućeg tipa u opsegu od **`-MAX`** do **`+MAX`**, gde **`MAX`** ima vrednost 1024. Za sve zadatke je potrebno napisati ili iskoristiti zadatu sekvencijalnu implementaciju odgovarajućeg problema. U cilju postizanja najboljih performansi, dozvoljeno je menjati izvorni sekvencijalni kod. Izvorni kod, ili njegova modifikacija koja postiže najbolje performanse, je potrebno koristiti kao referentnu (*gold*) implementaciju prilikom testiranja programa.

Svaki program treba da:

- Generiše ili koristi već obezbeđene ulazne test primere.
- Izvrši sekvencijalnu implementaciju nad zadatim test primerom.
- Izvrši paralelnu implementaciju nad zadatim test primerom.
- Ispište vreme izvršavanja sekvencijalne i paralelne implementacije problema.
- Uporedi rezultat sekvencijalne i OpenMP implementacije problema.

- Ispiše "**`Test PASSED`**" ili "**`Test FAILED`**" u zavisnosti da li se rezultat izvršavanja paralelizovane implementacije podudara sa rezultatom izvršavanja sekvencijalne implementacije.

Poređenje rezultata paralelizovane i sekvencijalne implementacije problema izvršiti na kraju sekvencijalnog dela programa. Kod zadataka koji koriste realne tipove (**`float`**, **`double`**) tolerisati maksimalno odsupanje od **`±ACCURACY`** prilikom poređenja rezultata sekvencijalne i OpenMP implementacije. Smatrati da konstanta **`ACCURACY`** ima vrednost 0.01 ili usvojiti razumnu vrednost u skladu sa problemom koji se rešava. **Prilikom rešavanja zadataka voditi računa da se postigne maksimalni mogući paralelizam.** Dozvoljeno je ograničeno preuređivanje dostupnih sekvencijalnih implementacija prilikom paralelizacije. **Ukoliko u nekom zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku i da nastavi da izgrađuje svoje rešenje na temeljima uvedene pretpostavke.**

Dostupne sekvencijalne implementacije se nalaze u arhivi koje se mogu preuzeti na adresi sa predmetnog sajta. Na **`rtidev5.etf.rs`** računaru arhiva se može dohvatiti i raspakovati sledećim komandama:

Dohvatanje: **`wget http://mups.etf.rs/dz/2025-2026/MPS_DZ_2025_2026.zip`**

Raspakivanje: **`unzip MPS_DZ_2025_2026.zip`**

1. **[2]** Paralelizovati program koji formira sliku tačaka koje pripadaju Julia skupu tačaka (https://en.wikipedia.org/wiki/Julia_set). Neka se posmatra skup tačaka *(x, y)* u na pravougaonom domenu $x, y \in [-1{,}5, 1.5]$ i neka važi $z = x+yi$. Julia skup je skup tačaka za koji iteracija $z = z^2 + c$ ne divergira za određene zadate početne uslove. U zadatom programu početni uslov odgovara $c=-0.8+0.156i$. Ukoliko u bilo kom trenutku važi $1000 < |z|$, smatra se da tačka z ne pripada Julia skupu. Program formira sliku u *Targa* (*.tga*) formatu koja se može otvoriti u nekom od namenskih pregledača slika. Program se nalazi u datoteci **`julia.c`** u arhivi koja je priložena uz ovaj dokument, dok se primeri izlaznih datoteka nalaze u direktorijumu **`output`**.

   Program testirati sa parametrima koji su dati u **`run`** skripti.

   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.

2. **[2]** 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 **`run`** skripti.

   Ukoliko je moguće, koristiti rutine za neblokirajuću komunikaciju za razmenu poruka.

3. **[3]** Rešiti prethodni problem korišćenjem jednostrane komunikacije.

   Program testirati sa parametrima koji su dati u **`run`** skripti.

4. **[3]** Paralelizovati jednostavan program koji se bavi molekularnom dinamikom. Simulacija se bavi česticama (molekulima) i njihovom međusobnom interakcijom u funkciji distance. Čestice nisu prostorno ograničene, a algoritam iterativno rešava sistem diferencijalnih jednačina u diskretnim vremenskim koracima. Na osnovu zadate pozicije i brzine čestice u jednom trenutku, algoritam određuje poziciju i brzinu u narednom trenutku. Kod koji treba paralelizovati se nalazi u datoteci **`md.c`** u arhivi koja je priložena uz ovaj dokument. Analizirati dati kod i obratiti pažnju na funkciju **`compute`** za izračunavanje sila i energija. Ukoliko je potrebno međusobno isključenje prilikom paralelizacije programa, koristiti dostupne OpenMP sinhornizacione direktive. Obratiti pažnju na efikasnost paralelizacije.

Program paralelizovati korišćenjem manager - worker modela. Proces gospodar (master) treba da učita neophodne podatke, generiše poslove, deli posao ostalim procesima (worker) 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.

Program testirati sa parametrima koji su dati u **`run`** skripti.