# Multiprocesorski sistemi

Domaći zadatak 1
OpenMP – paralelizacija direktivama
(10 poena)

## Uvod

Cilj prvog domaćeg zadatka je da studentima približi osnovne koncepte rada sa **`OpenMP`** tehnologijom koja omogućava paralelizaciju direktivama na sistemima sa deljenom memorijom.

## Podešavanje okruženja

Za rad sa OpenMP tehnologijom koristiti **`gcc`**/**`g++`** na računaru **`rtidev5.etf.rs`** ili instalirati **`gcc`**/**`g++`** prevodilac na lokalnu Windows mašinu korišćenjem Cygwin ili MinGW alata. Za rešavanje domaćeg zadatka je potrebno imati **`gcc`**/**`g++`** prevodilac verzije 4.4.0 ili noviji koji podržava OpenMP standard 3.0.

## Zadaci

Svaki od programa treba napisati i paralelizovati tako da može biti izvršen sa bilo kojim brojem niti iz opsega navedenog iza postavke zadatka. **`N`** označava maksimalan mogući broj procesa u trenutno dostupnom OpenMP izvršnom okruženju. Za programe koji će biti izvršavani na samo jednom računaru, smatrati da **`N`** neće biti više od **`N=8`**. Preporučuje se testiranje zadataka sa 1, 2, 4 i 8 niti.

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 slučajnih 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 koja će biti korišćena kao referentna (*gold*) implementacija prilikom testiranja programa. Dostupne sekvencijalne implementacije se nalaze u arhivi **`MPS_DZ1_OpenMP.zip`** ili **`MPS_DZ1_OpenMP.tar`** koje se mogu preuzeti na adresi http://mups.etf.rs/dz/2013-2014/. Dozvoljeno je ograničeno preuređivanje dostupnih sekvencijalnih implementacija prilikom paralelizacije.

Svaki program treba da:

- Generiše ulazne test primere.
- Izvrši sekvencijalnu implementaciju nad zadatim test primerom.
- Izvrši paralelnu, OpenMP implementaciju nad zadatim test primerom.
- Uporedi rezultat sekvencijalne i OpenMP implementacije problema.
- Ispiše "**`Test PASSED`**" ili "**`Test FAILED`**" u zavisnosti da li se rezultat izvršavanja OpenMP implementacije podudara sa rezultatom izvršavanja sekvencijalne implementacije.

Poređenje rezultata OpenMP 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. **Prilikom rešavanja zadataka voditi računa da se postigne maksimalni mogući paralelizam.**

1. Paralelizovati program koji određuje vrednost broja π. Vrednost broja π se može statistički odrediti na više načina uz pomoć generatora pseudoslučajnih brojeva uniformne raspodele. Jedan od načina je generisanje tačaka u ravni sa obema koordinatama u realnom opsegu [0,1]. Vrednost broja π tada može biti određena na osnovu odnosa broja tačaka koje se nalaze u delu kruga poluprečnika 1 sa centrom u koordinatnom početku i broja tačaka koje pripadaju kvadratu stranice 1 koji obuhvata sve generisane tačke. Program se nalazi u datoteci **`piCalculation.c`** u arhivi koja je priložena uz ovaj dokument. Prilikom paralelizacije nije dozvoljeno koristiti direktive za podelu posla (*worksharing* direktive), već je iteracije petlje koja se paralelizuje potrebno raspodeliti ručno. Obratiti pažnju na ispravno deklarisanje svih promenljivih prilikom paralelizacije. [1, N]

2. Paralelizovati program koji vrši množenje dve kvadratne matrice realnih brojeva u dvostrukoj preciznosti. Program se nalazi u datoteci **`matMul.c`** u arhivi koja je priložena uz ovaj dokument. Paralelizaciju obaviti korišćenjem direktiva za podelu posla (*worksharing* direktive). Pored operacija množenja matrice, potrebno je paralelizovati i operacije za popunjavanje matrice podacima. Koristiti funkciju koja matricu uniformno popunjava pseudoslučajnim sadržajem. [1, N]
3. Paralelizovati prethodni program uz korišćenje funkcije koja jednu od matrica popunjava kao gornje trougaonu matricu. Paralelizaciju obaviti korišćenjem direktiva za podelu posla (*worksharing* direktive). Obratiti pažnju na performanse programa i balansiranje opterećenja prilikom raspodele iteracija nitima. [1, N]
4. Rešiti problem množenja matrica korišćenjem koncepta poslova (*tasks*). Koristiti funkciju koja matricu uniformno popunjava pseudoslučajnim sadržajem. Prilikom definisanja granularnosti poslova, dati rešenje sa granularnošću na nivou elementa rezultujuće matrice ili na nivou kolone rezultujuće matrice. [1, N]
5. 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:

   a) Čestice (atomi) se pomeraju zavisno od njihovih brzina i brzine se parcijalno ažuriraju u pozivu funkcije **`domove`**.
   b) Sile koje se primenjuju na nove pozicije čestica se izračunavaju; takođe, akumuliraju se prosečna kinetička energija (*virial*) i potencijalna energija u pozivu funkcije **`forces`**.
   c) Sile se skaliraju, završava ažuriranje brzine i izračunavanje kinetičke energije u pozivu funkcije **`mkekin`**.
   d) Prosečna brzina čestice se računa i skaliraju temperature u pozivu funkcije **`velavg`**.
   e) 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 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, N]

**Važno:** 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.