Multiprocesorski sistemi

Domaći zadatak 2
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.

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`**.

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 MPI okruženju. Za programe koji će biti izvršavani na samo jednom računaru, pretpostaviti da važi **`N=4`**. Svaki program treba da vrši proveru da li je broj procesa tekućeg izvršavanja odgovarajući postavci zadatka. U slučaju da to nije zadovoljeno, prekinuti izvršavanje korišćenjem MPI poziva **`Abort`**.

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 koja će biti korišćena kao referentna (*gold*) implementacija prilikom testiranja programa.

Svaki program treba da:

- Generiše ili koristi već obezbeđene ulazne test primere.
- Izvrši MPI implementaciju nad zadatim test primerom.
- Izvrši sekvencijalnu implementaciju nad zadatim test primerom.
- Ispište vreme izvršavanja sekvencijalne i paralelne implementacije problema.
- Uporedi rezultat MPI i sekvencijalne implementacije problema.
- Ispiše "**`Test PASSED`**" ili "**`Test FAILED`**" u zavisnosti da li se rezultat izvršavanja MPI implementacije podudara sa rezultatom izvršavanja sekvencijalne implementacije.

Poređenje rezultata MPI i sekvencijalne implementacije problema izvršiti unutar procesa sa rangom 0. Kod zadataka koji koriste realne tipove (**`float`**, **`double`**) tolerisati maksimalno odsupanje od **`±ACCURACY`** prilikom poređenja rezultata CPU i MPI implementacije. Smatrati da konstanta **`ACCURACY`** ima vrednost 0.01. **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 **`MPS_DZ2_MPI.zip`** ili **`MPS_DZ2_MPI.tar.bz2`** koje se mogu preuzeti na adresi http://mups.etf.rs/dz/2015-2016/. Na **`rtidev5.etf.rs`** računaru arhiva se može dohvatiti i raspakovati sledećim komandama:

Dohvatanje: **`wget http://mups.etf.rs/dz/2015-2017/MPS_DZ2_MPI.tar.bz2`**

Raspakivanje: **`tar xjvf MPS_DZ2_MPI.tar.bz2`**

1. Paralelizovati program koji vrši množenje dve kvadratne matrice realnih brojeva u dvostrukoj preciznosti. 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 se nalazi u datoteci **`matMul.c`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**. [1, N]
2. Paralelizovati prethodni program korišćenjem principa *N-version* programiranja (http://en.wikipedia.org/wiki/N-version_programming). Procese treba podeliti u dve približno jednake grupe i dva komunikatora. Proces sa rangom 0 ne treba da učestvuje ni u jednoj novostvorenoj grupi i ne učestvuje u izračunavanju, već samo učitava podatke i raspodeljuje ih procesima sa rangom 0 u novoformiranim grupama. U jednoj grupi, izvršiti blokovsku dekompoziciju jedne od ulaznih matrica po vrstama. U drugoj grupi, izvršiti blokovsku dekompoziciju jedne od ulaznih matrica po kolonama. Proces sa rangom 0 u MPI svetu treba da sakupi i uporedi dobijene vrednosti. Za razmenu podataka, koristiti rutine za kolektivnu komunikaciju. Program se nalazi u datoteci **`matMul.c`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**. [3, N]
3. Paralelizovati program koji rešava *pathfinder* problem. *Pathfinder* problem koristi dinamičko programiranje da pronađe put sa najmanjom akumuliranom težinom u 2D mreži (matrici). Put započinje od početne i ide do krajnje vrste matrice krećući se preko elemenata sa najmanjom težinom. U svakom koraku, put se pomera ili pravo ili dijagonalno u odnosu na trenutni element. Proces sa rangom 0 treba da učita ulazne podatke, raspodeli posao ostalim procesima, na kraju prikupi dobijene rezultate i ravnopravno učestvuje u obradi. Svakom procesu dodeliti određen broj kolona matrice na obradu. Za slanje jedne kolone matrice koristiti odgovarajući izvedeni tip. Program se nalazi u datoteci **`pathfinder.cpp`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**. [1, N]
4. Prethodni program rešiti korišćenjem rutina za neblokirajuću komunikaciju za razmenu graničnih elemenata. Dok razmenjuju granične elemente, procesi treba da računaju vrednost unutrašnjih elemenata u tekućoj iteraciji. Program se nalazi u datoteci **`pathfinder.cpp`** u arhivi koja je priložena uz ovaj dokument. Program testirati sa parametrima koji su dati u datoteci **`run`**. [1, N]
5. Paralelizovati program koji vrši obilazak grafa po širini. Zadatak 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 jedan neobrađen čvor na obradu. Proces radnik prima čvor koji treba da obiđe, vrši obilazak, vraća rezultat, signalizira gospodaru kada je spreman da primi sledeći posao i ponavlja opisani postupak dok ne dobije signal da prekine sa radom. Program se nalazi u datoteci **`bfs.cpp`** u arhivi koja je priložena uz ovaj dokument. Ulazni test primeri se nalaze u direktorijumu **`data`**, a način pokretanja programa u datoteci **`run`**. [2,N]