Laborator 12

Tutorial llnl
MPI The complete Reference

Exerciții

Pentru fiecare exercițiu se va scrie în fișierul REPORT.txt rezultatul rulărilor și răspunsul la întrebări. Asigurați-vă că sunt afișate mesaje reprezentative pentru fiecare transmisie (și la send și recv).

1. (**1_nonBlocking.c**) Implementați un program MPI ce are două procese.
   - Procesul cu rank 0 trimite procesului cu rank 1 un vector folosind funcții **non-blocante**. Afișați vectorul după primire.
   - Scrieți în *REPORT.txt* ce se întâmplă dacă imediat dupa send, în procesul cu rank 0 este modificat vectorul.
   - Scrieți în *REPORT.txt* ce se întâmplă dacă modificarea făcută în vector este la o poziție foarte mare 100.000+.
2. (**2_sendIsBlocking.c**) Pornind de la programul dat arătați că MPI_Send poate fi și blocant și non-blocant. Notați cum ați determinat acest lucru în *REPORT.txt*.
3. (**3_timer.c**) Implementați un program MPI cu două procese:
   - Procesul cu rank 0 trimite după **X** (dat ca parametru programului) secunde un mesaj **non-blocant** procesului cu rank 1.
   - Procesul cu rank 1 face o primire de mesaj non blocantă.
   - Timp de 5 secunde procesul cu rank 1 va verifica dacă mesajul a venit (verificarea se face o dată pe secundă).
   - Se va testa programul cu X setat la 2,4 și 6 și se vor scrie rezultatele în REPORT.txt.
4. (**4_friends.c**) Implementați un program MPI cu 5 procese:
   - Presupunem că procesele sunt organizate în linie (rank 0 vecin cu rank 1, rank 1 vecin cu 0 și 2, și tot așa). **Este permisă comunicare doar între procesele vecine.**
   - Procesul cu rank 0 trimite un mesaj **non-blocant**.
   - Toate celelalte procese încep să primească **non-blocant** (pentru maxim 5 secunde cu verificare o dată la secundă) mesaje ce pot veni de la oricare vecin.
   - O dată primit un mesaj se așteaptă random între 0 și 20 de secunde (se afișează perioada de timp) înainte de a fi transmis tuturor vecinilor (**inclusiv cel de la care am primit mesajul**).
   - Dacă s-a primit un mesaj timer-ul de 5 secunde se resetează și după transmisie se așteaptă iar mesaje.
5. (**5_spanningTree.c**) Implementați un program MPI pentru determinarea unui spanning Tree.
   - În schelet sunt date două grafuri (unul mereu comentat. Se folosește doar unul la un moment dat și fiecare graf necesită un număr diferit de procese). Aceste grafuri reprezintă cum sunt conectate procesele. **Este permisă**

comunicare doar între procesele vecine. Matricea graf nu trebuie folosită de voi, informațiile din aceasta sunt obținute doar prin apelul funcției getNeighbors().

- Apelul funcției getNeighbors() întoarce lista vecinilor cu care este permisă comunicarea. Pe poziția 0 din acest vector este specificat numărul vecinilor + 1. Pe pozițiile începând cu 1 până la poziția specificată de elementul de la poziția 0 (exclusiv) sunt notate rank-urile vecinilor.
- Comunicația va fi non-blocantă, un mesaj va fi așteptat maxim 15 secunde și se va verifica recepția o dată la secundă.
- Un proces al cărui rank este dat ca parametru programului va fi considerat inițiator. Acesta va trimite mesajul inițial tuturor vecinilor săi.
- Toate nodurile așteaptă primirea unui mesaj. O dată primit, se va nota nodul de la care mesajul a venit ca părinte și se va trimite un mesaj fiecărui nod vecin diferit de părinte.
- O dată primit acest mesaj inițial se așteaptă răspuns de la nodurile considerate copii.
- După ce a fost primit răspuns de la toți copiii, sau timer-ul de așteptare a trecut, se va trimite ca răspuns părintelui **un singur mesaj** în care este notat părintele nodului curent alături de toate răspunsurile de la copiii. Fiind dat un vector de 100 de elemente părintele nodului curent poate fi notat pe poziția rank din vector.
- Mesajul inițial (cel care poate fi considerat conceptual probă) poate fi gol, dar pentru ca algoritmul să meargă și în cazul ciclurilor va conține int-uri și va avea o lungime fixă de 100 de elemente. Aceeași formă o vor avea și celelalte mesaje (cele ce pot fi considerate ecou).
- O dată ajuns la inițiator acesta va afișa lista de părinți.

Exercițiile de la 1 la 5 sunt **obligatorii**. Conceptele explorate sunt esențiale pentru obținerea notei **minime** de promovare.

Vă recomandăm, pentru a crește șansele de a obține o notă cât mai mare să explorați și următoarele exerciții:

6. (**6_epidemic.c**) Implementați algoritmul epidemic.
   - ATENȚIE: Un nod poate comunica doar cu vecinii. MPI vă va permite să trimiteți mesaje de la orice nod la oricare altul, dar noi vrem să simulăm o rețea reală în care se poate comunica doar cu vecinii. Astfel, nu aveți voie să faceți send sau recv, decât de la un nod care este în lista de vecini.
   - Se începe de la programul anterior, avem nevoie de un lider ales. Faceți o copie codului anterior.
   - Fiecare proces are o valoare. Pentru toți valoarea este 0 cu excepția liderului, acesta are valoarea 1.
   - Valoarea este transmisă și la fiecare pas se modifică ca fiind media între valoarea primită și cea locală.
   - Se vor folosi exclusiv funcții de comunicare **MPI_Sendrecv()**.
   - Presupunem că nu știm mărimea rețelei, scopul este să o aflăm.