Tema de casă #2 – Sortarea folosind MPI

Responsabil de temă: Ciprian Dobre (ciprian.dobre@cs.pub.ro)

Data publicării: 26.10.2008
Data ultimei modificări a enunțului: 26.10.2008
Termenul de predare: 16.11.2008

Obiective

După realizarea acestei teme de casă studentul va fi capabil să:

- Realizeze o aplicație bazată pe comunicația prin mesaje între procese MPI distribuite.
- Gestioneze eficient structuri de date folosite în rezolvarea unor probleme de sortare.
- Realizeze programe ce folosesc paradigma „bag-of-tasks”.

Cunoştințele necesare rezolvării acestei teme de casă:

- Programare folosind MPI
- Structuri de date: cozi, vectori.
- Algoritmul de sortare quick-sort.

Enunțul problemei

În cadrul acestei teme se propune dezvoltarea unei aplicații de tip „bag-of-tasks” ce are ca scop sortarea unui vector de N (de valoare mare) elemente întregi pozitive utilizând procese MPI.
Soluția de sortare pe care o veți implementa este următoarea. În cadrul sistemului de procese MPI se consideră un proces (cel având rangul 0) având un rol special, de gestionar al task-urilor curente – el ține un „bag-of-tasks”. Restul proceselor (cele având rangul mai mare ca 0) pun task-uri în coadă (trimit mesaje conținând task-uri către procesul cu rang 0) şi pot prelua task-uri din coada deținută de procesul 0 (trimit cereri şi primesc răspunsuri conținând următoarele task-uri de rezolvat). Tot procesul având rangul 0 va gestiona şi vectorul final de valori sortate.
Procesele având rang mai mare ca 0 sunt „workeri”. Ele vor executa următoarele:
Pentru început procesul având rangul 1 citeşte valorile vectorului de sortat şi realizează o primă împărțire în subvectori. Astfel, considerând *k* prima valoare din vector, el va pune elementul *k* pe poziția corespunzătoare *i* din vectorul final de sortat. Deci, după împărțire, procesul va obține un subvector de *i-1* elemente având valori mai mici sau egale cu *k* şi un subvector de *N-i* elemente având valori mai mari sau egale cu *k*. Va trimite cei doi subvectori, împreună cu elementul *k*, procesului 0.
Procesul 0 va realiza următoarele. Va pune elementul *k* pe poziția *i* în vectorul final şi va adăuga cei doi subvectori primiți în pool-ul de task-uri curente. Un alt proces worker va

prelua un task (un subvector) şi va realiza aceeaşi operație (va pune primul element din subvector pe poziția corespunzătoare din vectorul final sortat). Operația va genera din nou două subtask-uri. Algoritmul continuă până când toate valorile ajung să fie inserate în poziția corectă din vectorul final sortat.
În implementarea algoritmului se va considera că unele procese pot începe procesarea unui task şi pot eşua (din diverse motive ele pot înceta să funcționeze, aceasta însemnând că ele nu vor trimite niciodată înapoi rezultate). Se cere implementarea unei soluții tolerante la defecte. Astfel, procesul 0 poate detecta defectarea unui proces şi, în acest caz, poate retrimite task-ul ce era procesat de respectivul proces unui alt proces încă funcțional.

Precizări legate de implementarea temei

Se cere implementarea eficientă a soluției descrise folosind procese MPI. În fişierul Readme veți argumenta decizia făcută pentru tipurile de apeluri MPI de transmitere a datelor pe care le-ați folosit în implementare (blocante, neblocante...). Simularea defectării unui proces se face astfel: procesele vor executa într-o buclă primirea unui task, operația asupra task-ului, trimiterea răspunsului; la un moment dat, în funcție de o decizie aleatoare în program, un proces poate doar recepta un task şi relua bucla de operații, fără realizarea operației şi trimiterea răspunsului. Procesul 0, în acest caz, trebuie să poată depista că procesul nu mai e dispus să trimită înapoi răspuns şi va reasigna task-ul respectiv unui alt proces.