Tema 3 - Structuri Multiprocesor

Bloțiu Mihnea-Andrei - 343C1

Universitatea Politehnica București, Facultatea de Automatică și Calculatoare
mihnea.blotiu@stud.acs.upb.ro

Abstract. În această temă, urmează să analizez Problema Comis - Voiajorului prin compararea variantei de brute-force serială cu paralelizarea acesteia prin diferite tehnologii: Pthreads, OpenMP, MPI, MPI - Pthreads, MPI - OpenMP

Keywords: Problema Comis-Voiajorului · MPI · Serial · Pthreads · OpenMP · Hybrid (MPI-OpenMP / MPI-Pthreads)

1 Introducere

1.1 Descriere sumară a problemei

Problema Comis-Voiajorului este definită astfel: Fie G = (V, E) (unde V este mulțimea nodurilor grafului, iar E mulțimea muchiilor din graf) un graf neorientat complet (în care oricare două vârfuri diferite ale grafului sunt unite printr-o latură) cu costuri strict pozitive pe laturi (costurile pentru sensuri contrare pot fi diferite). Cerinţa este de a determina un ciclu care începe de la un nod aleatoriu al grafului, care trece exact o dată prin toate celelalte noduri şi care se întoarce la nodul iniţial, cu condiţia ca acest ciclu sa aiba un cost minim. Costul unui ciclu este definit ca suma tuturor costurilor ataşate laturilor ciclului.

1.2 Exemple de aplicații practice ale acestei probleme

Prima aplicație practică vine chiar din denumirea problemei. Numele problemei provine din analogia cu un vanzator ambulant care pleacă dintr-un oraş, care trebuie să viziteze un număr de oraşe dat şi care apoi trebuie să se întoarcă la punctul de plecare, cu un efort minim (de exemplu timpul minim, caz în care costul fiecărei laturi este egal cu timpul necesar parcurgerii drumului). Astfel, putem să extindem această aplicație la ocupația curierilor din viața de zi cu zi ce pleacă dintr-un sediu și se întorc la finalul zilei în același loc după livrarea unui număr de colete în locații diferite. Alte exemple de aplicații practice:

1. Generarea traseelor urmate de dispozitivele de producere a circuitelor integrate;
2. Secvențierea unui genom (reconstruirea genomului pornind de la fragmente secvențiate între care există suprapuneri).

2 Analiza soluțiilor alese

2.1 Specificarea soluțiilor alese

Pentru rezolvarea acestei teme au fost selectate următoarele variante de implementare:

1. Serială
2. Multi-threading
   (a) OpenMP
   (b) Pthreads
3. Multi-process
   (a) MPI
4. Hybrid
   (a) MPI-OpenMP
   (b) MPI-Pthreads

2.2 Prezentarea sumară a soluțiilor

De precizat este faptul că toate implementările paralele au plecat de la ideea paralelizării variantei seriale de rezolvare, menționată ulterior. Ideea paralelă a fost aceeași pe toată durata temei însă adaptată specific pentru fiecare tehnologie în parte.

2.2.1 Serială Implementarea serială are la bază citirea numărului de orașe și a matricei de costuri din problemă, generarea tuturor permutărilor posibile pentru numărul citit de orașe și calcularea costului de parcurgere pentru fiecare permutare a orașelor dată pe baza matricei de costuri. Dintre toate aceste variante se reține costul minim pentru care este afișată și permutarea corespunzătoare.

2.2.2 OpenMP și Pthreads Implementările ce au la bază multi-threading își propun îmbunătățirea variantei seriale în felul următor. Se vor diviza permutările care încep cu o anumită cifră către toate thread-urile existente, iar fiecare dintre thread-uri va genera toate permutările posibile pentru mulțimea primită fără prima cifră. Altfel spus, fiecare dintre thread-uri generează permutări pentru (n - 1) orașe, primul fiind mereu fix și stabilit de la început. Dacă numărul de orașe nu se împarte exact la numărul de thread-uri, munca nu va fi împărțită egal. Aceeași situație există și în cazul în care sunt mai multe thread-uri decât orașe, caz în care unele thread-uri nu vor avea nimic de făcut.

2.2.3 MPI Exact aceeași abordate este folosită și în cazul MPI, cu mențiunea că fiind vorba despre o abordare multi-proces, înainte de începerea rezolvării trebuie să ne asigurăm că toate procesele au la dispoziție matricea de costuri, numărul de orașe, etc. Acest lucru presupune existența unei comunicări suplimentare față de variantele de multi-threading.

2.2.4 MPI-OpenMP și MPI-Pthreads Pentru variantele hibride se pleacă de la varianta de implementare de la MPI în care fiecare proces are de generat permutări pentru (n - 1) orașe în care primul oraș este mereu fixat. În cazul în care nu există suficiente procese pentru a face acest lucru, anumite procese vor avea de generat permutări pentru două sau mai multe orașe pe prima poziție. Aceste situații vor fi împărțite pe thread-uri în cazul fiecărui proces.

3 Rezultate obținute

În ceea ce urmează se vor prezenta sumar rezultatele obținute prin câteva grafice comentate.

Graficele de timp reprezintă timpul de execuție înregistrat pentru fiecare dintre implementări în funcție de dimensiunea testului și numărul de thread-uri folosite. Pe același grafic, toate variantele de implementări paralele sunt comparate în cazul folosirii aceluiași număr de thread-uri. În cazul implementărilor hibride, s-au pus pe grafice doar acele situații în care produsul dintre numărul de procese și numărul de thread-uri din fiecare proces este egal cu numărul total de thread-uri/procese din implementările simple. Vor fi prezentate doar o parte din graficele realizate pentru evidențierea concluziei proiectului.

Graficele de speed-up reprezintă ce speed-up s-a obținut pentru testele relevante (cu un număr suficient de mare de orașe) pentru fiecare dintre implementările paralele și pentru implementarea hibridă din fiecare variantă care a obținut speedup-ul mediu maxim. Pentru acest tip de grafice am ales testele 13 și 14 care conțin 13 și respectiv 14 orașe fiind cele mai mari folosite în realizarea proiectului.

3.1 Grafice de timp

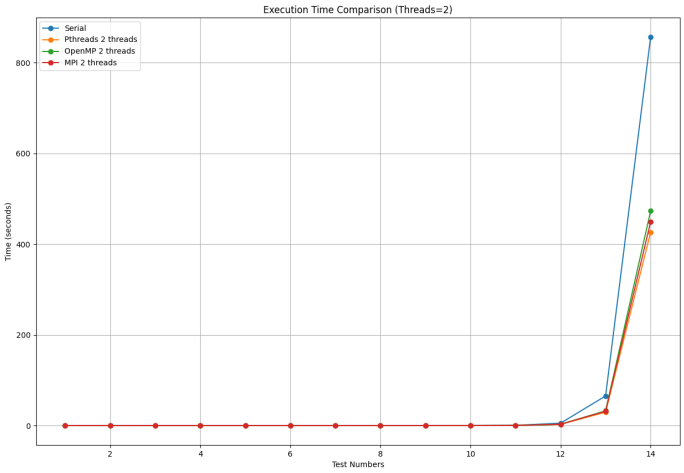

Fig. 1. Timpul de execuție în funcție de dimensiunea testului pentru soluțiile implementate folosind 2 thread-uri în cele paralele

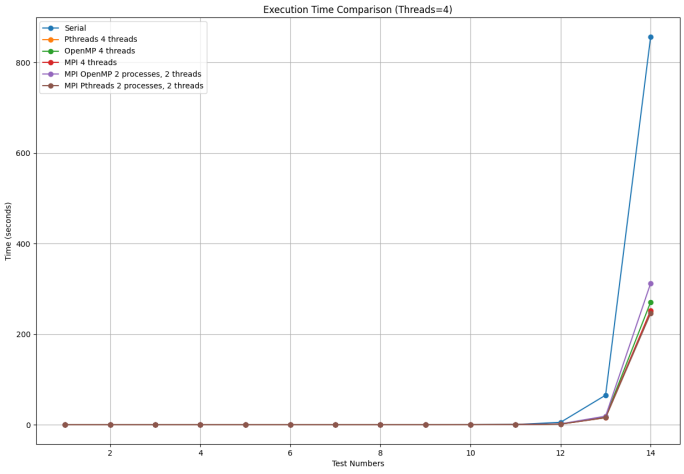

Fig. 2. Timpul de execuție în funcție de dimensiunea testului pentru soluțiile implementate folosind 4 thread-uri în cele paralele

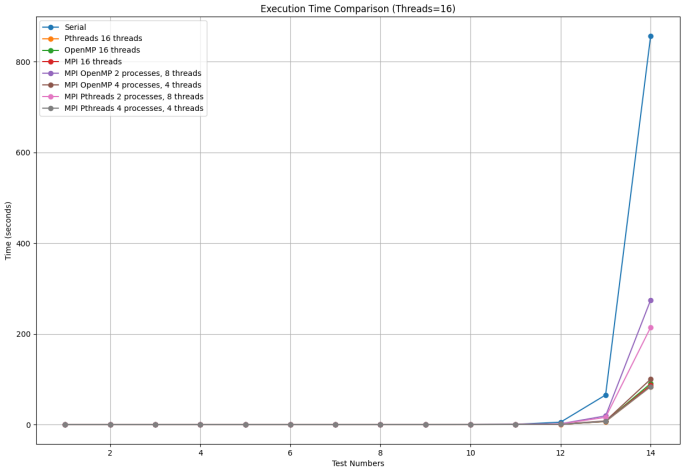

Fig. 3. Timpul de execuție în funcție de dimensiunea testului pentru soluțiile implementate folosind 16 thread-uri în cele paralele

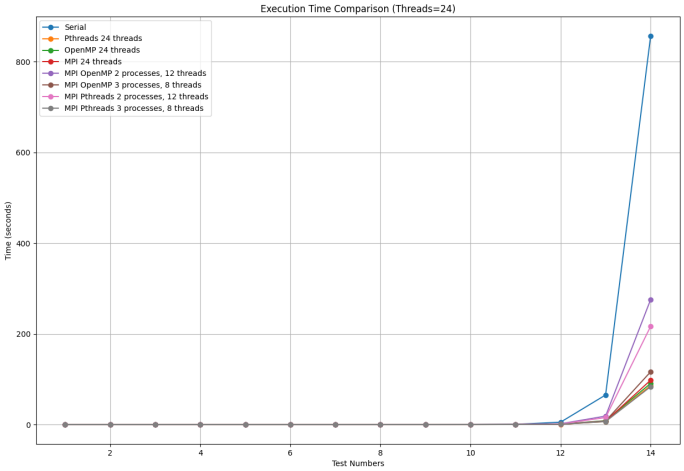

Fig. 4. Timpul de execuție în funcție de dimensiunea testului pentru soluțiile implementate folosind 24 thread-uri în cele paralele

3.2 Grafice de speed-up

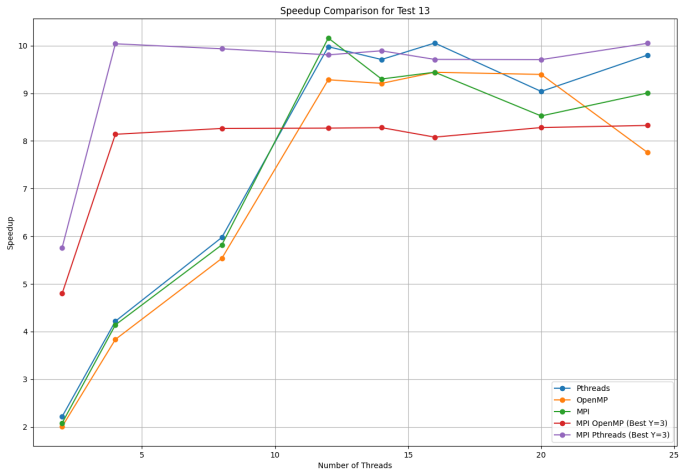

Fig. 5. Speed-up pentru fiecare dintre implementările paralele și pentru implementările hibride cu speed-up mediu maxim

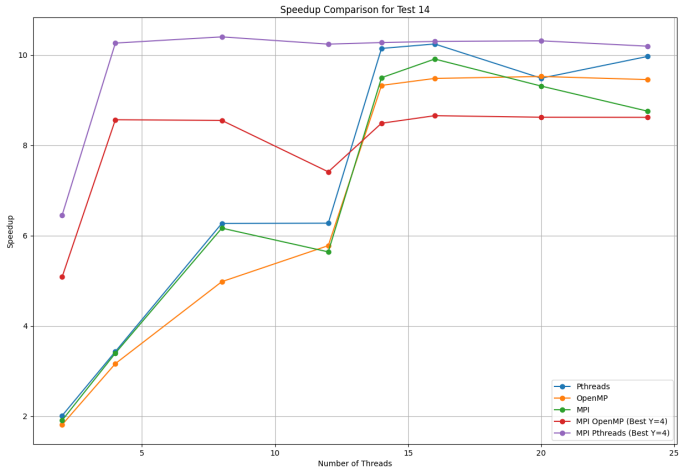

Fig. 6. Speed-up pentru fiecare dintre implementările paralele și pentru implementările hibride cu speed-up mediu maxim

4 Concluzii

1. Din graficele de timp se pot observa:
   (a) Toate implementările paralele sunt mai bune decât cea serială ceea ce trebuia obligatoriu să se întâmple pentru a avea sens paralelizarea
   (b) Crescând numărul de thread-uri, scade timpul de execuție pentru toate implementările paralele ceea ce din nou era de asteptat să se întâmple până la un punct care va fi discutat în concluziile de speed-up.
   (c) Implementările au timpi de execuție foarte similari deoarce practic pun în aplicare aceeași idee de paralelizare, doar tehnologia diferind.
2. Din graficele de speed-up se pot observa:
   (a) Implementările paralele sunt corecte deoarece respectă legea lui Amdahl. Niciodată speedup-ul nu depășește numărul de thread-uri folosite.
   (b) Speedup-ul se aplatizează, chiar începe să scadă odată cu creșterea numărului de thread-uri. Practic, în momentul în care avem mult prea multe thread-uri comparativ cu numărul de orașe, iar o parte din ele nu lucrează, durează mai mult crearea acelor thread-uri comparativ cu beneficiul pe care îl aduc. În anumite cazuri acesta este chiar 0.
   (c) Cea mai bună implementare în medie în funcție de speedup-ul obținut este MPI-Pthreads cu un număr de 3 și respectiv 4 procese (Y din figură) trecând prin toată varietatea de thread-uri, deoarece în medie graficul mov se află mereu deasupra celorlalte.