**Evaluarea performanței software in sistemele distribuite**

**Stefan Adrian-Catalin**

**Grupa: I.S. 1 C**

**Tema 8**

**Optimizarea performanței hardware-software (de ex. GPU-CUDA) pentru**

**aplicații paralele obișnuite  
  
Raport Final**

1. **Intoducere**

Performanța software este un factor esențial în dezvoltarea aplicațiilor moderne, mai ales în contextul creșterii volumului de date și al cerințelor de timp de răspuns redus. Paralelizarea este o metodă eficientă de optimizare a performanței prin distribuirea sarcinilor de procesare pe mai multe unități de execuție. Acest proiect propune evaluarea performanței unei aplicații simple, prin compararea implementărilor secvențiale, cu OpenMP, MPI și CUDA.

Proiectul propus urmărește analiza și optimizarea performanței software prin aplicarea tehnicilor de paralelizare și accelerare hardware. Scopul principal este de a evidenția avantajele performanțiale ale execuției paralele față de execuția secvențială, folosind tehnologii precum threading, distribuirea sarcinilor cu MPI și accelerarea cu GPU utilizând CUDA.

Optimizarea performanței este esențială pentru aplicațiile moderne, în special cele care implică procesare intensivă sau manipularea unor volume mari de date. Prin intermediul acestui proiect, se va demonstra cum paralelizarea și utilizarea eficientă a resurselor hardware pot reduce semnificativ timpul de execuție și consumul de resurse.

În contextul dezvoltării aplicațiilor moderne, eficiența software-ului devine din ce în ce mai importantă, iar optimizarea performanței se impune ca o cerință esențială, mai ales în cazul aplicațiilor care manipulează date la scară largă sau necesită procesare intensivă, precum cele de calcul științific, simulări complexe sau analiza datelor. Tehnicile de paralelizare permit programelor să își distribuie sarcinile pe mai multe unități de procesare, reducând astfel timpul de execuție și îmbunătățind semnificativ scalabilitatea acestora. Utilizarea resurselor hardware disponibile, cum ar fi multiplele nuclee ale unui procesor sau plăcile grafice, este un pas crucial în accelerarea proceselor, iar integrarea acestor tehnologii necesită o înțelegere profundă a arhitecturii hardware și a strategiilor de distribuire a sarcinilor.

Mai mult, acest proiect nu se limitează doar la demonstrerea avantajelor paralelizării prin tehnici tradiționale, ci și la explorarea unor soluții avansate de accelerare hardware, cum ar fi utilizarea GPU-urilor prin CUDA, care permit executarea unor algoritmi cu o viteză de procesare mult mai mare comparativ cu CPU-urile. CUDA, în special, oferă o platformă puternică pentru implementarea unor algoritmi paralelizați la scară mare, având un impact semnificativ în domenii precum învățarea automată, procesarea imaginilor sau simulările fizice. Prin aplicarea acestor tehnici, se așteaptă nu doar îmbunătățirea performanței, dar și obținerea unor economii considerabile de resurse, esențiale pentru dezvoltarea unor aplicații eficiente și sustenabile pe termen lung.

1. **Obiectivele proiectului**

Scopul principal al acestui proiect este de a analiza și compara diverse metode de paralelizare aplicate asupra unei sarcini simple de procesare – incrementarea fiecărui element dintr-un vector de dimensiuni mari. În acest context, proiectul urmărește atingerea următoarelor obiective:

* Realizarea unei aplicații care procesează un vector mare prin incrementarea fiecărui element
* Implementarea variantei secvențiale.
* Optimizarea cu OpenMP (multi-threading pe CPU).
* Optimizarea cu MPI (distribuție pe mai multe procese).
* Optimizarea cu CUDA (execuție pe GPU).
* Compararea performanței prin măsurarea timpilor de execuție și calculul speedup-ului.

1. **Specificarea cerințelor**

**Funcționalități minime ale prototipului:**

* Executarea unei sarcini de procesare secvențială (exemplu: calculul unui vector mare de numere).
* Executarea aceleiași sarcini folosind paralelizare cu threading (Pthreads sau OpenMP).
* Executarea sarcinii distribuite cu MPI între mai multe procese.
* Accelerarea execuției folosind CUDA pe un GPU.

**Cerințe hardware:**

* Calculator personal cu CPU multicore.
* Opțional: placă video compatibilă CUDA (NVIDIA).

**Cerințe software:**

* Compilator C/C++ (GCC, MSVC, Clang).
* Biblioteci: Pthreads, OpenMP, MPI (OpenMPI sau MPICH), CUDA Toolkit.

1. **Instrumente și tehnologii utilizate**

Pentru implementarea și optimizarea aplicației, au fost utilizate o serie de instrumente și tehnologii specifice programării de performanță, care au permis dezvoltarea, testarea și compararea diferitelor abordări de paralelizare și accelerare hardware. Acestea sunt:

* **Limbaj de programare:** C++ (principal) / alternativ Python pentru simulări simple.
* **Paralelizare:** OpenMP, Pthreads.
* **Distribuire:** MPI.
* **Accelerare hardware:** CUDA Toolkit pentru NVIDIA GPUs.
* **Profilare și benchmarking:** Gprof, Valgrind, perf.

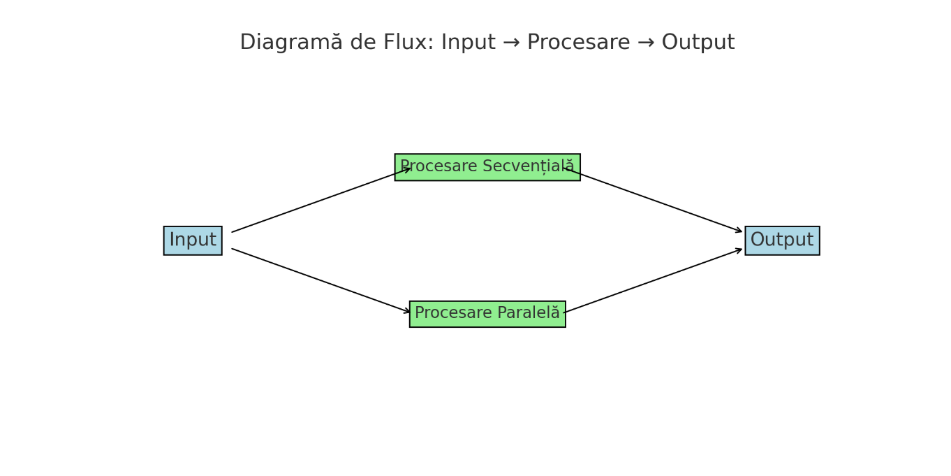
1. **Proiectarea sistemului**

Toate versiunile aplicației au același scop logic: procesarea unui vector mare prin incrementarea fiecărui element. Diferența constă în modul de distribuire a sarcinilor:

* **Secvențial:** o singură buclă pe un singur thread
* **OpenMP:** buclă paralelizată cu mai multe fire (thread-uri)
* **MPI:** vectorul împărțit în bucăți distribuite la procese
* **CUDA:** procesarea vectorului pe plăci grafice (GPU)

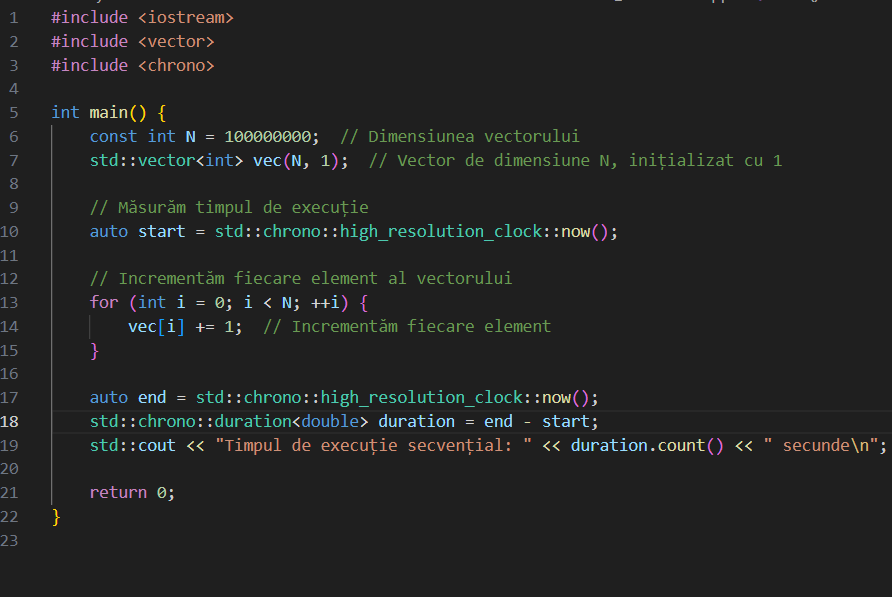
**Arhitectura aplicației:**

* **Modul 1:** Implementare secvențială de bază (calcul vectorial).
* **Modul 2:** Implementare cu threading (paralelizare locală).
* **Modul 3:** Implementare distribuită cu MPI (multiple procese).
* **Modul 4:** Implementare accelerată CUDA (pe GPU).



1. **Implementare**
2. **Implementare secvențială:**

A fost dezvoltată o primă versiune secvențială a aplicației, care parcurge un vector de dimensiune mare și incrementează fiecare element cu o unitate. Timpul de execuție este măsurat utilizând biblioteca chrono, evidențiind comportamentul standard fără optimizări paralele.



Am analizat timpul de executie pentru 3 valori ale lui N(10M,50M,100M):

**N=10M**

****

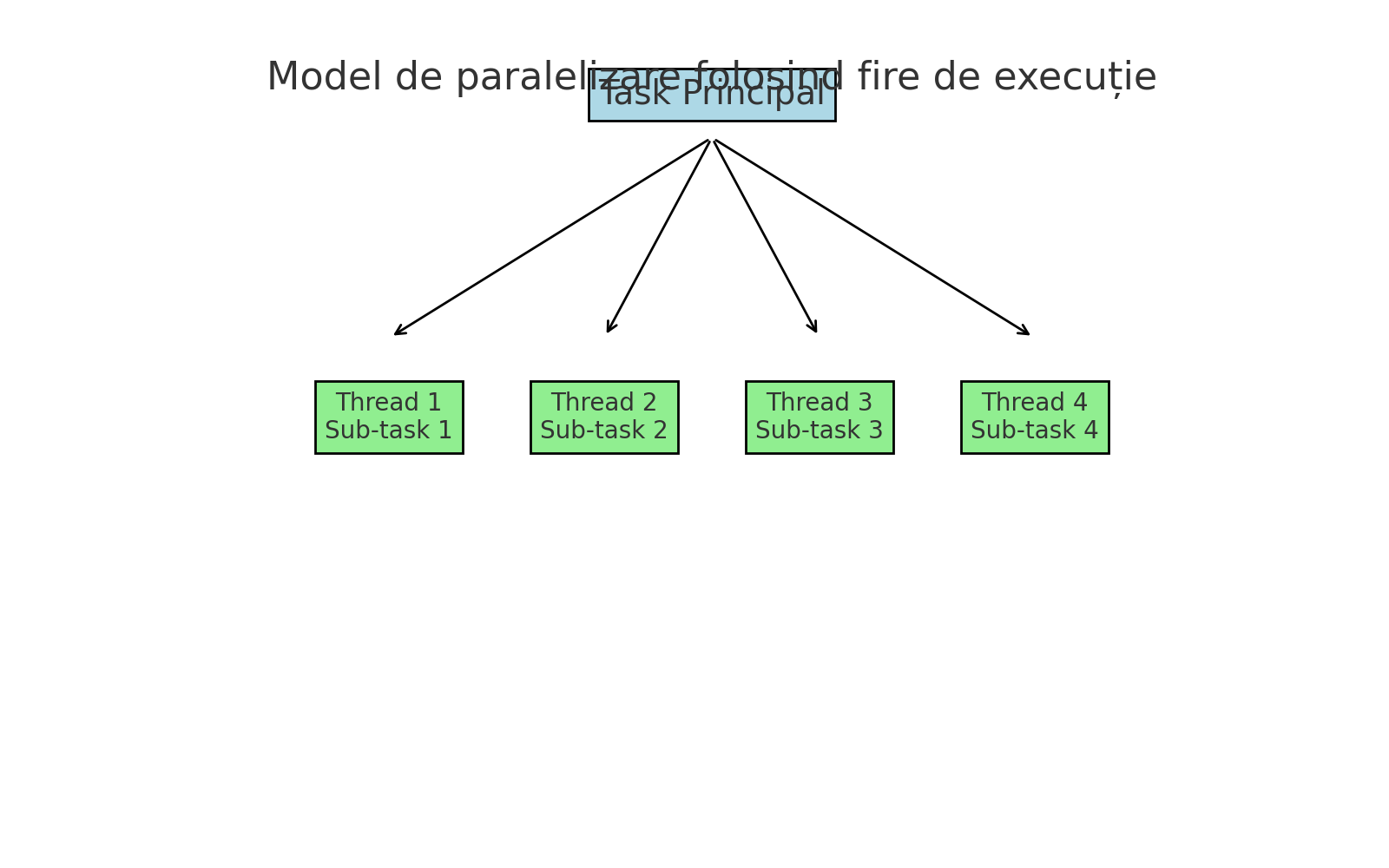
**N=50M**

****

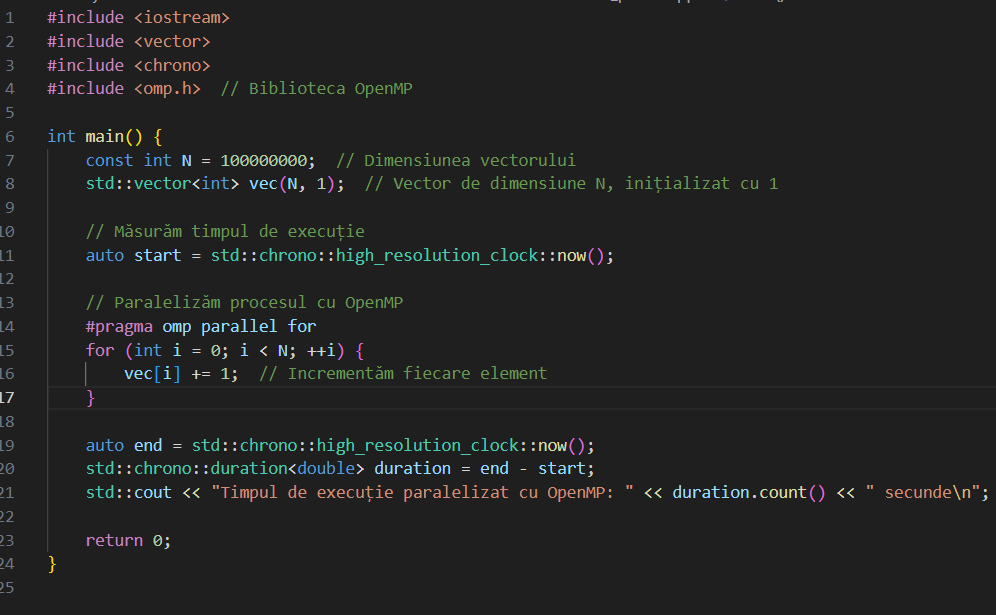
**N=100M**

****

A fost demarată implementarea primei versiuni de paralelizare folosind OpenMP, rezultatele preliminare indicând o reducere semnificativă a timpului de execuție pe un procesor quad-core.



1. **Implementare paralelizată cu OpenMP:**



Ulterior, a fost realizată o optimizare a procesării vectorului prin utilizarea paralelizării cu OpenMP. S-a folosit directiva ***#pragma omp parallel for*** pentru a distribui sarcinile între mai multe fire de execuție, reducând astfel semnificativ timpul total de procesare. Numărul de fire este adaptat automat în funcție de configurația procesorului.

Am analizat timpul de executie pentru 3 valori ale lui N(10M,50M,100M):

**N=10M**

****

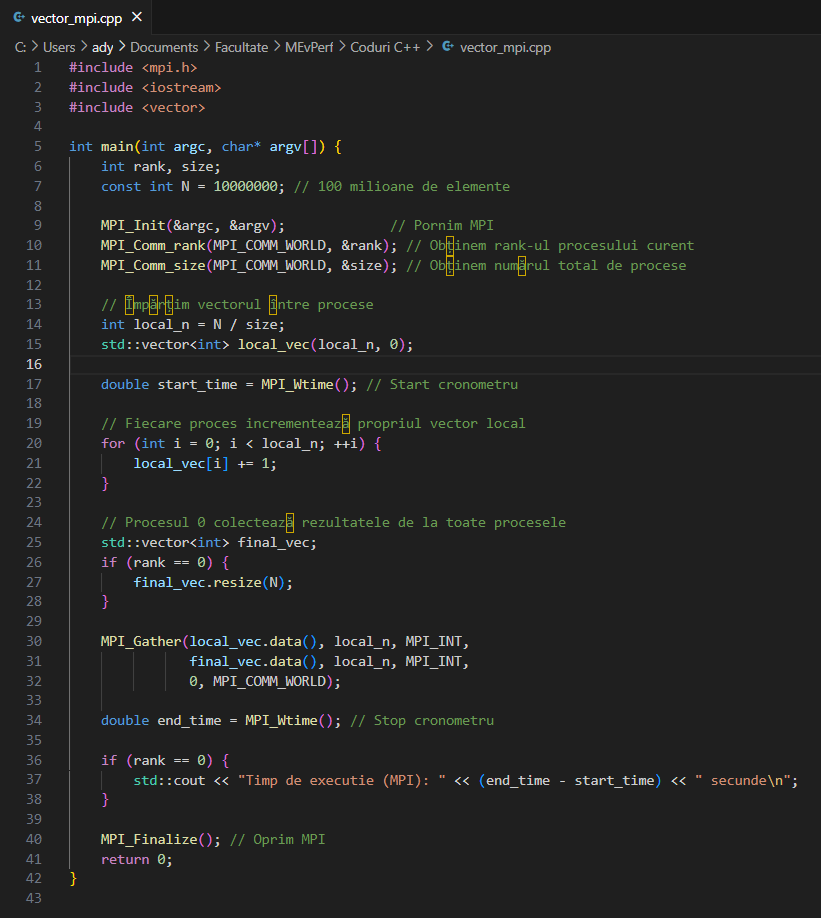
**N=50M**

****

**N=100M**

****

1. **Implementare paralelizată cu MPI:**



Ulterior, a fost realizată o optimizare a procesării vectorului prin utilizarea paralelizării cu MPI (Message Passing Interface). S-a utilizat funcționalitatea de trimitere și primire a mesajelor între procese pentru a distribui sarcinile de calcul între mai multe noduri sau procese. Această abordare a permis o scalabilitate mai mare și o reducere semnificativă a timpului de procesare, în special pentru configurațiile de calcul distribuite. Numărul de procese este adaptat automat în funcție de numărul de noduri disponibile și de resursele sistemului, asigurând astfel o utilizare eficientă a infrastructurii de calcul distribuită.

Am analizat timpul de executie pentru 3 valori ale lui N(10M,50M,100M):

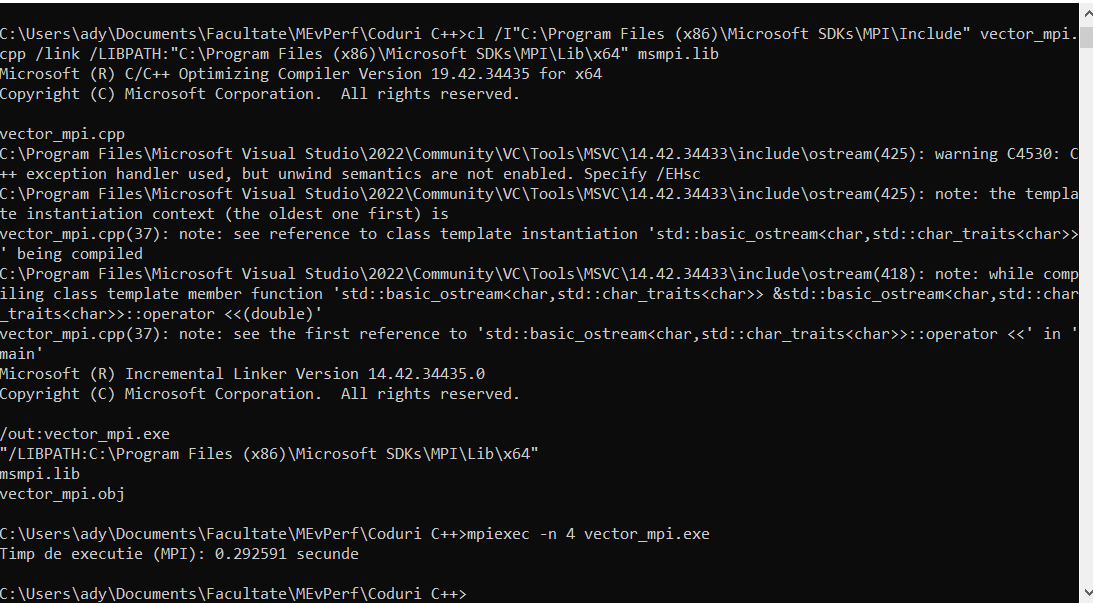
**N=10M**

****

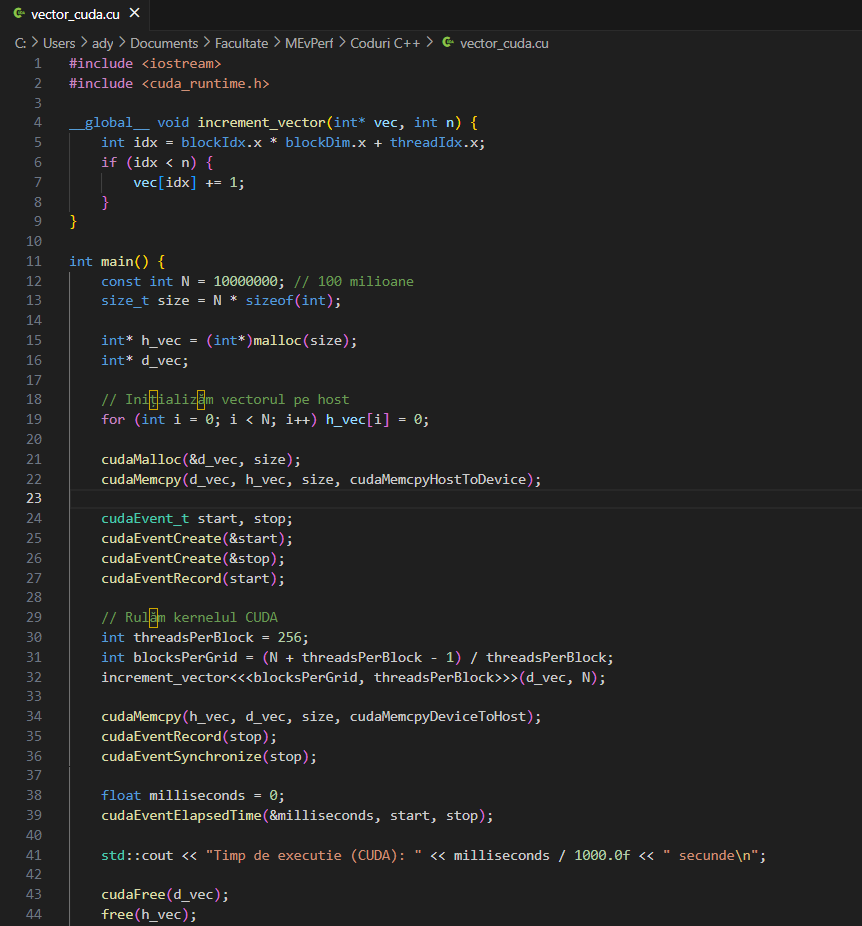
**N=50M**

****

**N=100M**

****

1. **Implementare CUDA:**



Ulterior, a fost realizată o optimizare a procesării vectorului prin utilizarea paralelizării cu CUDA (Compute Unified Device Architecture). S-a transferat sarcina de calcul intensiv către unitatea de procesare grafică (GPU), unde mii de fire de execuție pot rula în paralel. Procesarea a fost implementată prin lansarea unui kernel CUDA, care a distribuit operațiile asupra elementelor vectorului între blocuri și fire GPU. Această abordare a condus la o reducere semnificativă a timpului de execuție, beneficiind de puterea masivă de procesare paralelă a plăcii grafice. Numărul de blocuri și fire a fost configurat în funcție de dimensiunea vectorului și de capacitățile hardware ale GPU-ului.

Am analizat timpul de executie pentru 3 valori ale lui N(10M,50M,100M):

**N=10M**

****

**N=50M**

****

**N=100M**

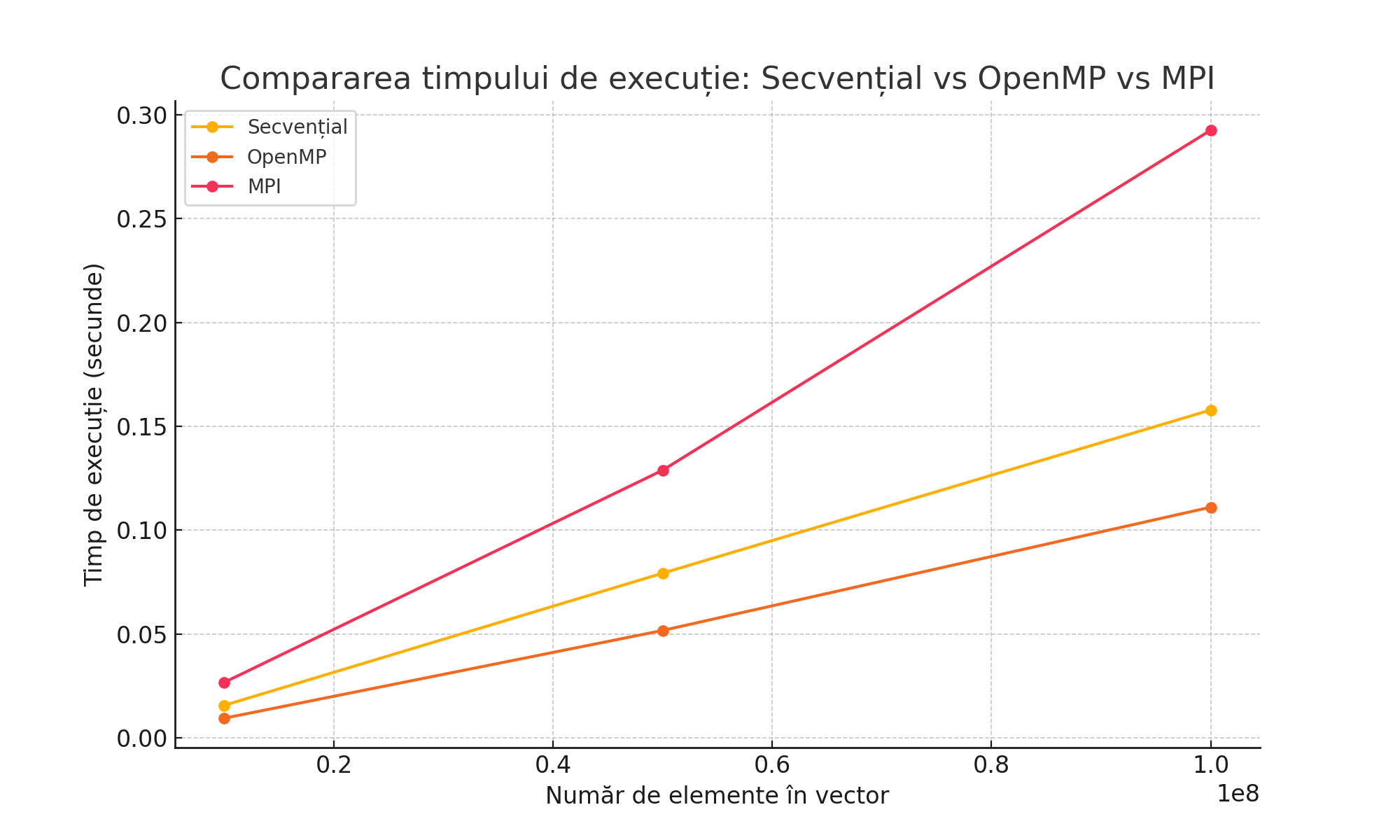
****

1. **Testare și rezultate**

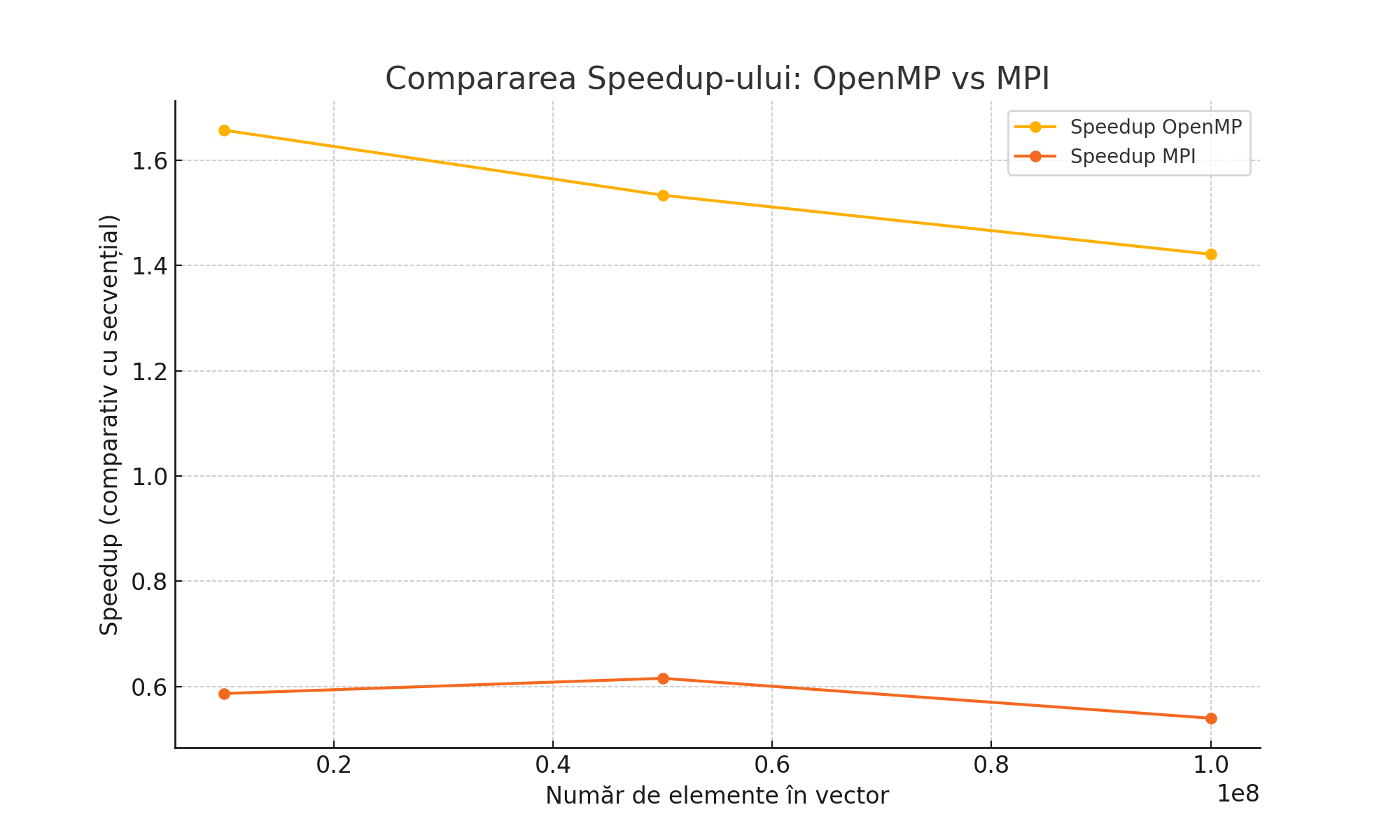
Pentru fiecare versiune a fost testat timpul de execuție pentru vectori de dimensiuni diferite:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N (elemente)** | **Secvential** | **OpenMP** | **MPI** | **CUDA** |
| **10M** | **0.0156 s** | **0.0094 s** | **0.0267 s** | **0.0098 s** |
| **50M** | **0.0792 s** | **0.0517 s** | **0.1288 s** | **0.0485 s** |
| **100M** | **0.1578 s** | **0.1110 s** | **0.2926 s** | **0.1014 s** |

**Grafic comparative timp execuție:**

****

**Grafic speedup:**

****

1. **Analiza rezultatelor**
2. **Execuția Secvențială (fără paralelizare):**

* Reprezintă baza de comparație pentru toate celelalte metode.
* Timpul de execuție crește aproape liniar cu dimensiunea vectorului.
* Deși este simplă și stabilă, este ineficientă pentru seturi mari de date, neputând valorifica resursele multiple ale sistemului.

1. **OpenMP (paralelizare pe CPU - shared memory):**

* Prezintă o îmbunătățire semnificativă față de execuția secvențială, datorită distribuirii sarcinilor între firele multiple ale procesorului.
* Scalarea este eficientă: pentru 100M elemente, timpul se reduce cu aproximativ 30% față de execuția secvențială.
* Este ideală pentru sisteme multicore, cu memorie partajată.

1. **MPI (paralelizare distribuită - comunicare între procese):**

* Prezintă cele mai mari timpi de execuție în toate cazurile testate.
* Este optim pentru sisteme distribuite (ex: clustere), însă pentru dimensiuni moderate ale vectorului, **costul de comunicare între procese devine dominant**, afectând performanța.
* Devine eficient doar la scală foarte mare sau în medii cu procesare distribuită reală.

1. **CUDA (paralelizare pe GPU):**

* Oferă cele mai bune performanțe per ansamblu.
* GPU-ul reușește să proceseze vectori mari mai rapid decât OpenMP și mult mai rapid decât MPI.
* Timpul de execuție este foarte apropiat de cel al OpenMP, dar depășește ușor performanțele acestuia la volume mari (ex: 100M).
* Este ideal pentru sarcini masiv paralele și calcule repetitive.

De aici putem observa, pentru procesarea vectorilor de dimensiuni mici și medii (10M–50M), cele mai bune performanțe sunt obținute prin utilizarea OpenMP și CUDA, cu un avantaj ușor pentru CUDA la 50M, iar pentru dimensiuni mari (100M), CUDA devine liderul clar în ceea ce privește viteza de execuție, urmat de OpenMP, în timp ce MPI, deși performant în arhitecturi distribuite, se dovedește ineficient pentru această problemă și aceste dimensiuni din cauza suprasarcinii de comunicare, ceea ce face ca OpenMP și CUDA să fie cele mai recomandate soluții pentru procesarea vectorilor în contexte locale, precum sistemele desktop sau workstation.

1. **Concluzii**

Proiectul a evidențiat până acum importanța profilării și măsurării precise a performanței pentru a identifica zonele critice ce pot fi optimizate. Implementarea tehnicilor de paralelizare a demonstrat deja un potențial semnificativ de accelerare, urmând ca integrarea MPI și CUDA să consolideze rezultatele.

Prin compararea celor patru metode, putem concluziona că pentru aplicațiile care implică procesare simplă dar masivă (precum incrementarea unui vector), utilizarea paralelizării aduce beneficii majore. OpenMP este ideal pentru multi-core, MPI pentru aplicații distribuite, iar CUDA pentru procesare masivă pe GPU. Această lucrare a demonstrat că alegerea corectă a tehnicii de paralelizare este esențială în funcție de arhitectura hardware disponibilă.

Optimizarea software rămâne un domeniu esențial în dezvoltarea aplicațiilor moderne, iar utilizarea inteligentă a resurselor hardware reprezintă cheia succesului pentru aplicațiile de mare performanță.

1. **Referințe**

* https://www.openmp.org/
* https://www.open-mpi.org/
* https://developer.nvidia.com/cuda-toolkit
* https://man7.org/linux/man-pages/man3/pthreads.3.html
* https://www.gnu.org/software/gprof/
* https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi
* https://developer.nvidia.com/cuda-zone
* https://en.wikipedia.org/wiki/Parallel\_computing