Benchmark al proceselor in diferite limbaje de programare

Student: Ivan Alexandru – Mihai

Structura Sistemelor de Calcul

Universitatea Tehnica Cluj Napoca

1. Introducere

1.1 Context

Scopul acestui proiect este de a masura si compara timpul de executie si operatia unor procese in trei limbaje de programare diferite: C, C++ si Java. Procesele masurate vor fi alocarea de memorie, accesul la memorie (static si dinamic), crearea unui thread, thread context switch si thread migration.

Compararea acestor procese este utila pentru a intelege cum functioneaza fiecare limbaj de programare diferite situatii, ce avantaje si ce dezavantaje exista. Rezultatele obtinute vor fi utile pentru programatorii software pentru a putea alege ce limbaj de programare sa aleaga pentru dezvoltarea cat mai eficienta a programului lor.

1.2 Obiective

Obiectivul principal al proiectului este de a masura timpul de executie al diferitor procese. Acesta va dispune de o interfata grafica, prin care utilizatorul va putea selecta limbajul de programare si procesul dorit, la care se va afisa timpul de executie, timp masurat in nanosecunde.

Procesele care vor fi masurate:

- Alocarea de memorie: Modul in care programul aloca cantitatea de memorie pentru a stoca date, aceasta poate fi statica sau dinamica

- Accesul la memorie: Proces prin care programul citeste sau scrie date in memoria alocata. Accesul static este pentru variabile fixe, iar accesul dinamic este pentru date alocate la runtime

- Crearea unui thread: Crearea unui fir de executie care ruleaza in paralel cu mai multe fire de executie din program.

- Thread context switch: Modul in care sistemul de operare schimba executia de la un thread la altul

- Thread migration: Operatie prin care un thread de pe un nucleu de procesor se muta pe altul, acest proces poate afecta performanta din cauza sincronizarii memoriei intre nuclee

2. Cercetare Bibliografica

2.1 Benchmarking

Benchmarking-ul este procesul de masurare al produselor, serviciilor si proceselor. Acesta ofera informatiile necesare pentru a intelege cum se compara doua sau mai multe produse sau servicii pe care se face masurarea, scopul principal fiind scoaterea in evidenta a celui mai bun serviciu pentru nevoile utilizatorului.

2.2 Microbenchmarking

Microbenchmarking-ul masoara si evalueaza performanta unei bucati mici de cod, spre deosebire de benchmarking, care evalueaza intreaga performanta a mediului de executie. Acesta este important sa fie folosit locul potrivit, altfel poate fi o pierdere de timp. Este esential sa se confirme utilitatea fiecarui microbenchmark inainte de a fi adaugat intr-un proiect.

2.3 Alocarea de memorie

In limbajul de programare Java memoria nu trebuie sa fie alocata dinamic, aceasta este gestionata automat prin garbage collector. C si C++ nu dispun de garbage collector, ceea ce inseamna ca memoria trebuie sa fie alocata dinamic cu functii. Daca memoria nu este dezalocata direct se poate ajunge la leakuri de memorie.

2.4 Crearea Thread-urilor

Thread-ul este un flux secvential de control in cadrul unui program. Acestea pot fi rulate in acelasi timp pentru a face diferite sarcini intr-un program. In C si C++ threadurile sunt gestionate prin biblioteci precum POSIX, in timp ce Java ofera o implementare nativa pentru gestionarea thread-urilor prin API-ul sau.

2.5 Thread context switch

Thread context switch reprezinta cedarea controlului de executie de la un thread la altul. In C, C++ aceasta schimbare este gestionata de sistemul de operare, iar in Java de masina virtuala.

2.6 Thread migration

Thread migration reprezinta procesul prin care un thread care este executat pe un nucleu al procesorului este mutat pentru executie pe alt nucleu. In general, thread migration este gestionat automat de sistemul de operare, dar exista metode de a influenta aceasta migrare prin setarea afinitatii threadurilor.

3. Analiza

Proiectul se axeaza pe masurarea si compararea performantei unor procese in trei limbaje de programare: C, C++ si Java. Fiecare dintre aceste limbaje are caracteristici unice care influenteaza modul in care gestioneaza alocarea si accesul la memorie, crearea si migrarea thread-urilor, precum si operatiunile de context switching intre thread-uri. Analiza acestor aspecte in diverse medii poate oferi o perspectiva asupra eficientei fiecarui limbaj in situatii diferite si asupra modului in care fiecare gestioneaza resursele hardware.

3.1 Limbaje de programare utilizate

3.1.1 C

Limbajul C este cunoscut pentru accesul direct si eficient la resursele hardware, datorita faptului ca utilizeaza un management manual al memoriei. Aceasta caracteristica permite programatorilor sa aloce si sa dezaloce memoria manual, prin intermediul functiilor malloc, calloc si free. Totodata, limbajul C permite crearea si controlul thread-urilor prin biblioteca POSIX, dar gestioneaza toate operatiunile de threading la nivel de sistem de operare.

Librarie pentru masurare timpului: <time.h>

Functii utilizate: Folosind functia clock_gettime() cu ceasul CLOCK_MONOTONIC timpul este masurat in secunde si nano, apoi este convertit in nanosecunde pe baza formulei de mai jos.

timp in nanosecunde = timp masurat in secunde * 10^9 + timp masurat in nanosecunde

3.1.2 C++

Limbajul C++ extinde functionalitatea limbajului C, oferind suport pentru programarea orientata pe obiecte si functii suplimentare de alocare dinamica a memoriei prin operatorul new si dezalocare cu delete. In ceea ce priveste managementul thread-urilor, C++ beneficiaza de biblioteca standard C++11 pentru threading, care adauga un nivel de abstractizare fata de POSIX, dar in esenta este influentat de sistemul de operare in acelasi mod ca in C.

Librarie pentru masurare timpului: <chrono>

Functii utilizate: std::chrono::steady_clock::now() / std::chrono::duration_cast<std::chrono::nanoseconds>, timpul este masurat apoi convertit in nanosecunde.

3.1.3 Java

Limbajul Java dispune de o gestionare automata a memoriei, datorita sistemului de garbage collection, eliminand necesitatea de a dezaloca explicit memoria. Acest lucru simplifica managementul resurselor, dar introduce o latenta specifica operatiilor de colectare a memoriei neutilizate. Java include suport nativ pentru threading prin intermediul claselor din pachetul java.lang, iar operatiunile de threading sunt gestionate de masina virtuala Java (JVM), care aduce o serie de optimizari si mecanisme de sincronizare proprii.

Librarie pentru masurare timpului: java.lang.System

Functii utilizate: System.nanoTime() timpul este masurat in nanosecunde.

3.2 Testarea proceselor

3.2.1 Alocare de memorie

Alocarea de memorie reprezinta procesul prin care un program rezerva o anumita cantitate de memorie pentru a stoca date. Exista doua tipuri principale de alocare:

- alocare statica, efectuata la momentul compilarii
- alocare dinamica, efectuata la momentul rularii.

Algoritmul 1: Masurarea Timpului pentru Alocarea de Memorie Statica

1. Porneste Timerul
2. Declara un array de numere intregi cu valori predefinite
3. Opreste Timerul
4. Calculeaza timpul necesar pentru alocarea memoriei statice

Implementare

In C/C++: Se foloseste time.h si functia clock() pentru a masura timpul, iar valoarea este transformata in nanosecunde pentru comparatie.

In Java: Metoda System.nanoTime() returneaza valoarea curenta a celui mai precis timer disponibil in sistem, in nanosecunde.

Algoritmul 2: Masurarea Timpului pentru Alocarea de Memorie Dinamica

1. Porneste Timerul
2. Aloca dinamic un array de o dimensiune predefinita
3. Opreste Timerul
4. Calculeaza timpul necesar pentru alocarea memoriei dinamice

Implementare

- **In C**: Se folosesc functiile malloc() sau calloc() pentru alocare si free() pentru dezalocare.
- **In C++**: Se utilizeaza operatorul new pentru alocare dinamica si delete pentru dezalocare.
- **In Java**: Memoria este alocata automat prin colectorul de memorie (garbage collector).

3.2.2 Accesul la memorie

Accesul la memorie se face prin citirea sau scrierea datelor in memoria alocata. Accesul static are loc pentru variabilele fixe, iar accesul dinamic este pentru datele alocate in timpul executiei.

Algoritmul 3: Masurarea Timpului pentru Accesul la Memorie

1. Porneste Timerul
2. Se acceseaza elemente ale unui array
3. Opreste Timerul
4. Calculeaza timpul necesar pentru accesul la memorie

Implementare

In C/C++: Accesul static se face pe variabile fixe sau arrayuri initializate, iar accesul dinamic pe variabile sau arrayuri alocate cu malloc() sau new.

In Java: Se folosesc variabile sau array-uri definite in timpul executiei.

3.2.3 Crearea unui Thread

Crearea unui thread permite executia paralela a mai multor taskuri intr-un program. Masurarea timpului pentru crearea unui thread este importanta pentru a evalua performanta programului in scenarii concurente.

Algoritmul 4: Masurarea Timpului pentru Crearea unui Thread

1. Porneste Timerul
2. Creeaza un thread
3. Opreste Timerul
4. Calculeaza timpul necesar pentru crearea unui thread

Implementare

In C: Se utilizeaza biblioteca POSIX (pthread_create()).

In C++: Se foloseste clasa std::thread pentru a crea thread-uri.

In Java: Se foloseste clasa Thread sau interfata Runnable

3.2.4 Thread Context Switch

Thread context switch reprezinta trecerea controlului de executie de la un thread la altul, proces ce poate avea un impact semnificativ asupra performantei in aplicatiile concurente.

Algoritmul 5: Masurarea Timpului pentru Schimbarea Contextului unui Thread

1. Creeaza si initializeaza n thread-uri
2. Porneste Timerul
3. Pentru fiecare pereche de thread-uri care trebuie sa faca switch, schimba contextul de la un thread la altul
4. Opreste Timerul
5. Calculeaza timpul necesar pentru schimbarea contextului

Implementare

- **In C/C++**: Se folosesc resurse partajate, cum ar fi un mutex (pthread_mutex_t sau std::mutex) sau o variabila de conditie (pthread_cond_t sau std::condition_variable).
- **In Java**: Se folosesc primitive de sincronizare (ex. Locks, Semaphores)

3.2.5 Thread Migration

Thread migration reprezinta mutarea unui thread de pe un nucleu al procesorului pe altul. Acest proces poate afecta performanta din cauza sincronizarii datelor intre nuclee.

Algoritmul 6: Masurarea Timpului pentru Migrarea unui Thread

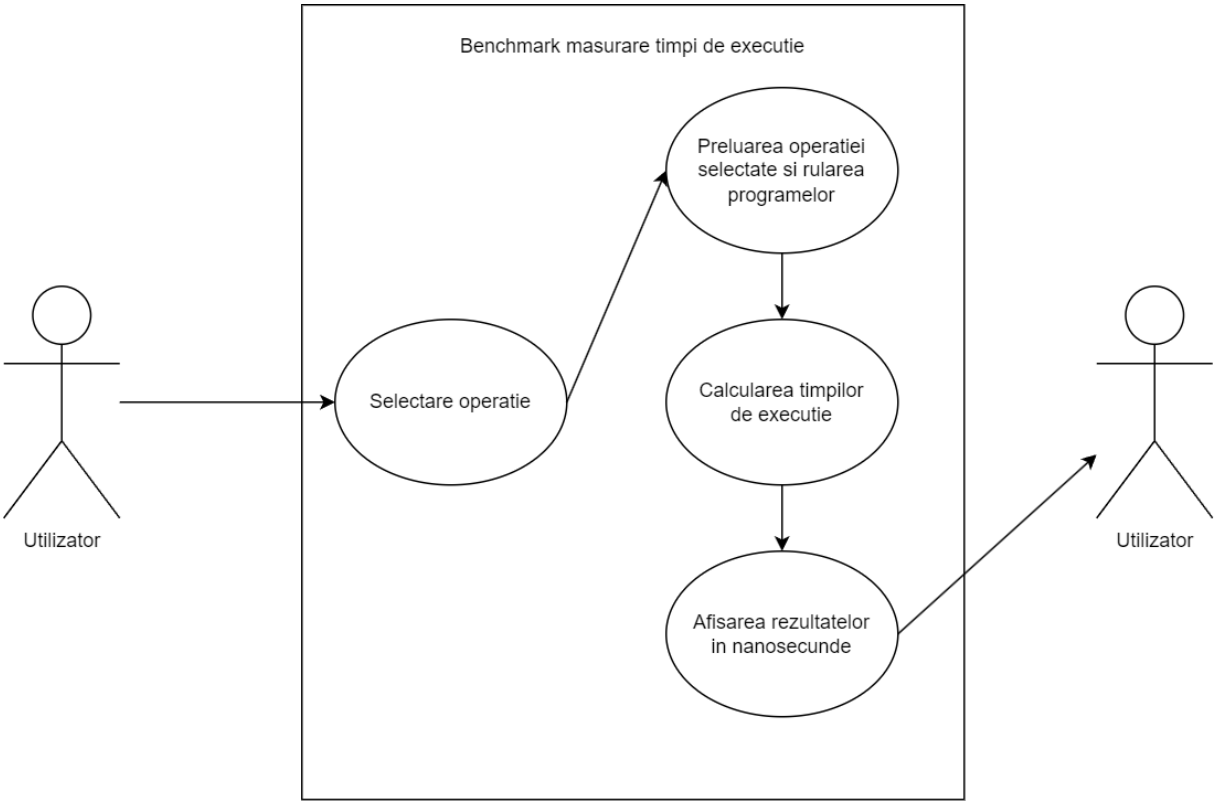
1. Porneste Timerul
2. Asigneaza thread-ul altui nucleu
3. Muta thread-ul pe alt nucleu
4. Opreste Timerul
5. Calculeaza timpul necesar pentru migrarea thread-ului

Implementare

- **In C/C++**: Se poate seta afinitatea procesorului folosind biblioteci specifice sistemului de operare (Windows API pentru Windows, folosind functia SetThreadAffinityMask).
- **In Java**: Migrarea se face automat de catre sistemul de operare, dar afinitatea poate fi influentata prin API-uri externe.

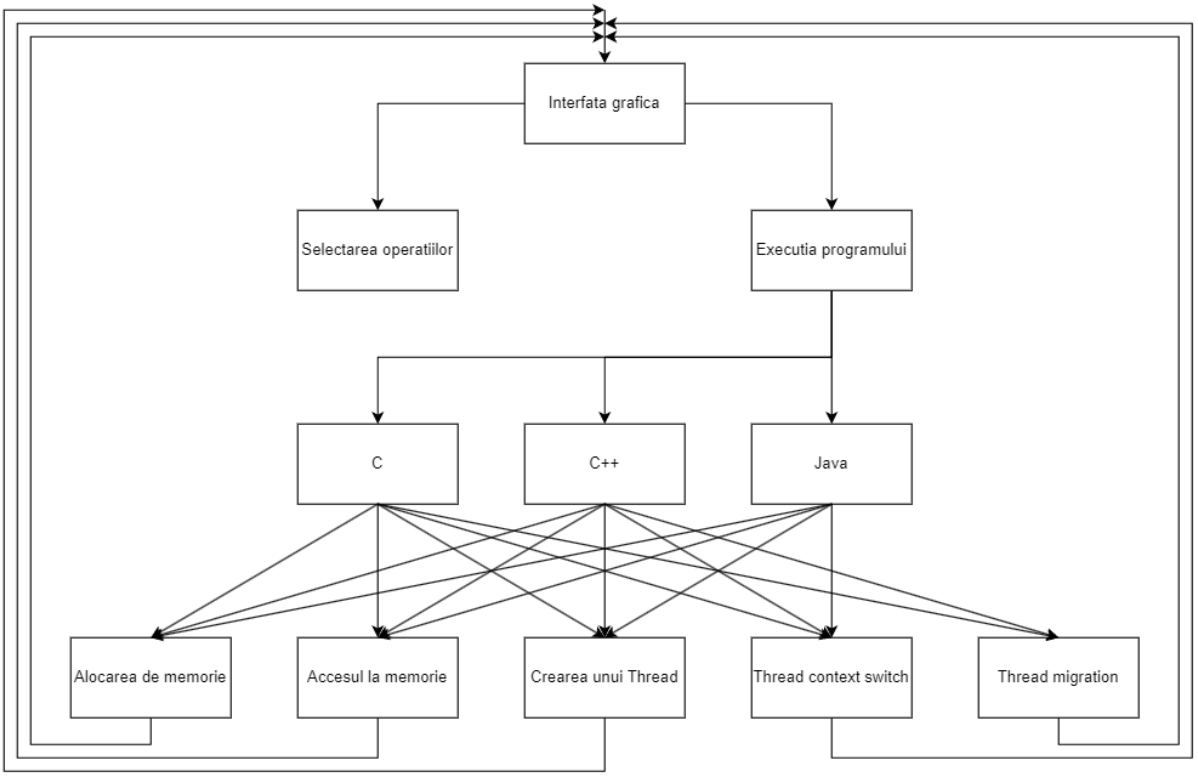
3.3 Diagrama Use-Case

Aceasta diagrama Use-Case prezinta interactiunea dintre un utilizator si aplicatia de masurare a timpilor de executie a mai multor programe. Utilizatorul selecteaza o operatie, aplicatia o preia, ruleaza operatia pe programele integrate, masoara timpul de executie in nanosecunde, apoi transmite rezultatele utilizatorului.



4. Design

Interfata proiectului este o structura ierarhica pe doua nivele, top module, unde utilizatorul va putea selecta o operatie care va fi executata, iar partea de lower module, unde se afla programele care vor rula operatia selectata de utilizator, oferindu-I dupa executie timpul necesar rularii operatie. Timpii ii vor fi afisati in nanosecunde utilizatorului



5. Implementare

- **Alocare de memorie statica**: Se creeaza un sir de numere intregi de o lungime prestabilita (int array[size])

    - int array[array_size] in **C** si **C++**
    - int[] array = new array[size] in **Java**

- **Alocare de memorie dinamica**: Se aloca un sir de numere intregi de o lungime prestabilita folosind functia

    - int* array = malloc() (int* array = (int*)malloc(size*sizeof(int))) in **C**
    - int* array = new int[size] in **C++**
    - Integer array = new Integer[size] in **Java**

- **Accesare de memorie**: Se creeaza un sir de o lungime prestabilita, apoi o variabila ia valoarea fiecarui element din sir

    - x = array[i] in **C**, **C++** si **Java**

- **Crearea unui thread**: Se creeaza un thread si se masoara in cat timp a fost creat

    - pthread_create(&thread, NULL, thread_function, NULL) in **C**
    - thread t(thread_function) in **C++**
    - Thread thread = new Thread(() -> {}); thread.start() in **Java**

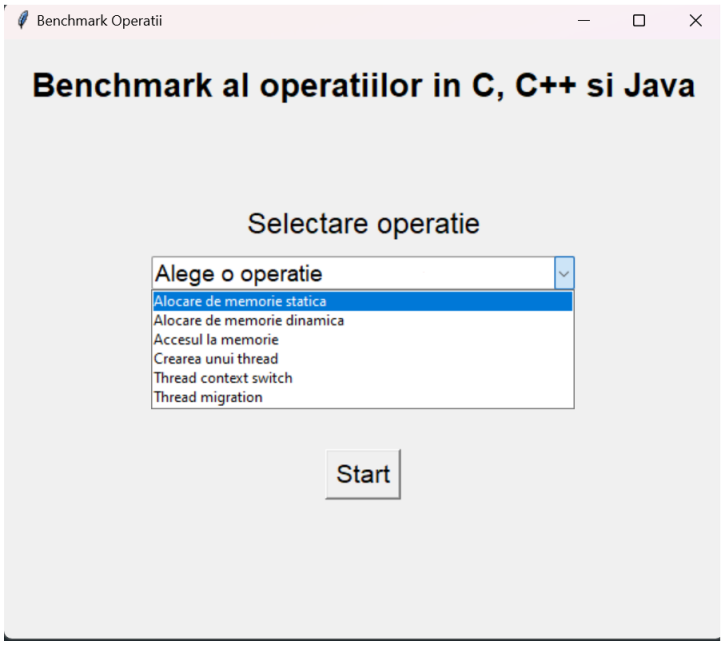
- **Context switch**: Se creeaza mai multe threaduri, care nu isi incep executia, numai in momentul in care s-au creat toate threadurile. Dup ace s-au creat threadurile incep sa blocheze si deblocheze un lacat astfel masurandu-se context switch-ul intre threaduri. Acest proces este realizat de mai multe ori (de 100 de ori de exemplu) pentru mai multe threaduri

- **Thread migration**: Se creeaza un thread si se seteaza pe ce CPU sa ruleze. Se masoara timpul de executie al unui test, insa se ia in considerare doar daca thread-ul s-a rulat pe CPU-ul setat initial, altfel testul se repeta.

    - DWORT_PTR affinityMask = 1 << 1; SetThreadAffinityMask(thread, affinityMask) in **C** si **C++**

- **Interfata**

    - Interfata este una intuitiva si simplu de folosit, utilizatorul selecteaza o operatie din cele 6, apasa pe butonul de start, iar apoi se v-a genera un grafic care indica timpul de executie al operatiei in C, C++ si Java. In cazul in care utilizatorul nu alege nici o operatie se va afisa un mesaj de eroare

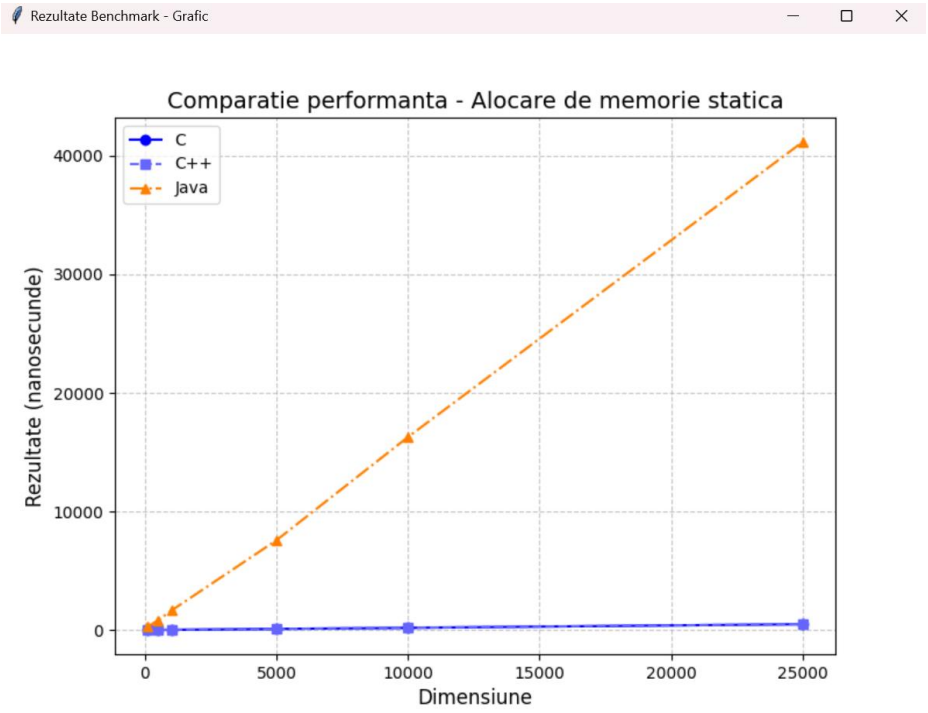


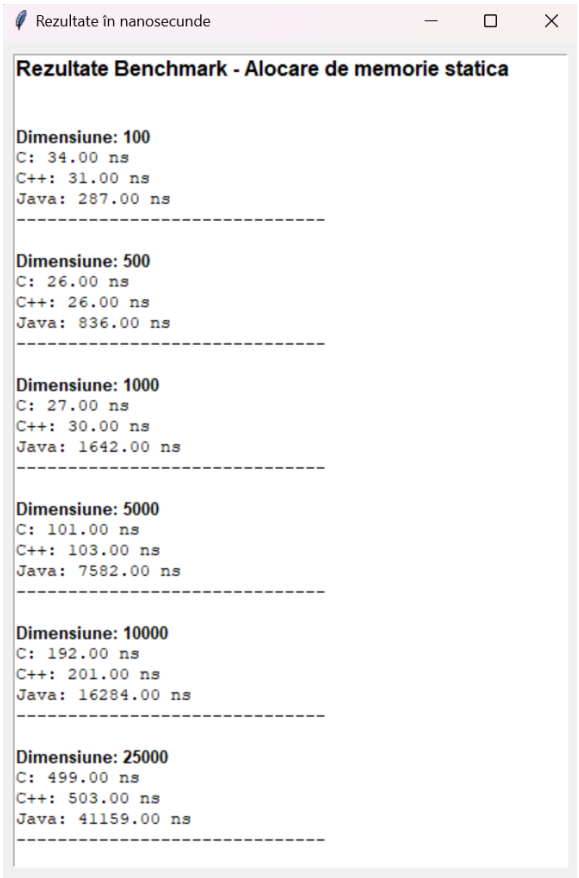
6. Grafice generate

Graficele de mai jos ilustreaza performanta fiecarei operatii masurate, fiecare grafic avand un titlu descriptiv care indica operatia pentru care s-a efectuat testul, o axa Ox ce reprezinta cele trei limbaje de programare analizate si o axa Oy care indica timpul masurat pentru operatii, exprimat in nanosecunde. In plus, aplicatia include o fereastra dedicata pentru afisarea timpilor exacti masurati (in nanosecunde) pentru fiecare operatie si dimensiune testata, iar acesti timpi sunt salvati automat intr-un fisier text, oferind atat o vizualizare grafica, cat si detalii exacte pentru documentarea completa a rezultatelor.

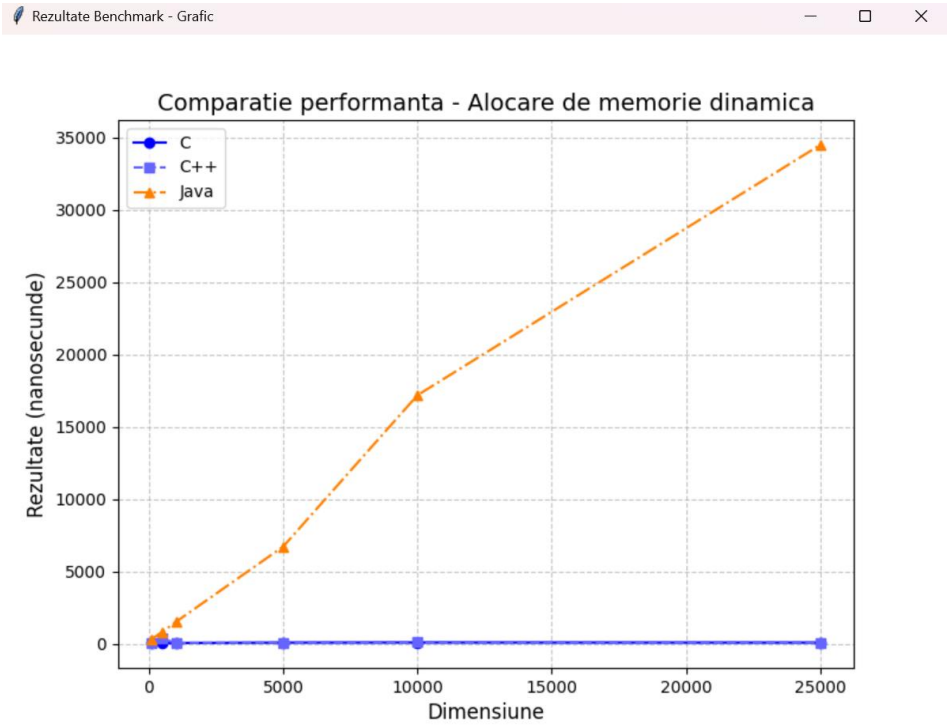
Masuratorile au fost realizate pe un sistem de operare Windows, utilizand un procesor Intel Core i9-12900H, asigurand astfel un mediu performant si stabil pentru testele efectuate.

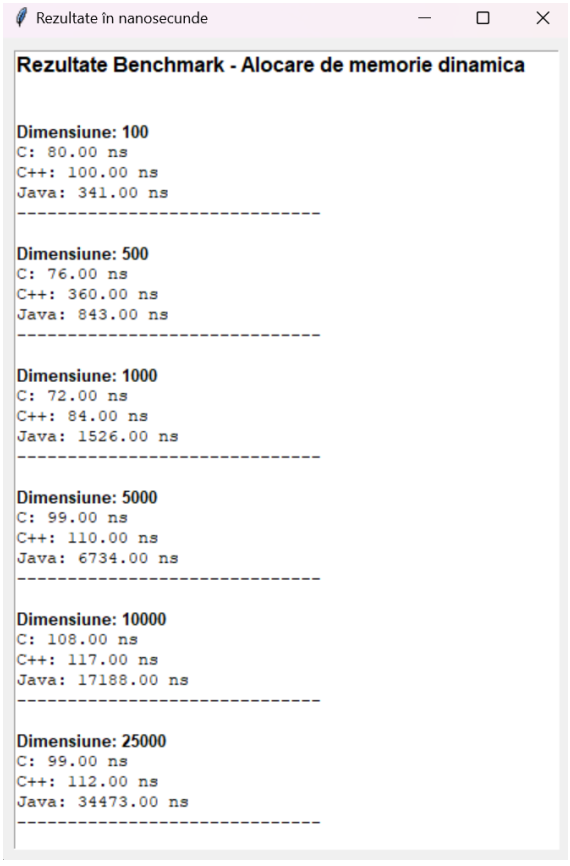
6.1 Alocarea de memorie statica



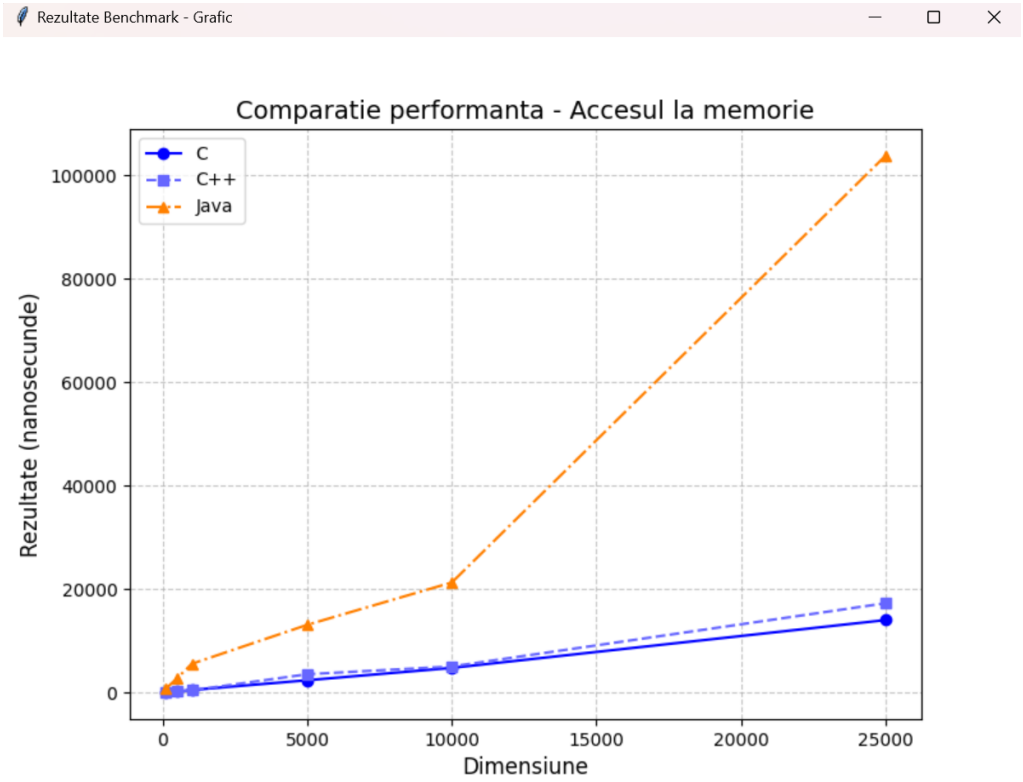


6.2 Alocarea de memorie dinamica

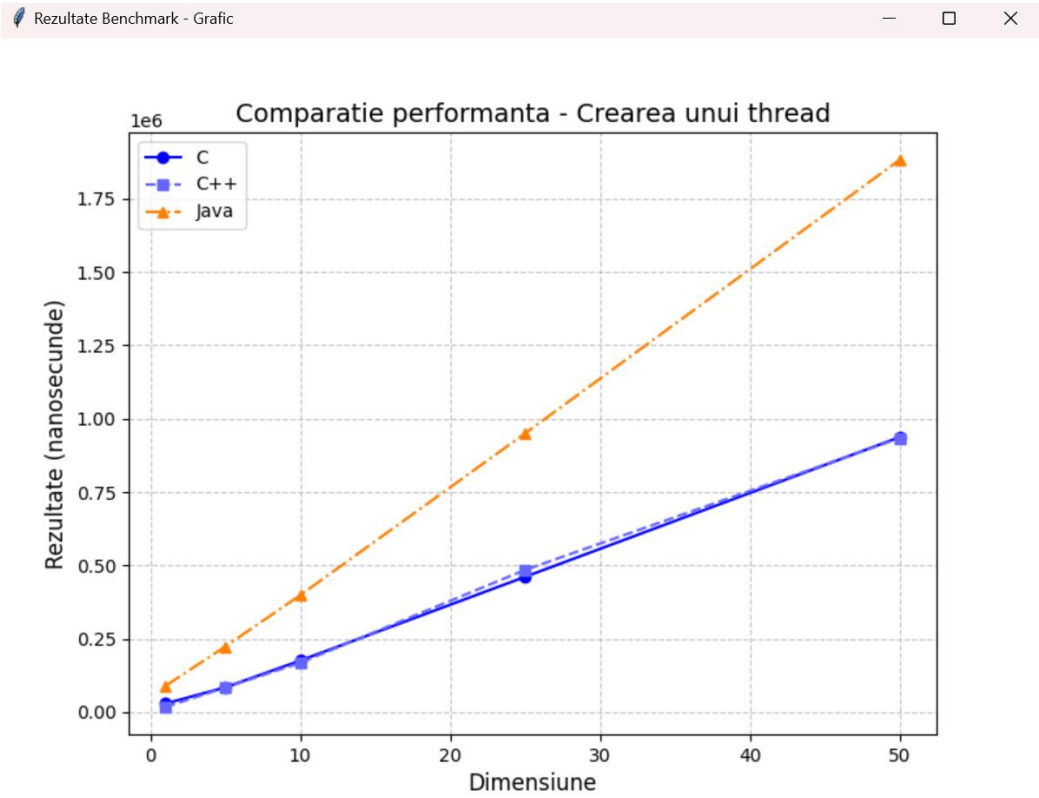


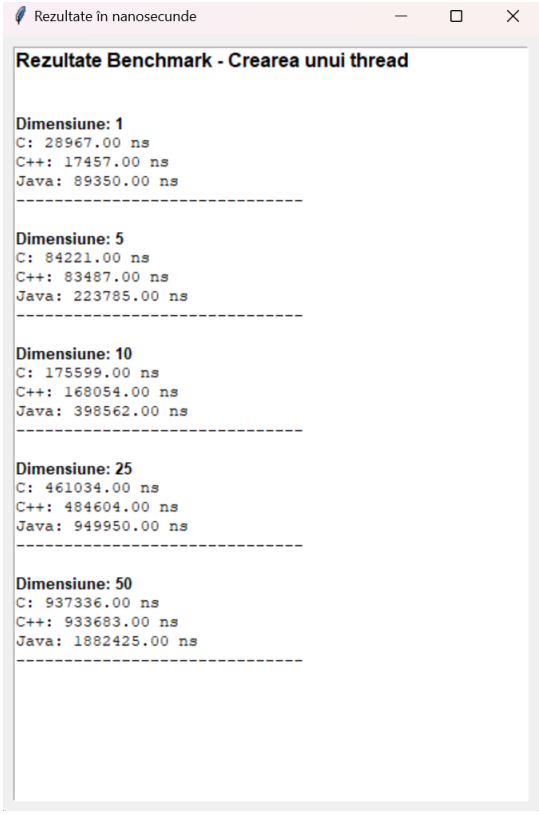


6.3 Accesul la memorie

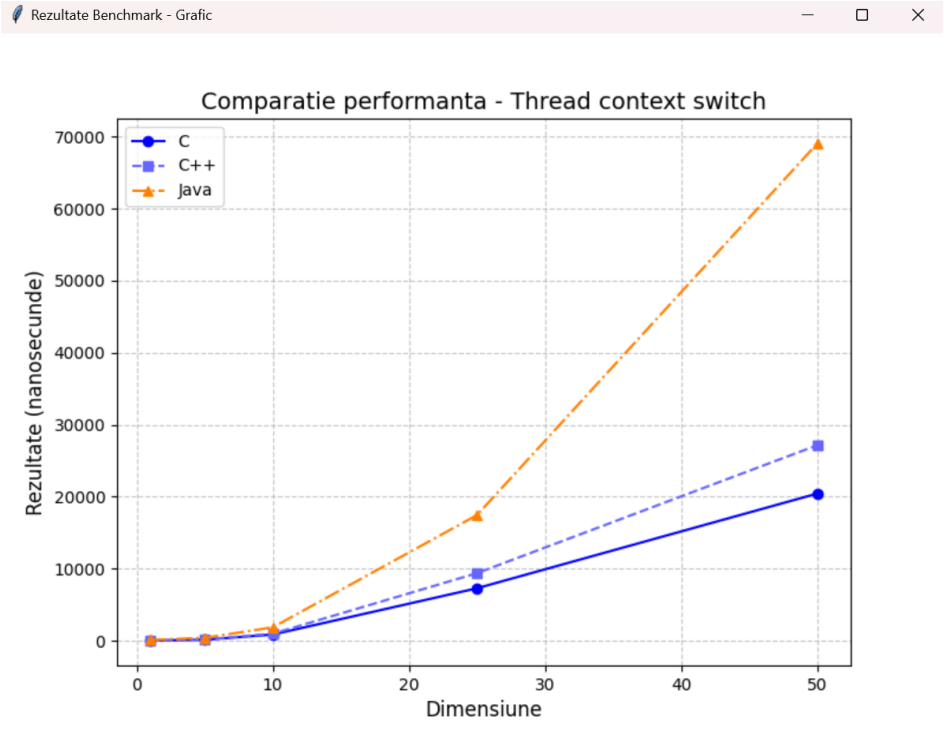


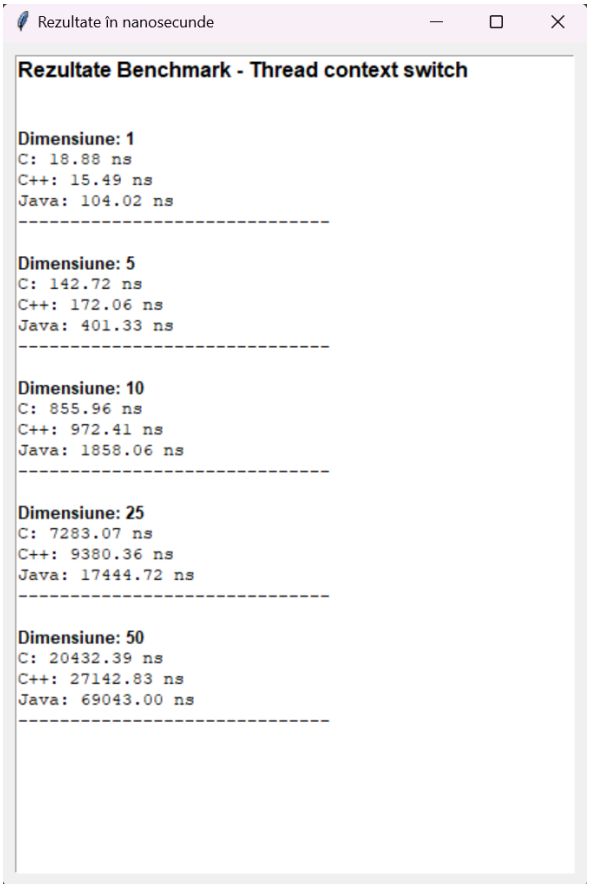
6.4 Crearea unui Thread



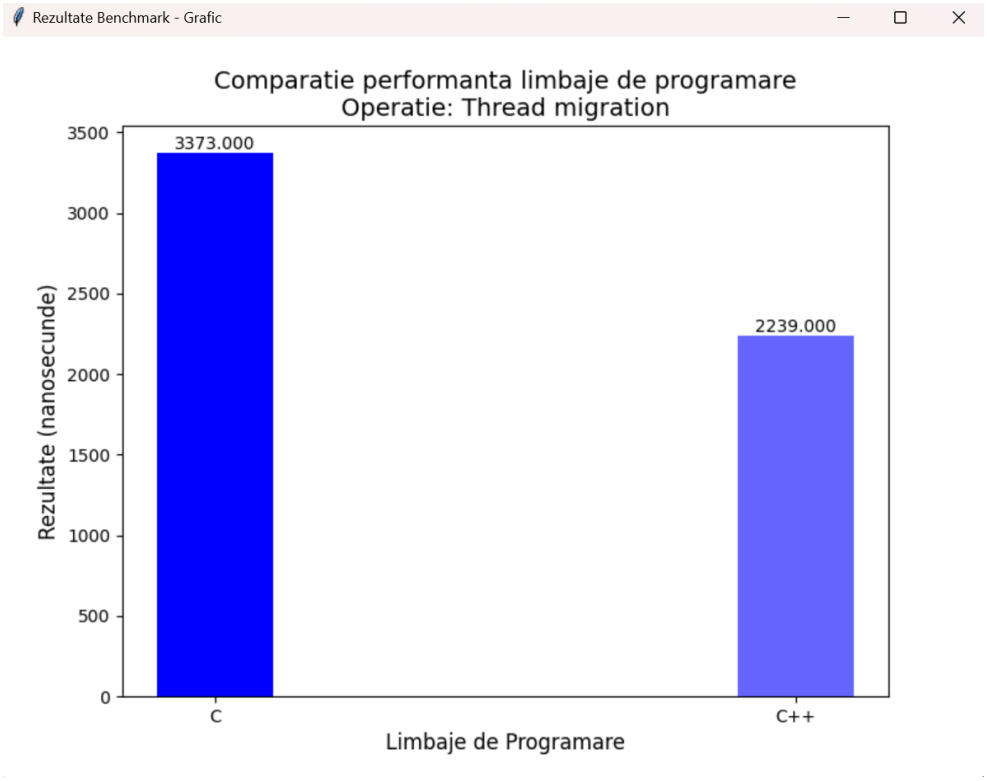


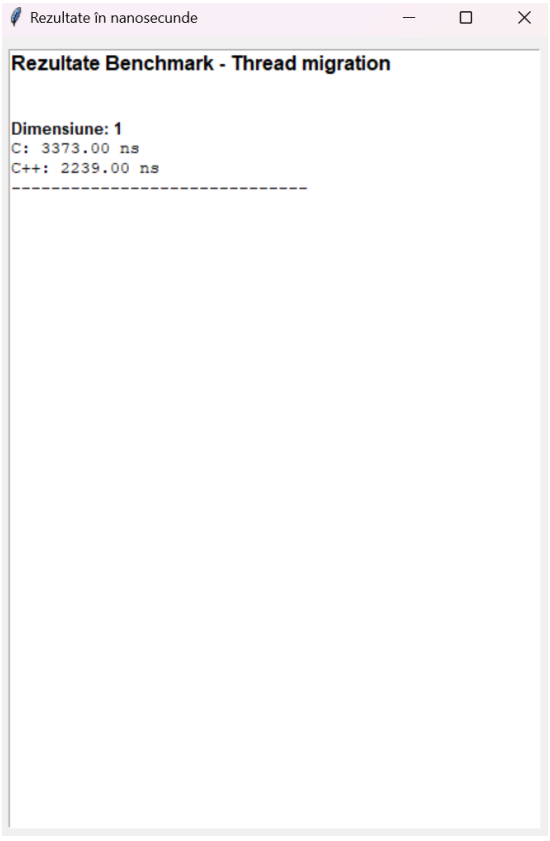
6.5 Thread context switch





6.6 Thread migration





Bibliografie

[1] Shauna Wilson, Paul Russell – „Remote Auditing Fundamentals, Second Edition”

URL: https://asq.org/quality-resources/benchmarking

[2] ADSERVIO Group – „What is Micro-benchmarking?”

URL: https://www.adservio.fr/post/what-is-microbenchmarking

[3] By Ravikiran A S – „All You Need to Know About C++ Memory Management”

URL: https://www.simplilearn.com/tutorials/cpp-tutorial/cpp-memory-management

[4] Sun Microsystems – „What Is a Thread?”

URL:https://www.iitk.ac.in/esc101/05Aug/tutorial/essential/threads/definition.html

[5] “Thread functions in C/C++”

URL: https://www.geeksforgeeks.org/thread-functions-in-c-c/

[7] Daniel – “Measure elapsed time with chrono lib in C++”

URL: https://doorsofstone.hashnode.dev/measure-elapsed-time-with-chrono-lib-in-cpp

[8] baeldung, Tom Hombergs – „Measure Elapsed Time in Java”

URL: https://www.baeldung.com/java-measure-elapsed-time

Plan de lucru

- Saptamana 3-4: Completarea introducerii si a cercetarii bibliografice
- Saptamana 5-6: Completarea analizei si a design-ului
- Saptamana 7-8: Implementarea unor programe pentru alocarea de memorie, accesul la memorie, crearea de threaduri, thread context switch, thread migration
- Saptamana 9-10: Realizarea interfetei grafice si preluarea datelor de intrare de pe interfata grafica
- Saptamana 11-12: Conectarea interfetei grafice cu programele implementate si realizarea graficelor