UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

# Măsurarea timpului de execuție în diferite limbaje de programare

*Structura Sistemelor de Calcul*

Autor: Stăvar Sebastian
Grupa: 30231

FACULTATEA DE AUTOMATICĂ
ȘI CALCULATOARE

15 ianuarie 2024

# Cuprins

# 1 Introducere

## 1.1 Context

În cadrul acestei prezentări, se propune realizarea unui proiect axat pe dezvoltarea unor programe dedicate măsurării aspectelor precum alocarea memoriei, accesul la memorie (atât static, cât și dinamic), crearea de thread-uri, comutarea contextului între acestea și migrația thread-urilor.

## 1.2 Specificații

Pentru implementarea programelor de măsurare, se va utiliza mediul de dezvoltare integrat specific fiecărui limbaj de programare: Intellij pentru Java, Visual Studio 19 pentru C++, și PyCharm 2023 pentru Python.

Aceste programe vor fi executate pe un laptop ASUS ExpertBook echipat cu un procesor Intel Core i3-1115G4, cu o frecvență de 3.00GHz, și o memorie RAM de 16GB, ce operează la o frecvență de 3200 MHz. Dispozitivul rulează pe sistemul de operare Windows 11 și este conectat la sursă de alimentare pentru a beneficia de întreaga capacitate de performanță a dispozitivului.

## 1.3 Obiective

Scopul proiectului constă în implementarea programelor de măsurare a execuției proceselor în trei limbaje de programare distincte, facilitând observarea comportamentului fiecărui limbaj în contextul unei implementări cât mai similare.

Rezultatele obținute în urma fiecărei cerințe vor fi meticulos documentate și încorporate într-un tabel Excel. Acest tabel va oferi datele necesare pentru generarea de grafice, contribuind astfel la conturarea unei imagini clare asupra descoperirilor realizate în cadrul proiectului.

# 2 Studiu Bibliografic

## 2.1 Măsurarea timpului de execuție

Măsurea timpului de execuție ar putea fi considerată o operațiune ușor de realizat, doar folosim un cronometru la lansarea execuției programul, apoi îl oprim la finalizarea lui. Acest tip de măsurătoare se numește “wall-clock time” și nu este validă pentru a caracteriza un algoritm implementat pe un calculator.

Astfel, un etalon (benchmark) descrie performanța unui program pe o mașină specifică la un moment al zilei. Procesoare diferite pot avea rezultate complet diferite. Chiar dacă lucrăm pe aceeași mașină, pot există uneori rezultate diferite, iar de asta se prefer măsurarea timpului de execuție într-un mod mai abstract, diferit de la un limbaj de programare la altul.

## 2.2 Alocarea de memorie

Alocarea de memorie este procesul în care o secțiune de memorie este rezervată într-un program pentru a utilizată în operații de stocare a unor variabile și instanțe de structuri de date și clase. Aceasta poate fi de două feluri: statică și dinamică.

Alocarea statică este utilizată atunci când se declară o variabilă, structură sau clasă la începutul unei clase sau funcții. Memoria este alocată de către sistemul de operare, iar numele declarat va fi utilizat pentru accesul la blocul de memorie.

Alocarea dinamică este utilizată atunci când programul rulează apeluri precum new (C++) sau malloc (C). Sistemul de operare desemnează un bloc de memorie de o mărime specifică în timp ce programul se află în execuție. Atunci când un bloc de memorie este alocat, un pointer spre acel bloc este returnat care facilitează accesul spre datale stocate acolo.

### 2.3 Accesul la memorie

Accesul la memorie este procesul în care programul ajunge în posesia datelor care se află în anumite blocuri de memorie atunci când sunt alocate, fie static, fie dinamic.

Accesul la memorie în modul static se realizează prin utilizarea numelor variabilelor și a indicilor.

Accesul la memorie în modul dinamic se realizează aproximativ ca în modul static, doar că se utilizează și pointeri.

### 2.4 Crearea unui thread

Thread-ul este unealta care ne permite efectuarea operațiilor complexe fără a fi întâmpinate probleme în programul principal. De obicei, dorim să lucrăm cu thread-uri când dorim să eficientizăm timpul de execuție al unui program sau când dorim să împărțim cantitatea de muncă în cantități mai mici.

Thread-uri sunt create în diferite moduri, în funcție de limbajul de programare utilizat.

### 2.5 Thread context switching

Thread context switching este un process eficient și recomandat din cauza nivelului scăzut de resurse folosite, deoarece implică doar schimbul de identități și resurse, precum counterul de program, regiștrii și pointerii stivei. Costul acestei operații este aproximativ egal cu cel al deschiderii și închiderii kernel-ului.

Un context switch pe Thread-uri se poate întâmpla când procesorul salvează starea curentă al firului de execuție și schimbă pe un alt fir de execuție din același proces, neimplicând schimbarea de adrese de memorie.

### 2.6 Thread migration

Thread migration este operațiunea în care procesorul mută execuția unui fir de execuție de pe un core al CPU-ului pe alt core într-un sistem multi-core sau care este multi-CPU.

Este de obicei utilizată pentru balansarea cantității de lucru pe un procesor, îmbunătțind performanța sistemului și optimizează utilizarea resurselor. Se poate întâmpla la diferite nivele de component, fie hardware sau software, cum ar fi sistemul de operare, rutine de mediu sau chiar a componentelor fizice ale sitemului.

## 3 Analiză

### 3.1 Măsurarea timpului de execuție

În limbajul de programare C++, timpul de execuție este evaluat utilizând **time point** din high resolution clock.

În Java, metoda utilizată pentru această măsurare este **System.nanoTime()**.

În Python, timpul de lansare al programului este salvat cu ajutorul funcției **time()**, iar diferența dintre timpul final și cel inițial este calculată la finalul codului.

### 3.2 Operații

| Operație | Descriere |
|---|---|
| **Alocarea de memorie** | • C++: Alocarea dinamică se realizează cu ajutorul funcțiilor 'malloc', 'calloc' și 'new'.<br>• Java: Se utilizează operatorul 'new' pentru alocarea obiectelor pe heap.<br>• Python: Echivalentul operatorului 'new' este utilizarea constructorului, având în vedere că orice element în Python este, de fapt, un obiect, iar majoritatea variabilelor sunt referințe. |
| **Accesul la memorie (static și dinamic)** | • C++: Accesul la memorie se măsoară cu operații de citire și scriere în variabile statice sau alocate dinamic.<br>• Java: Se oferă acces la memorie prin variabile statice și prin obiecte alocate pe heap.<br>• Python: Desigur, este un limbaj cu gestionare automată a memoriei, dar se pot utiliza module precum 'ctypes' pentru acces direct la memorie în limbajul C. |
| **Crearea unui thread** | • C++: Se utilizează biblioteca 'thread', iar un thread se instanțiază în funcția principală astfel: std::thread t1(<funcție>, <referința pointerului asupra căruia dorim să operăm>, <id>);<br>• Java: Se implementează o clasă care extinde clasa Thread sau implementează interfața Runnable, iar un Thread se instanțiază în funcția principală astfel: Thread t1 = new Thread(new ClassT(<nume>)); t1.start();<br>• Python: Se utilizează modulul 'threading', iar un thread se instanțiază astfel: t1 = threading.Thread(target=<funcție>) t1.start() t1.join(). |
| **Thread context switching** | • C++: Utilizarea mutex-urilor pentru gestionarea context switching-ului.<br>• Java: Operația este asigurată de JVM și sistemul de operare, dar există funcții pentru sincronizare și așteptarea thread-urilor, implicând schimbarea contextului.<br>• Python: Se utilizează funcții de sincronizare precum 'yield_thread()' pentru a ceda controlul altui thread. |
| **Thread migration** | • C++: Utilizarea bibliotecilor 'thread' și 'chrono', cu funcția 'sleep_for' pentru gestionarea migrării thread-urilor.<br>• Java: Oferă facilități automate pentru gestionarea thread-urilor și migrarea lor pe diferite nuclee ale procesorului, iar ExecutorService poate gestiona distribuția volumului de muncă asupra firelor de execuție disponibile.<br>• Python: Controlată prin funcții de control al thread-ului, dar în general gestionată de sistemul de operare. |

Figura 1: Descrierea operațiilor

## 4 Design

Structura sistemului este constituită dintr-o interfață grafică dedicată utilizatorului, un compilator și un executor performant. Utilizatorul are posibilitatea de a alege unul sau mai multe limbaje de programare, inclusiv C++, Java și Python. De asemenea, se oferă opțiunea de a selecta una sau mai multe operații specifice. Rezultatele obținute sunt ulterior prezentate într-o formă vizuală coerentă, sub forma unui tabel sau a unei diagrame, pentru a facilita înțelegerea și analiza eficientă în cadrul mediului corporativ.

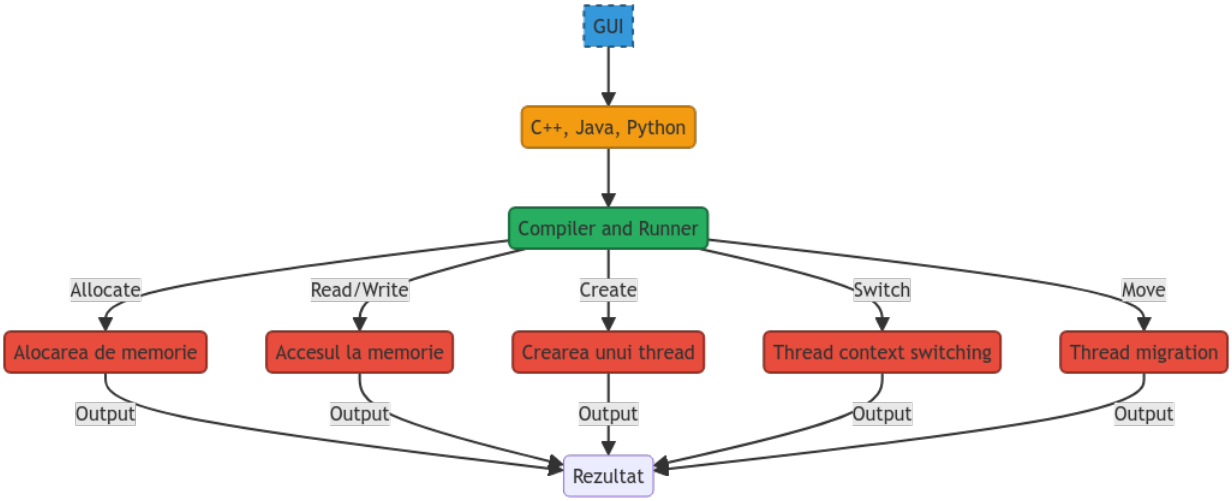


Figura 2: Structura sistemului

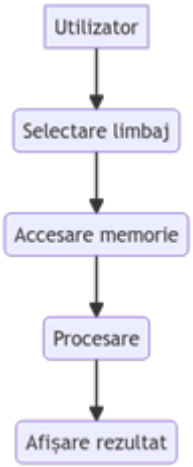


Figura 3: Access Memorie

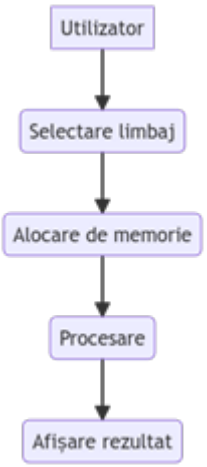


Figura 4: Alocare Memorie

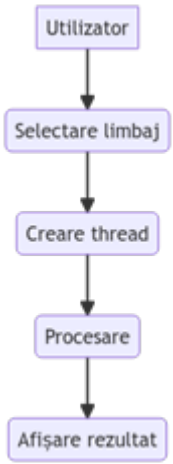


Figura 5: Creare Thread

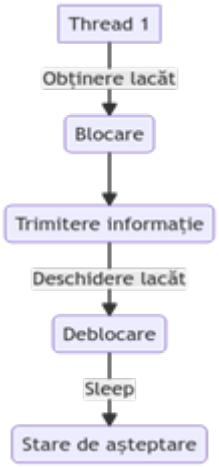


Figura 6: Thread Context Switching

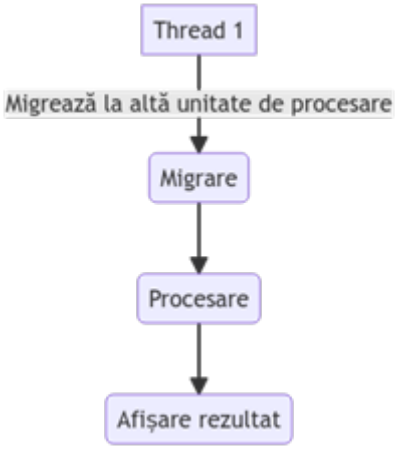


Figura 7: Thread Migration

În cadrul interfeței grafice consacrate utilizatorului, există posibilitatea de a selecta un număr specific de execuții prin intermediul unui script elaborat în limbajul de programare Python, care gestionează alegerea executabilelor. Prin intermediul unui alt script Python, se efectuează preluarea datelor obținute anterior și se realizează o reprezentare vizuală a informațiilor și performanțelor asociate acestora.

GUI-ul va fi realizat cu ajutorul limbajului Python și al librăriei Tkinter. Pentru a asigura executarea fisierelor, va trebui să luăm în considerare extensia fiecăruia la rulare: pentru C++, .cpp, pentru Java, .jar, și pentru Python, .py.

# 5 Implementare

Pentru a asigura un bun demers al acestei aplicații de măsurarea a performanței, fiecare program va opera asupra unei structuri cu 10000 de elemente, care este executat de 100 de ori.

Operațiile de alocare și accesare ale memoriei, atât statice, cât și dinamice, se vor efectua asupra unor vectori de dimensiunea 10000, număr ales din cauza limitării impuse de stivă: stack overflow.

| Operație | Limbaj | | |
|---|---|---|---|
| | C++ | Java | Python |
| **Alocarea de memorie statica** | int staticArray[value] | int[] staticArray = new int[value] | Se face automat când se creează o variabilă. |
| **Alocare de memorie dinamica** | int* dynamicArray = new int[size]<br>delete[] dynamicArray | int[] dynamicArray = new int[value] | Se face automat când se creează o variabilă |
| **Accesul la memorie statica** | staticArray[i] = i + 1 | staticArray[i] = i + 1 | static_array = [i + 1 for i in range(value)] |
| **Accesul la memorie dinamica** | dynamicArray [i] = i + 1 | dynamicArray[i] = i + 1 | import numpy as np<br>dynamic_array = np.arange(1, value + 1, dtype=int) |
| **Crearea unui thread** | vector<thread> threads;<br>threads.emplace_back(…)<br>thread.join() | List<Thread> threads = new ArrayList<>()<br>threads.add(new Thread() -> threadFunction(i))<br>thread.start(0 | import threading<br>thread = []<br>t = threading.Thread(target=…, args=…)<br>t.join() |
| **Thread context switching** | mutex coutMutex;<br>lock_guard<mutex> lock(coutMutex);<br>counter++;<br>this_thread::yield(); | Lock coutMutex = new ReentrantLock()<br>coutMutex.lock()<br>Thread.yield()<br>coutMutex.unlock() | import threading<br>cout_mutex = threading.Lock()<br><br>with count_mutex: counter += 1<br>time.sleep |
| **Thread migration** | this_thread::sleep_for( chrono::milliseconds()); | thread.add(new Thread() -> threadFunction(i))<br>thread.start(0 | import threading<br>time.sleep() |

Figura 8: Implementarea Operațiilor

Îm urma implementării am reușit să descopăr anumite asemănări și deosebiri între aceste trei limbaje de programre, dar și care aventajele și dezavantajele utilizării lor. Următoarele afirmații sunt realizate asupra observațiilor făcute asupra graficilor efectuate în secțiunea 6.

C++ este un limbaj de programare care este ”expert” în manipularea memoriei, obținând în efectuarea operațiilor cu memoria un timp aproape instant, deoarece alocarea și dezalocarea este efectuată de user și nu de un Garbage Collector. Fiind un limbaje de programre care este controlat majoritar de user și nu are multe ”ajutoare”, este adeseori cel mai rapid dintre cele trei.

Java este un limbaj de programare care a avut cele mai bune performanțe în timpul testelor de alocare și acces al memoriei, atât statică, cât și dinamică, dar a pierdut considerabil în categoria firelor de execuție.

Python este um limbaj de programare care este ”user friendly”, având o sintaxă ușor de înțeles la un nivel de începător, dar este un limbaj greu de perfecționat. Pe baza implementării și a graficilor din secțiunea 6 putem observa că acest limbaj este adeseaori pe locul 3. Am observat că acest limbaj are dificultăți în lucrul cu variabile statice, deoarece limbajul este construit pentru a instanția orice variabilă ca un obiect, fiind o referință, asemător cu tehnica din Java. Asemena, și operațiile pe fire de execuție s-au dovedit a fi un task mai dificil.

Toate trei limbajele beneficiază de metode și funcții destul de asemănătoare, fapt care a făcut traducerea operațiilor dintr-un limbaj în altul un proces relativ rapid. Unele operații sunt realizate automat de limbaje, cum ar fi alocarea și accesarea memoriei, atât static, cât și dinamic, de către Python. În Java, alocarea memoriei în ambele stiluri este realizată identic.

### 5.1 Compilare și rulare

Pentru a salva spațiu, proiectul conține doar fișierele sursă pentru fiecare operație implementată. De fiecare dată când utilizatorul dorește să folosească aplicația, are opțiunea de a compila și rula testele.

Atât compilația, cât și rularea, se efectuează cu ajutorul un script Python, care pe baza selecției din GUI, începe procesul de măsurare a timpului. Fișierele care conțin testele se află în folderul ”benchmark”.

### 5.2 Parser

Pasăm rezultatele obținu din teste unui GUI cu ajutorul unui script Python care primește o listă de tuple de forma **(limbaj, valoare)**. Totul este realizat cu ajutorul unui script Python.

### 5.3 GUI

GUI-ul este implementat în Python folosind modulul Tkinter. Conține 7 butoane de tip radio, reprezentând operațiile cerute, un text field în care vor fi afișate rezultatele obține în urma executării, un buton **Execute**, care execută testele, un buton **Show Graph** care deschide un popup care conține un grafic realizat cu ajutorul modulului matplotlib.

## 6 Testare

În această secțiune vor fi prezentate niște line grafuri cu fiecare operație pentru a putea observa cu ochiul liber existența unor spike-uri în legătură cu timpul de execuție.

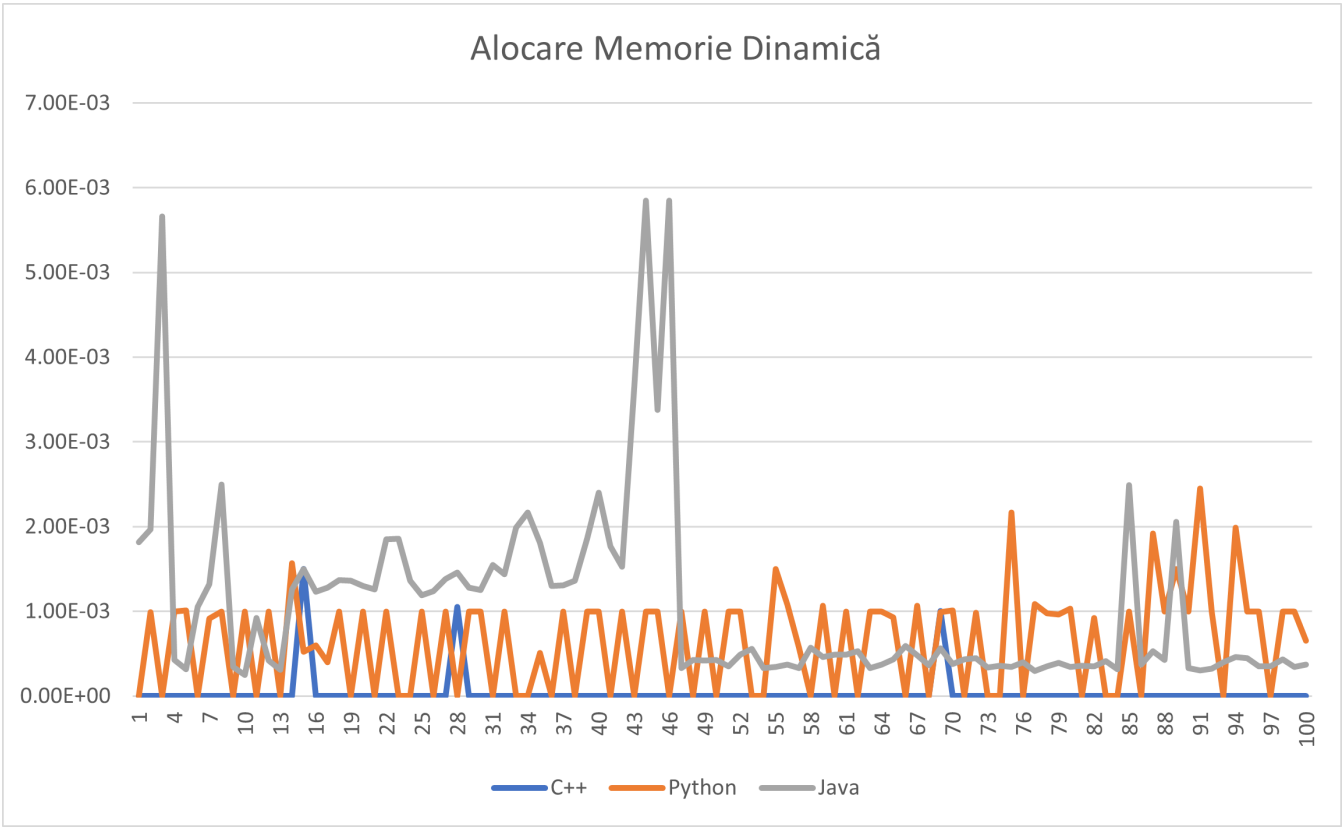


Figura 9: Line Graph Alocare Memorie Dinamică

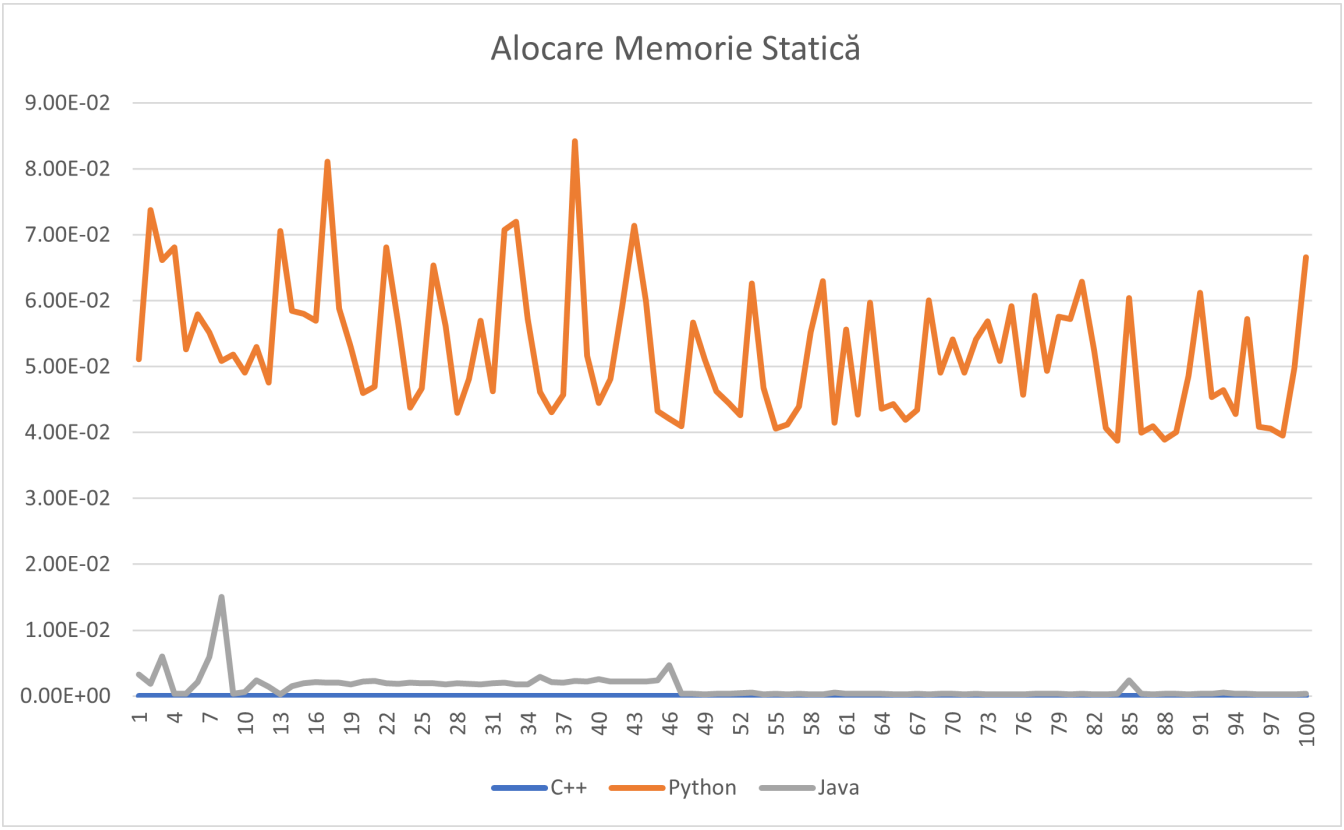


Figura 10: Line Graph Alocare Memorie Statică

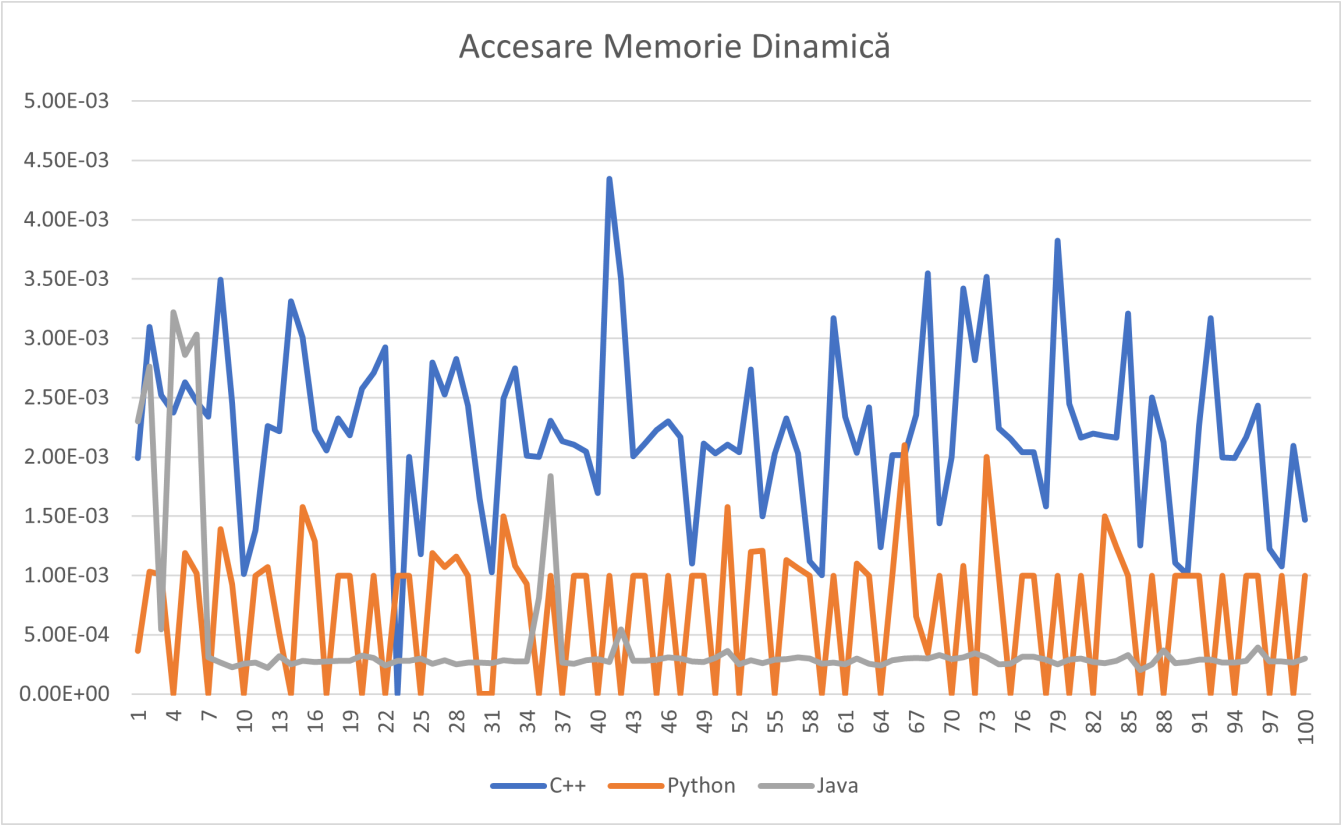


Figura 11: Line Graph Accesare Memorie Dinamică

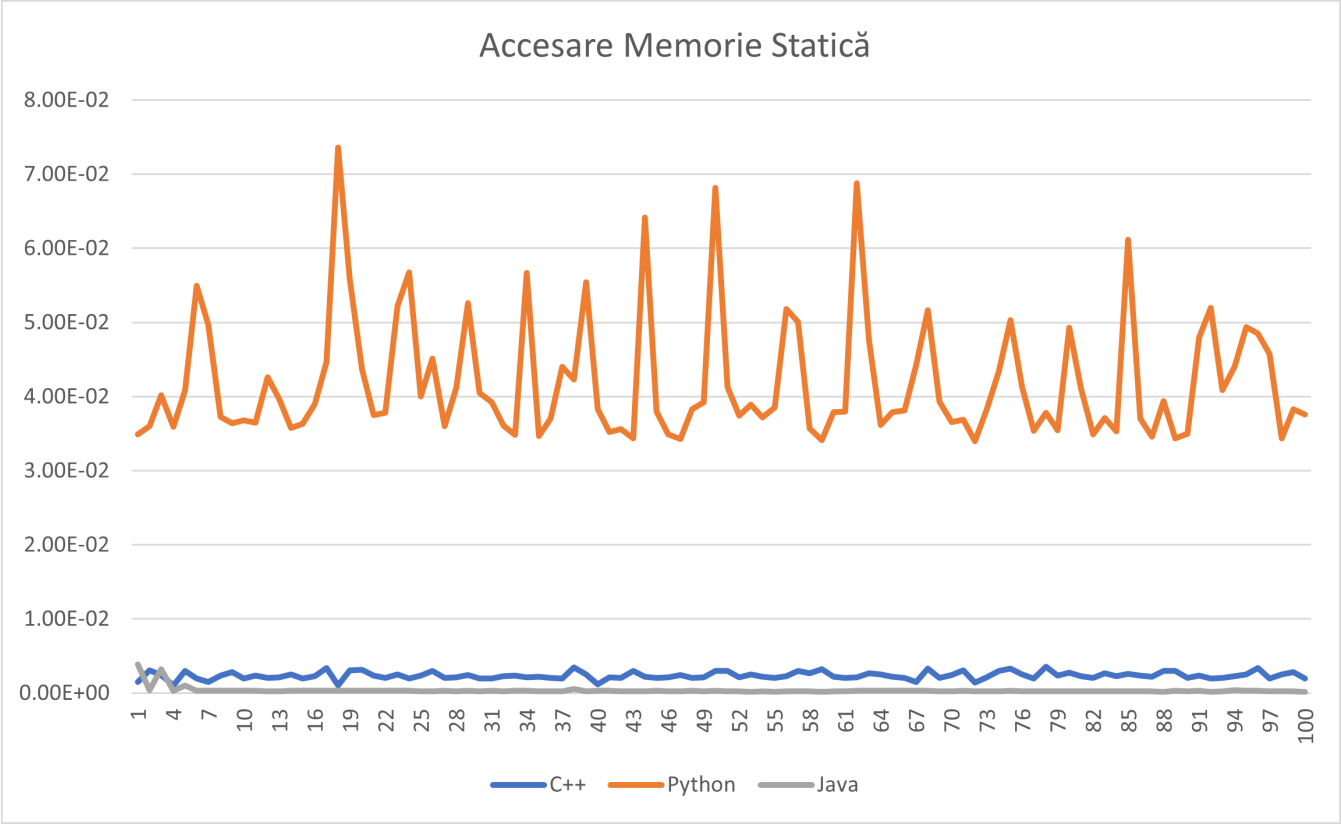


Figura 12: Line Graph Accesare Memorie Statică

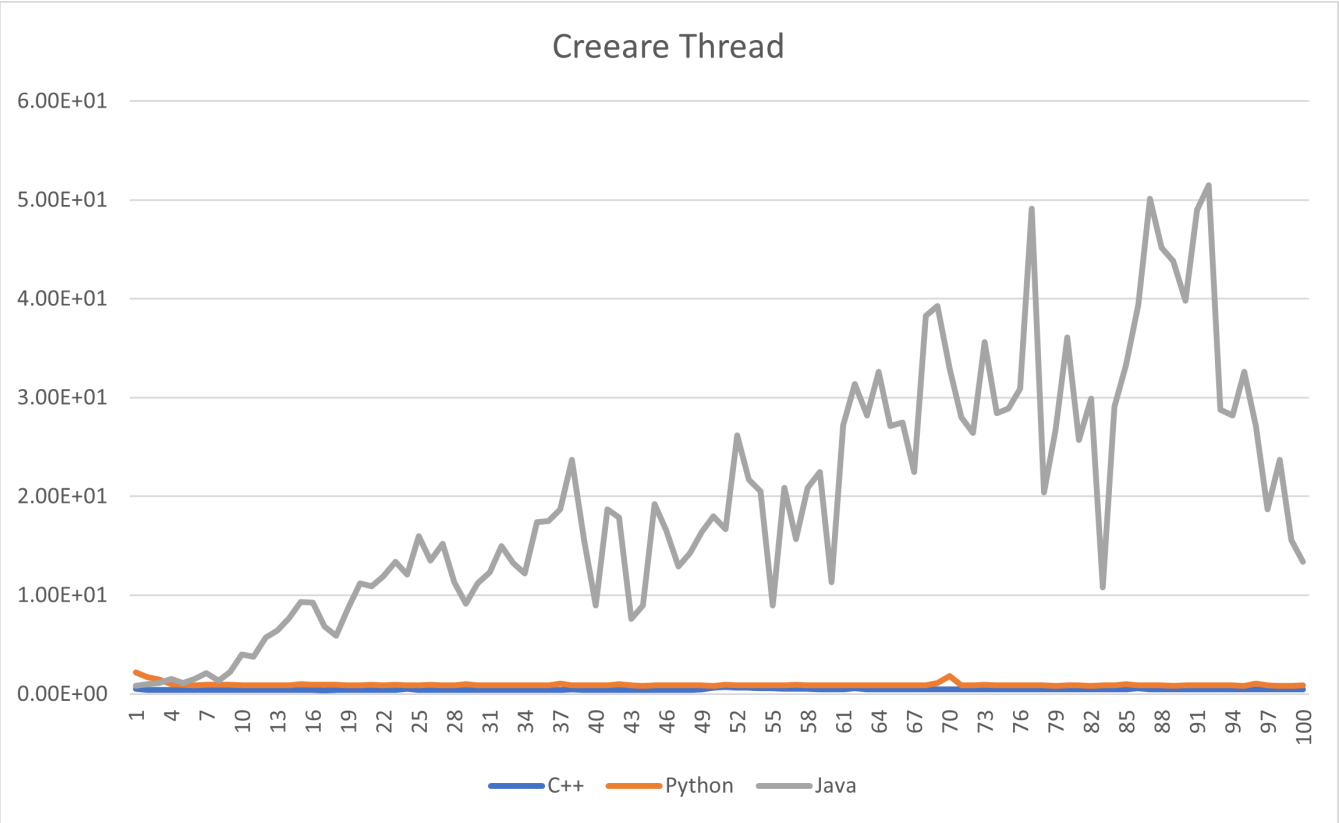


Figura 13: Line Graph Creeare Thread

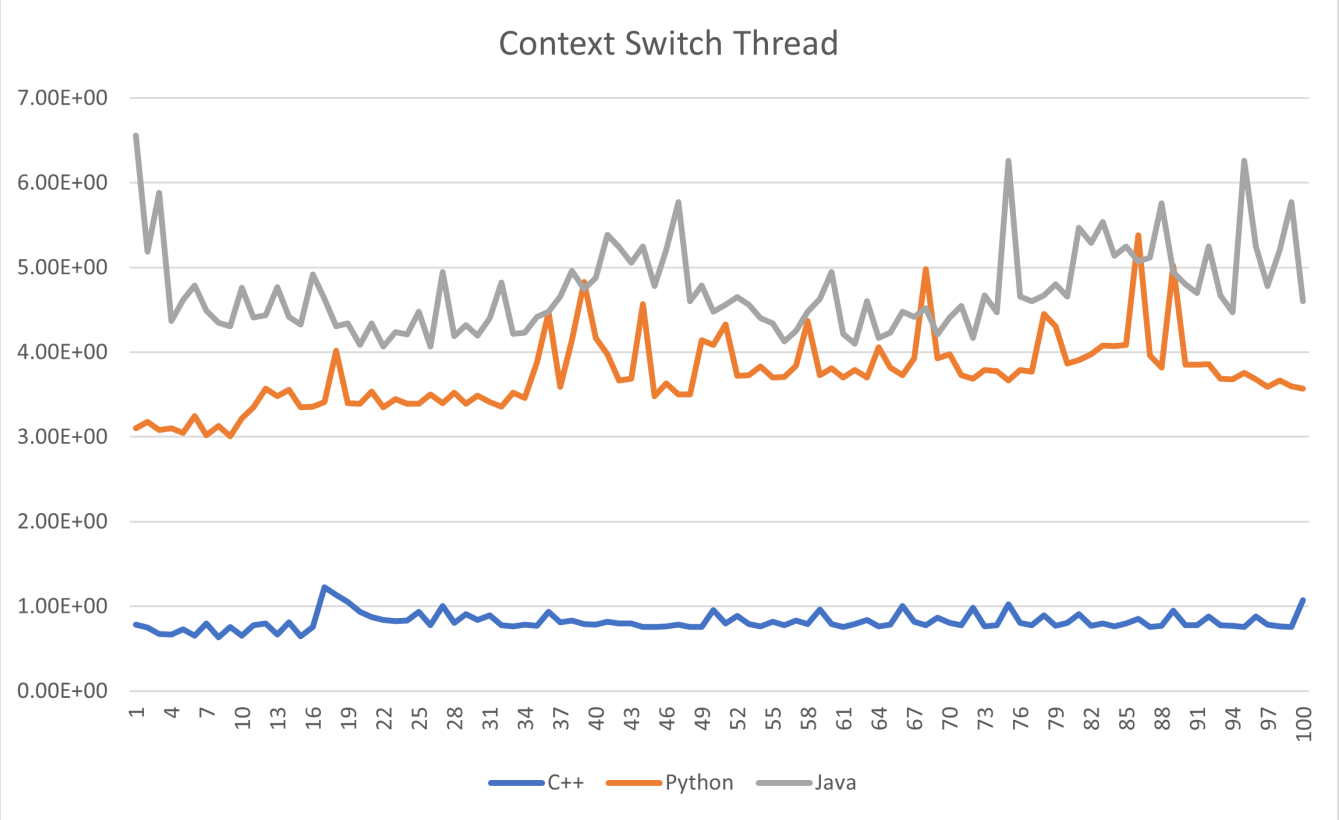


Figura 14: Line Graph Context Switch Thread

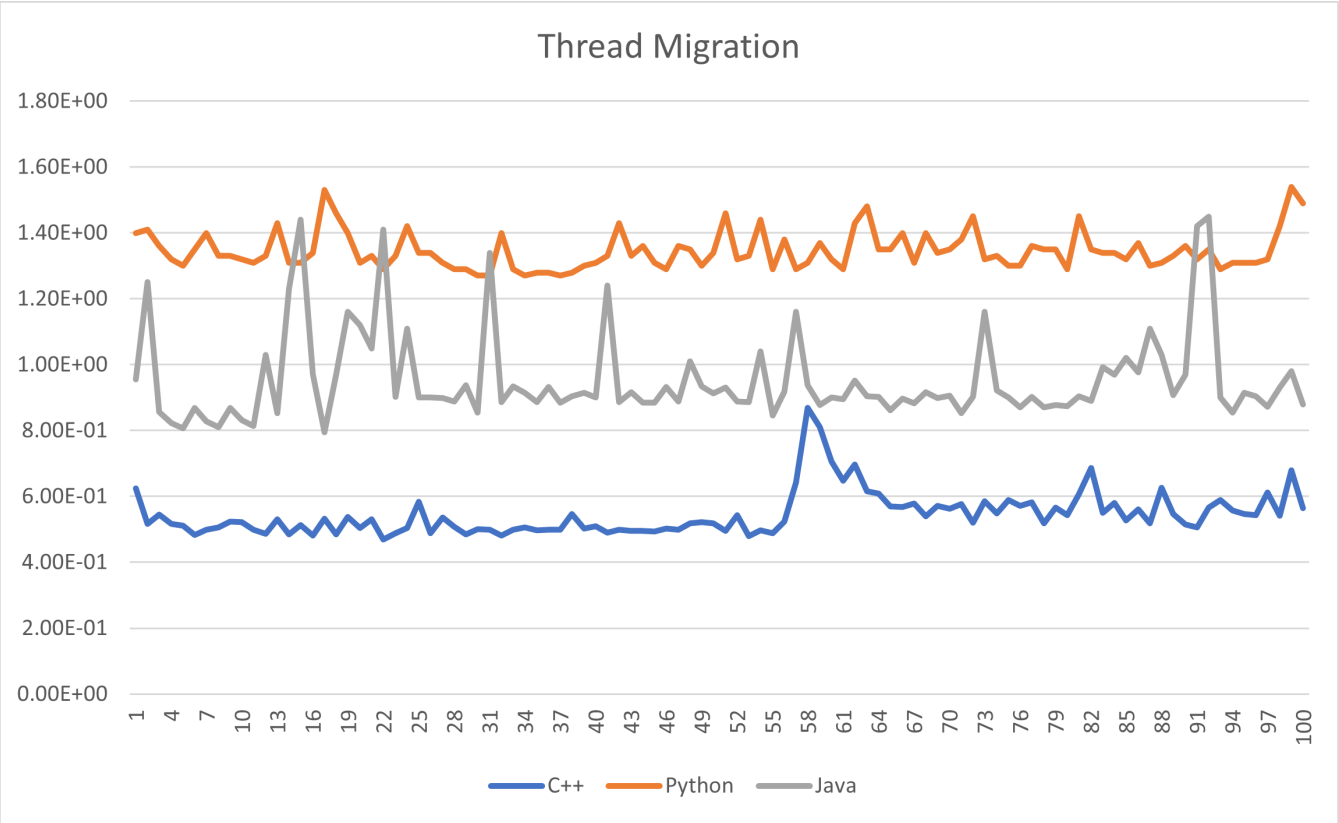


Figura 15: Line Graph Thread Migration

# 7 Concluzii

Aplicația finală arată conform figurii 9.

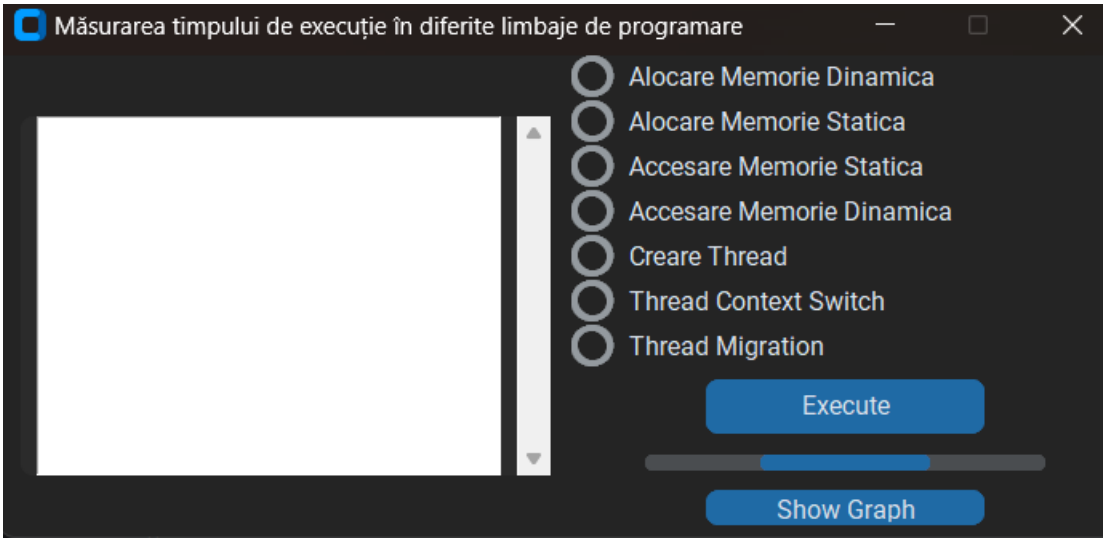


Figura 16: GUI

După ce testele sunt compilate și rulate, putem opta pentru a vizualiza rezulatele sub formă de grafic, conform figurii 10.

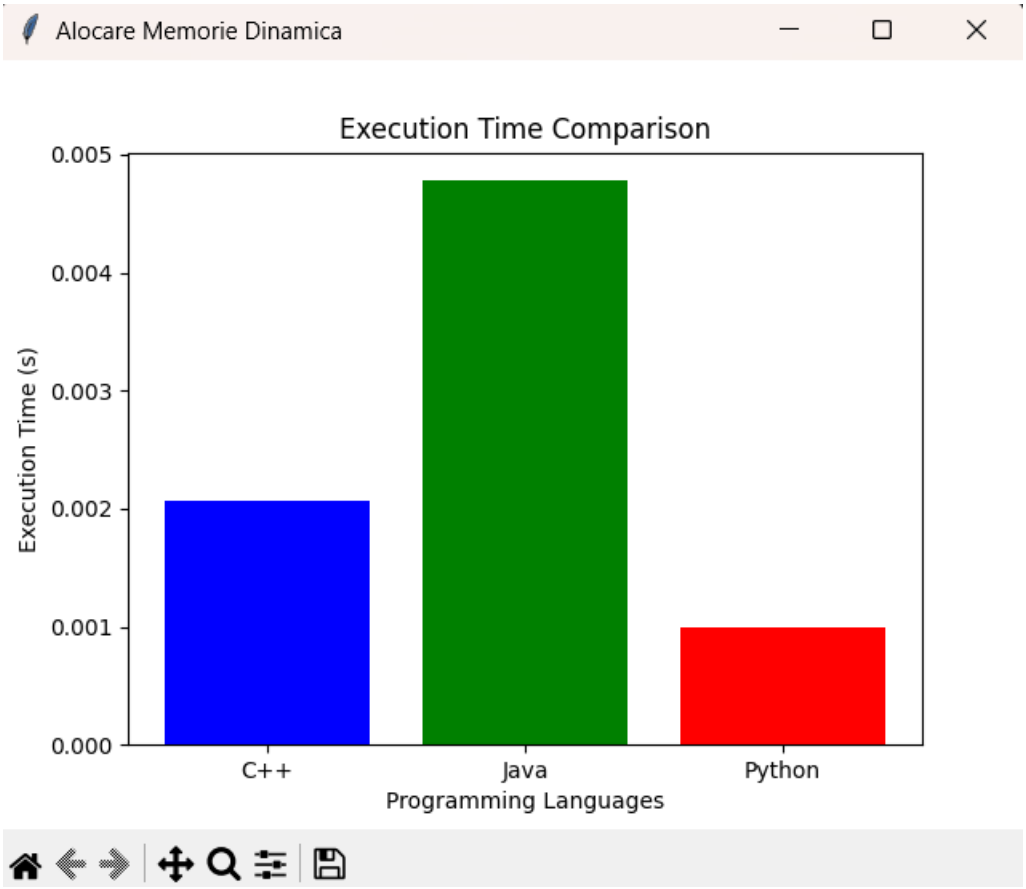


Figura 17: Exemplu de grafic

### 7.1 Posibile dezvoltări ulterioare

Acest proiect poate fi îmbunătățit în viitor prin eficientizarea codului și generalizarea lui la un nivel mai adânc. Totodată, introducerea mai multor limbaje de programare și operații pentru a putea determina performanța device-ului deținut.

O altă îmbunătățire ar putea consta în introducerea unui sistem care pe baza rezultatelor dorite asupra unor teste să îți poată oferi sugestii de calculatoare care ar putea satisface criteriile.

## 8 Bibliografie

1. Python Docs, [Online] https://docs.python.org/3/library/threading.html
2. College of Engineering, OSU, Chapter 4: Measuring Execution Time https://web.engr.oregonstate.edu/~sinisa/courses/OSU/CS261/CS261_Textbook/Chapter04.pdf
3. University of Alabama in Huntsville, Memory Allocation https://www.cs.uah.edu/~rcoleman/Common/C_Reference/MemoryAlloc.html
4. What are Threads in Java? https://www.freecodecamp.org/news/what-are-threads-in-java
5. Thread Context Switch, GeeksForGeeks, https://www.geeksforgeeks.org/difference-between-
6. Thread Migration, Usenix, https://www.usenix.org/legacy/events/usenix-nt99/full_papers/abdel-shafi/abdel-shafi.pdf