

Masurarea timpului de executie a proceselor in diferite limbaje de programare



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Opruta George

Grupa: 30232

1.Introducere

Scopul acestui proiect este de-a implementa un microbenchmark în 3 limbaje de programare diferite pentru a măsura timpul de execuție a unui proces bine definit. Prin acest microbenchmark se urmărește eficiența procesorului care execută operațiile descrise în cod, iar această eficiență este cuantificată în domeniul secundelor.

Microbenchmark-urile se ocupă de cel mai mic dintre toate benchmark-urile și reprezintă o măsură foarte simplă și ușor de definit, care urmărește și măsoară performanța unui fragment mic și specific de cod. Spre deosebire de benchmarking, care este procesul de rulare a unui program de calculator pentru a evalua cât de bine funcționează mediul de rulare. Un microbenchmark se referă întotdeauna la o cantitate foarte mică de cod. Prin urmare, sunt incredibil de rapid de implementat.

Redactarea și evaluarea eficientă a microbenchmark-urilor necesită angajament față de viteză și simplitate. Prin urmare, microbenchmark-urile ar trebui să aibă întotdeauna cât mai puțină suprasolicitare posibilă, și ar trebui să evite cu orice preț suprasolicitarea variabilă.

În scopuri de testare a performanței, abordarea ideală este să rulați o operațiune de mai multe ori, să zicem de 1.000 de ori, de fiecare dată când o testați. Puteți apoi să faceți o medie a rezultatelor, deși nu veți putea vedea cât de mult variază aceste rezultate între fiecare rulare. Acest fapt face ca microbenchmark-urile să fie mai bune pentru a testa ceva care durează întotdeauna aproximativ aceeași perioadă de timp.

Pentru a îndeplini cerința proiectului, microbenchmark-ul va consta dintr-o funcție simplă care calculează cel mai mare divizor comun dintr-o listă de perechi de numere în fiecare din limbajele de programare: C++, Java, Python. Astfel, se va măsura alocarea statică și dinamică a unei liste, accesarea memorie, cât și crearea de thread-uri pentru a optimiza task-ul. La final datele rezultate vor fi atașate acestui document.

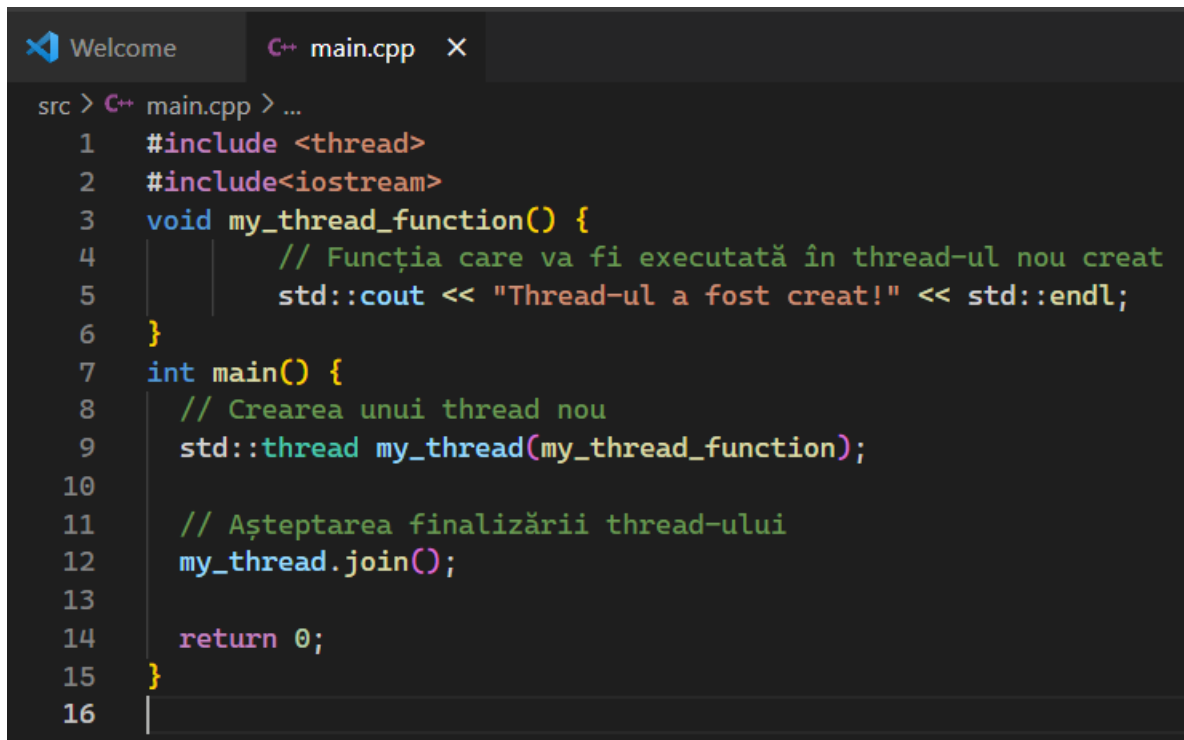
2.Studiu Bibliografic

În cadrul acestui proiect este necesară crearea unui array alocat atât static cât și dinamic prin funcții și sintaxa specifică fiecărui limbaj. De asemenea, se dorește crearea de thread-uri cu scopul de a vizualiza viteza de multi-threading.

Alocările de memorie, în toate cele 3 limbaje este extrem de asemănătoare din punct de vedere al sintaxei, cu specificarea că în limbajul de programare Python, nu se poate specifica de dinainte capacitatea array-ului, întrucât toate sunt definite astfel încât să fie de dimensiune variabilă

În cazul thread-urilor, fiecare limbaj are particularitatea sa.

În C++, crearea de thread-uri se realizează folosind biblioteca „thread”. Pentru a crea un thread, trebuie să furnizați o funcție care va fi executată în thread-ul nou creat. Această funcție se numește funcție de rutină a thread-ului (thread routine).

A screenshot of a code editor window with a dark theme. The window has a tab labeled 'C++ main.cpp'. The code is written in C++ and demonstrates creating a new thread. It includes the <thread> and <iostream> headers. A function named my_thread_function is defined, which prints 'Thread-ul a fost creat!' to the console. In the main function, a std::thread object is created by passing my_thread_function to its constructor. Then, my_thread.join() is called to wait for the thread to finish. Finally, the main function returns 0. Line numbers 1 through 16 are visible on the left side of the code.

```
src > C++ main.cpp > ...
1  #include <thread>
2  #include<iostream>
3  void my_thread_function() {
4      // Funcția care va fi executată în thread-ul nou creat
5      std::cout << "Thread-ul a fost creat!" << std::endl;
6  }
7  int main() {
8      // Crearea unui thread nou
9      std::thread my_thread(my_thread_function);
10
11     // Așteptarea finalizării thread-ului
12     my_thread.join();
13
14     return 0;
15 }
16
```

Figura 1 C++ Thread Creation

În Java, crearea de thread-uri se realizează folosind clasa Thread. Pentru a crea un thread, trebuie să se furnizeze o clasă care extinde clasa Thread. Această clasă va conține funcția de rutină a thread-ului.

```
import java.io.*;
class GFG implements Runnable {
    public static void main(String args[])
    {
        // create an object of Runnable target
        GFG gfg = new GFG();

        // pass the runnable reference to Thread
        Thread t = new Thread(gfg, "gfg");

        // start the thread
        t.start();

        // get the name of the thread
        System.out.println(t.getName());
    }
    @Override public void run()
    {
        System.out.println("Inside run method");
    }
}
```

Figura 2 Java Thread Creation

În Python, crearea de thread-uri se realizează folosind modulul threading. Pentru a crea un thread, trebuie să furnizați o funcție care va fi executată în thread-ul nou creat.

```
import threading

def my_thread_function():
    # Funcția care va fi executată în thread-ul nou creat
    print("Thread-ul a fost creat!")

def main():
    # Crearea unui thread nou
    my_thread = threading.Thread(target=my_thread_function)

    # Așteptarea finalizării thread-ului
    my_thread.start()

    # ...

if __name__ == "__main__":
    main()
```

Figura 3 Threading in Python

În final, mai este nevoie și de funcții care să măsoare durata execuției caracteristicilor descrise mai sus. Aceste funcții sunt de asemenea specifice fiecărui limbaj de programare, specifice unei librării, clase sau module, și returnează fie timpul calculat de procesor, în cazul limbajului C++, sau calculul direct și transformarea în milisecunde, în cazul limbajului Java și Python.

3. Analiza

Deși algoritmul prezentat mai sus are scopul de a solicita procesorul pentru a pune în lumină eficiența sa pe o anumită porțiune scurtă de cod, aceasta trebuie să poată fi și cuantificată în domeniul secundelor. Astfel, trebuie găsite anumite puncte cheie în cod pentru a inițializa un cronometru și a-l opri atunci când s-a efectuat porțiunea din cod, iar rezultatele să fie preluate, efectuate o medie asupra lor și notate.

Din punct de vedere al utilizatorului, acesta este pus în fața a 3 limbaje de programare diferite.

Pentru micro-benchmark-ul din C++, utilizatorul va porni un executabil, fără o interfață grafică, în care va specifica numărul de elemente din șirul de perechi de numere și numărul de

thread-uri. Numărul de elemente se refera la numărul de perechi pe care programul le va genera si le va stoca atât într-un sir alocat static cat si unul dinamic.

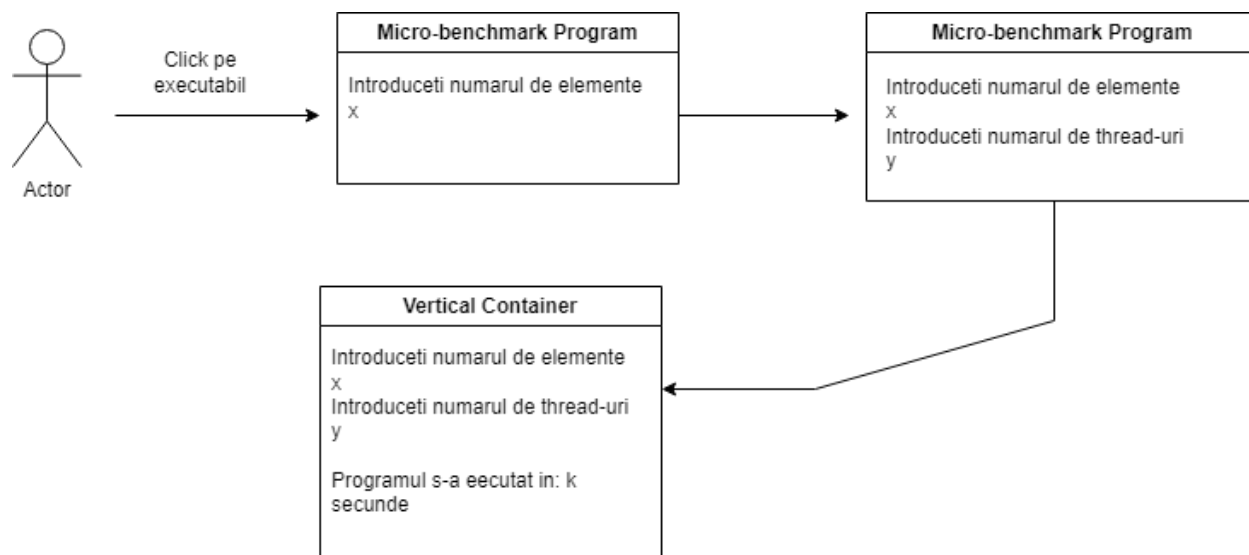


Figura 4 Folosirea aplicației CPP

Pentru cel din Java, utilizatorul va fi întâmpinat de-o interfață grafica care îl va informa ce date se așteaptă programul să primească, într-un mod mult mai explicit și ușor de înțeles. Textul pe care îl introduce va fi preluat de către program și va începe executarea acestuia. După ce programul s-a încheiat rezultatul va fi afișat într-o fereastră nouă, iar utilizatorului îi va fi prezentată opțiunea de a rula un micro-benchmark din nou, opțiune care dacă va fi selectată îl va trimite în fereastra principală de unde va putea să introducă date noi și să ruleze un nou micro-benchmark.

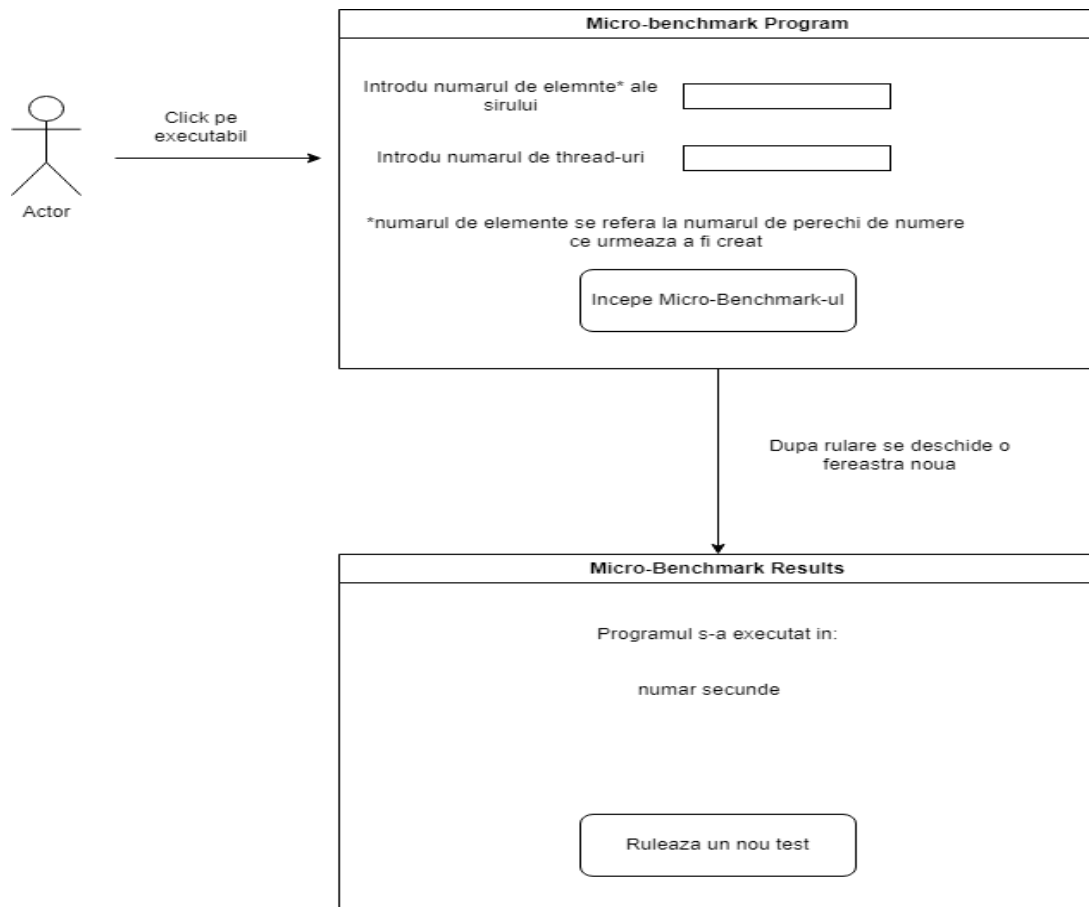


Figura 5 Folosirea aplicației în Java și Python

În final, programul din Python va avea și el o interfață grafică cu același scop ca și cea din Java.

4.Proiectare

Modelul de baza al proiectului îl reprezintă o serie de invocări ale funcțiilor de calculare a timpului in anumite porțiuni ale codului. Astfel, pe baza acestui model se vor structura toate aplicațiile scrise in cele 3 limbaje de programare.

In primul rand, se asteapta inputul de la user ca va specifica dimensiunea sirului care urmeaza sa fie declarat (static si dinamic) si numarul de thread-uri dorite sa indeplineasca task-ul.

Pe urmă, cu datele primite si va initializa pe rand sirul static si cel dinamic. Inaintea fiecărei initializari se vor declara variabile in care se va stoca timpul, respectiv in care se va calcula timpul in care au fost create.

In functie de numarul de thread-uri alese, fiecare thread va imparti sirul si va prelucra datele din portiunea sa. Cronometrul pentru thread-uri va incepe inainte de crearea lor si se va termina dupa ce toate si-au terminat task-ul.

In final, rezultatele de la variabile care calculeaza timpul de executie vor fi scrise intr-un fisier text din care vor fi prelucrate datele rezultate.

PSEUDOCOD:

Preia datele de la utilizator: nr = nr. elemente is sir, noThreads = nr. thread-uri

Porneste un cronometru pentru sirul alocat static

Aloca sirul

Opreste cronometrul si calculeaza timpul alocarii

Porneste un cronometru pentru crearea/executie de thread-uri

Porneste noThreads thread-uri

Join threads

Opreste cronometrul pentru thread-uri

Porneste un cronometru pentru sirul alocat dinamic

Aloca sirul

Opreste cronometrul si calculeaza timpul alocarii

Porneste un cronometru pentru crearea/executie de thread-uri

Porneste noThreads thread-uri

Join threads

Oprește cronometrul pentru thread-uri si calculeaza timpul final

Scrie in fisier rezultatele finale

Astfel, pe baza acestui pseudocod se pot construi ușor cele 3 aplicații de microbenchmark in limbajele de programare C++, Java, Python.

5.Implementare

Codul C++ măsoară și analizează performanța unui sistem în diverse scenarii, inclusiv alocarea statică și dinamică a memoriei, precum și prelucrarea șirurilor de numere.

Măsurarea Frecvenței Procesorului:

-Se utilizează instrucțiunile Time Stamp Counter (TSC) pentru a determina frecvența procesorului, furnizând informații despre viteza acestuia.

Alocare Statică:

-Se evaluează timpul necesar pentru alocarea statică a unui șir de dimensiune fixă, evidențiind performanța acestui tip de alocare și inițializare.

Alocare Dinamică:

-Se măsoară timpul de alocare și inițializare dinamică a unui șir, oferind date despre eficiența operațiilor de gestionare a memoriei.

Prelucrare Șir:

-Se calculează timpul de prelucrare al unui șir, utilizând o funcție pentru calculul celui mai mare divizor comun (cmmdc), atât în cazul alocării statice, cât și în cel al alocării dinamice.

Utilizarea Thread-urilor:

-Se investighează impactul utilizării firelor de execuție (thread-uri) asupra performanței, analizând timpul necesar pentru prelucrarea paralelă a șirului.

Ieșire în Fișier:

-Rezultatele măsurărilor, inclusiv frecvența procesorului și timpii operațiilor menționate mai sus, sunt salvate în fișierul "output.txt" pentru a permite analiza ulterioară.

Timpul Total de Execuție:

-Se calculează timpul total de execuție, oferind o perspectivă comprehensivă asupra performanței generale a codului în toate scenariile evaluate.

Această implementare oferă o abordare detaliată și structurată pentru evaluarea performanței sistemului în contextul alocării de memorie și prelucrării datelor, furnizând informații relevante pentru analiza și optimizarea ulterioară a codului.

Codul Java implementează o interfață grafică utilizator pentru efectuarea unui benchmarking de performanță, concentrându-se pe alocarea de memorie și prelucrarea unor șiruri de numere.

Interfața Grafică:

-Se construiește o interfață grafică utilizator (GUI) folosind JFrame, JTextField și JButton, pentru a permite utilizatorului să introducă numărul de teste și numărul de fire de execuție.

Măsurarea Frecvenței Procesorului:

-Se utilizează o funcție processor() pentru a evalua frecvența procesorului, măsurând timpul necesar pentru a realiza o serie de iterații și calculând astfel o estimare a frecvenței.

Alocare Statică și Dinamică:

-Se măsoară timpul necesar pentru alocarea și inițializarea unui șir de dimensiune fixă, atât în mod static, cât și dinamic, furnizând informații despre performanța alocării de memorie.

Prelucrare Șir:

-Se efectuează operații de prelucrare a unui șir, cu posibilitatea de paralelizare utilizând fire de execuție. Se măsoară timpul de execuție pentru aceste operații.

Utilizarea Thread-urilor:

-Se explorează impactul utilizării firelor de execuție (thread-urilor) pentru a paraleliza operațiile asupra șirului, cu scopul de a evalua eficiența și performanța în scenarii cu mai multe fire de execuție.

Leșire în Fișier:

-Rezultatele măsurărilor, inclusiv frecvența procesorului și timpii operațiilor enumerate mai sus, sunt salvate în fișierul "JAVA.txt" pentru analiza ulterioară.

Timpul Total de Execuție:

-Se calculează timpul total de execuție pentru a furniza o perspectivă asupra performanței globale a codului.

Această implementare oferă o interfață ușor de utilizat pentru efectuarea benchmark-urilor de performanță într-un mediu grafic, oferind totodată date relevante despre alocarea de memorie și timpul de execuție al operațiilor.

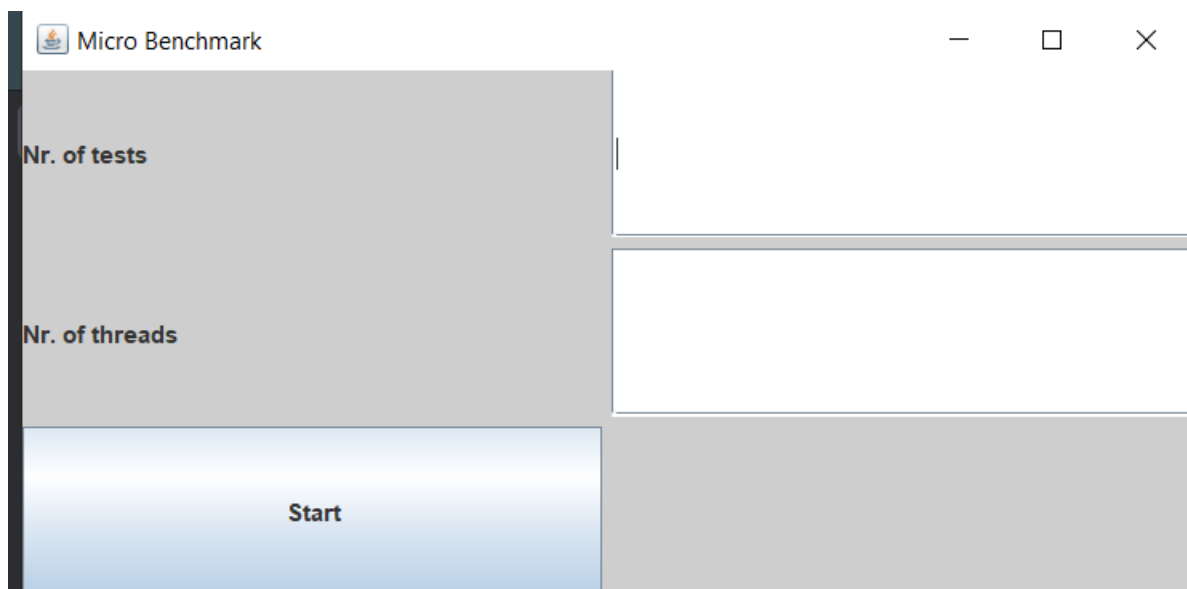


Figura 6 Interfata Java

Scriptul Python implementează un benchmark de performanță utilizând thread-uri și interfața grafică Tkinter. Iată o descriere detaliată a implementării:

Frecvența Procesorului:

-Se măsoară frecvența procesorului utilizând o funcție `processor()` care măsoară timpul necesar pentru așteptarea unei secunde și apoi calculează frecvența.

Interfața Grafică Tkinter:

-Se construiește o interfață grafică utilizator (GUI) cu Tkinter, pentru a permite utilizatorului să introducă numărul de teste și numărul de fire de execuție.

Alocare Statică și Dinamică:

-Se măsoară timpul necesar pentru alocarea și inițializarea unui șir de dimensiune fixă, atât în mod static, cât și dinamic, furnizând informații despre performanța alocării de memorie.

Prelucrare Șir:

-Se efectuează operații de prelucrare a unui șir, cu posibilitatea de paralelizare utilizând thread-uri. Se măsoară timpul de execuție pentru aceste operații.

Utilizarea Thread-urilor:

-Se explorează impactul utilizării thread-urilor pentru a paraleliza operațiile asupra șirului, cu scopul de a evalua eficiența și performanța în scenarii cu mai multe thread-uri.

Leșire în Fișier:

-Rezultatele măsurătorilor, inclusiv frecvența procesorului și timpii operațiilor enumerate mai sus, sunt salvate în fișierul "output.txt" pentru analiza ulterioară.

Timpul Total de Execuție:

-Se calculează timpul total de execuție pentru a furniza o perspectivă asupra performanței globale a scriptului.

Acest script furnizează o interfață grafică pentru efectuarea benchmark-urilor de performanță într-un mediu grafic, oferind totodată date relevante despre alocarea de memorie și timpul de execuție al operațiilor. De asemenea, rezultatele sunt salvate într-un fișier pentru analiză ulterioară.

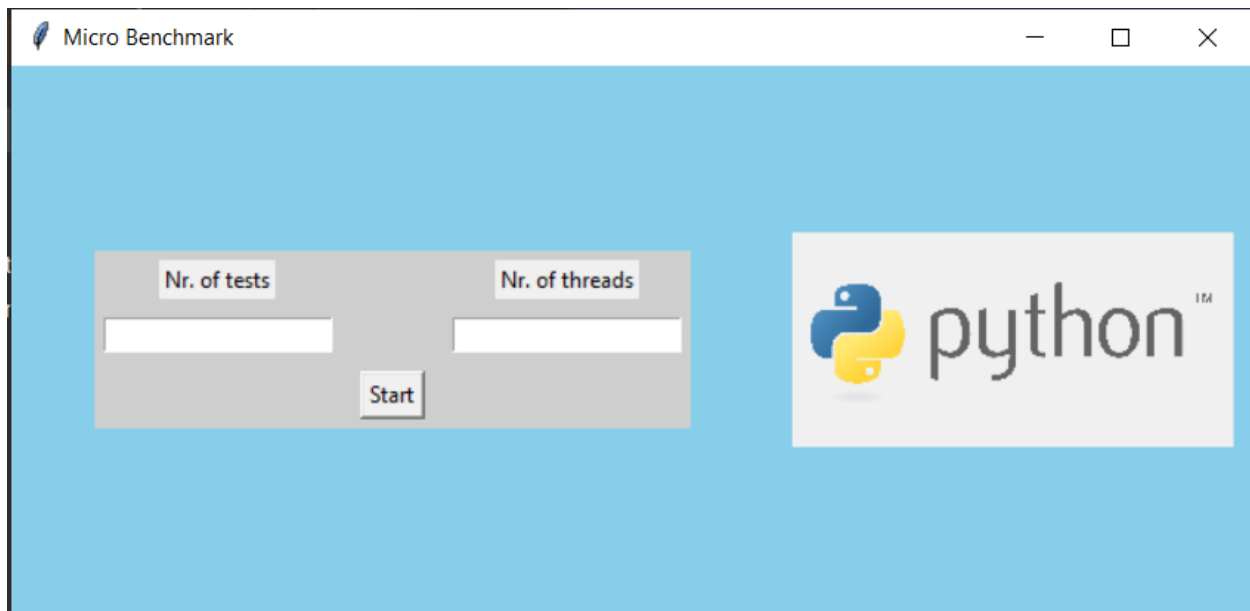


Figura 7 Interfața Python

6.Testare

Pentru testarea codului s-au dat parametrii necesari si s-au rulat programele de mai multe ori. S-a observat la fiecare test cum se schimba frecventa procesorului in functie si de alte task-uri care ruleaza in background si/sau sunt ale sistemului de operare.

CPP Test:

```

1 Frec:
2 3294849760.4798420532
3 Numar microsecunde pentru alocare statica:
4 0.0083220790
5 Numar microsecunde pentru initializarea sirului alocat static:
6 162.3544649642
7 Numar microsecunde pentru completare task pe sir alocat static cu 5 Thread-uri:
8 212.6618968806
9 Numarul de microsecunde pentru alocare dinamica:
10 6.1896690703
11 Numar microsecunde pentru initializarea sirului alocat dinamic:
12 75.1346519557
13 Numar microsecunde pentru completare task pe sir alocat dinamic cu 5 Thread-uri:
14 134.2894578345
15

```

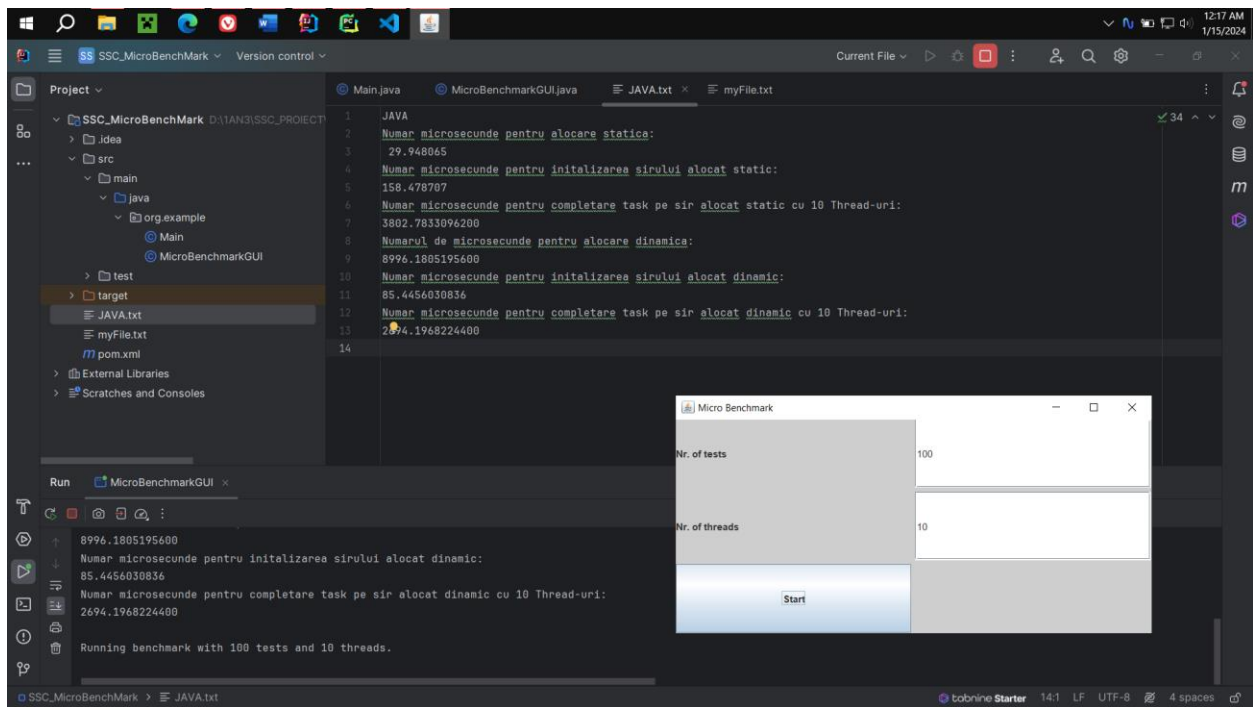
```

+ FullyQualifiedErrorId : CommandNotFoundException

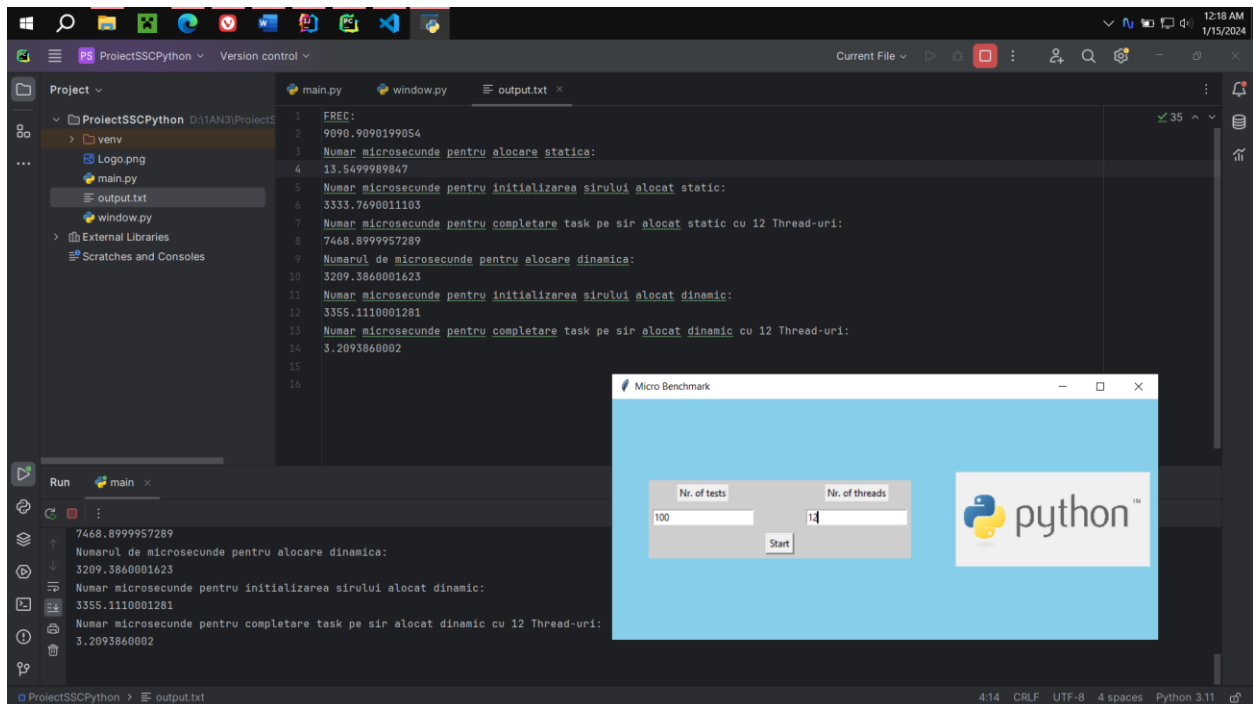
PS D:\IAN3\SSC_LABORATOR\SSC_Masurarea_Timpului_de_Executie> cd build
PS D:\IAN3\SSC_LABORATOR\SSC_Masurarea_Timpului_de_Executie\build> .\main
FREC:3.29485e+09
Specificati numarul de teste (nr_teste<=1000):
100
Specificati numarul de thread-uri
5
Numar microsecunde total 3979.431173
PS D:\IAN3\SSC_LABORATOR\SSC_Masurarea_Timpului_de_Executie\build>

```

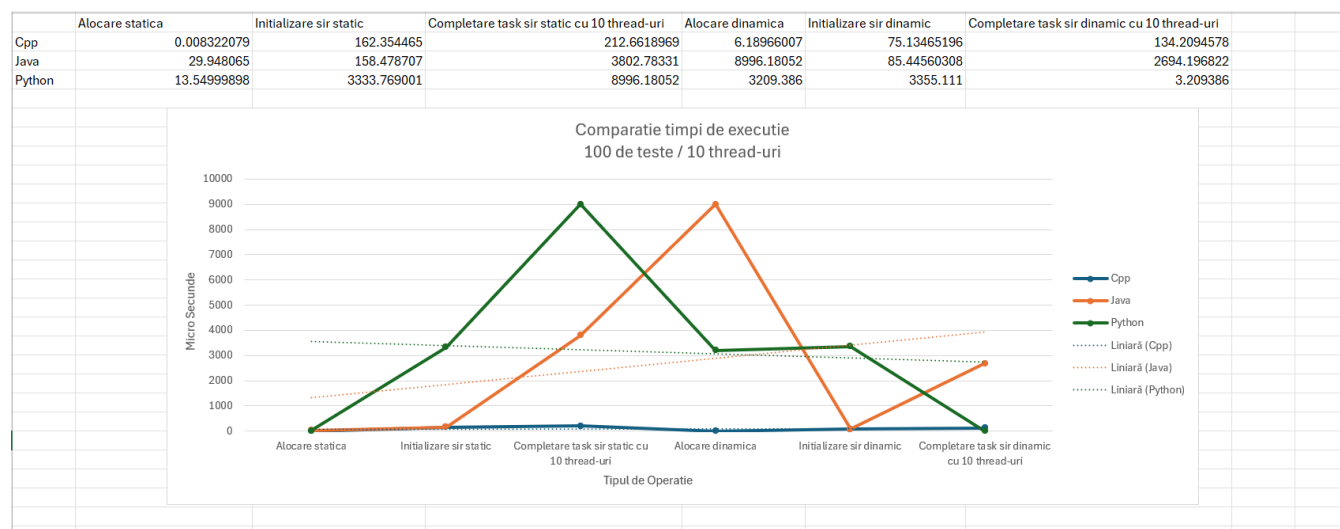
Java Test:



Python test:



Grafic:



7.Concluzii

Proiectul de benchmarking propus a avut ca scop evaluarea performanței în trei limbaje de programare diferite: C++, Java și Python. Prin implementarea unui microbenchmark, am măsurat timpul de execuție pentru operațiile de alocare statică și dinamică a memoriei, precum și pentru prelucrarea unui șir de numere în fiecare limbaj.

Principalele concluzii ale proiectului sunt:

-Alocarea de memorie:

Sintaxa pentru alocarea și inițializarea statică și dinamică a unui șir este similară în C++ și Java. În Python, alocarea dinamică este implicită, iar nu se specifică dimensiunea înainte de utilizare.

Proiectul a evidențiat diferențe semnificative între performanța alocării statice și celei dinamice, oferind o perspectivă asupra eficienței operațiilor de gestionare a memoriei în fiecare limbaj.

-Utilizarea thread-urilor:

Crearea și gestionarea thread-urilor prezintă particularități în fiecare limbaj.

C++ utilizează biblioteca "thread", Java utilizează clasa "Thread", iar Python folosește modulul "threading".

Efectuarea de operații în mod paralel cu ajutorul thread-urilor poate duce la îmbunătățirea performanței, dar impactul poate varia în funcție de limbaj și de implementare.

-Măsurarea timpului de execuție:

Proiectul a implementat funcții specifice fiecărui limbaj pentru a măsura timpul de execuție al diferitelor operații.

Frecvența procesorului a fost estimată folosind instrucțiunile Time Stamp Counter (TSC) în C++ și prin măsurarea timpului între iterații în Java și Python.

-Interfața grafică:

Java și Python au beneficiat de interfețe grafice pentru a facilita utilizatorului introducerea parametrilor și vizualizarea rezultatelor. C++ a avut o abordare de linie de comandă simplă.

-Testare:

Proiectul a fost supus unor teste repetate pentru a evalua consistența și variabilitatea rezultatelor.

S-au observat fluctuații ale frecvenței procesorului în funcție de sarcinile suplimentare ale sistemului de operare.

În concluzie, implementarea și evaluarea proiectului de benchmarking au oferit o perspectivă detaliată asupra performanței în diferite limbaje de programare. Rezultatele și concluziile obținute pot servi drept punct de plecare pentru optimizarea ulterioară a codului în funcție de cerințele specifice ale fiecărui limbaj.

8. Bibliografie

- <https://www.adservio.fr/post/what-is-microbenchmarking#:~:text=A%20microbenchmark%20is%20a%20program,Digital%20Quality>
- <https://www.rdocumentation.org/packages/microbenchmark/versions/1.4.10/topics/microbenchmark>
- <https://devblogs.microsoft.com/pfxteam/quick-microbenchmarks-in-visual-studio-with-code-snippets/>
- <https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/>
- <https://www.geeksforgeeks.org/introduction-to-recursion-data-structure-and-algorithm-tutorials/>
- <https://app.diagrams.net>
- <https://medium.com/tech-interview-collection/python-static-arrays-dynamic-arrays-and-deques-b9344aac80af>
- <https://cplusplus.com/reference/thread/thread/>
- <https://www.baeldung.com/java-measure-elapsed-time>
- <https://pythonhow.com/how/measure-elapsed-time-in-python/>