Mobile Benchmark

Student: Săvianu Bogdan-Casian

20.10.2024

Structura Sistemelor de Calcul

Universitatea Tehnică din Cluj-Napoca

Cuprins

1 Istorie

Termenul de benchmark provine din domeniul armelor militare, unde are aceeași însemnătate ca în tehnologie, aceea de a măsura și a compara performanța diferitelor dispozitive. Cu timpul, armele au devenit tot mai precise și puternice, acuratețea persoanei din spatele trăgaciului (**marksman**) devenind principala liabilitate în testarea armelor și astfel, în testele moderne, arma este prinsă de o bancă (**bench**) și se măsoară dispersia urmelor lăsate după un număr de focuri trase.

2 Introducere

2.1 Context

În domeniul calculatoarelor, un benchmark înseamnă o serie de teste aplicate unei componente hardware sau software cu scopul de a-i măsura performanța și de a o compara cu performanța altor componente similare disponibile pe piață. Aceste teste sunt, de obicei, create pentru a simula sarcini reale sau sintetice, care solicită resursele sistemului pentru a oferi o măsură obiectivă a capabilităților acestuia. De exemplu, un benchmark poate include algoritmi de procesare a datelor, operații matematice complexe sau alte programe sintetice care testează performanța unității centrale de procesare (CPU), unității de procesare grafică (GPU), memoriei RAM sau chiar a sistemului de stocare.

În contextul actual, testarea performantelor dispozitivelor mobile devine esențială, deoarece utilizatorii se așteaptă ca acestea să îndeplinească sarcini diverse, de la navigare pe internet, la jocuri 3D, apeluri video sau streaming video de înaltă calitate. Din acest motiv, benchmark-urile pentru dispozitive mobile sunt special concepute pentru a evalua toate aceste funcții, oferind o imagine de ansamblu asupra capacităților lor în lumea reală.

2.2 Dezavantajele testelor sintetice

Unul dintre dezavantajele majore ale testelor sintetice este că producătorii de componente hardware tind să optimizeze produsele lor pentru aceste benchmark-uri specifice, astfel încât rezultatele obținute să pară impresionante. De aceea, aceste rezultate nu reflectă întotdeauna performanțele reale din utilizarea de zi cu zi, aceste teste sintetice neputând să acopere întotdeauna varietatea largă de scenarii și sarcini pe care un utilizator obișnuit le-ar efectua. Acesta este motivul pentru care trebuie să existe o abordare dinamică și constantă a revizuirii metodologiilor de testare, pentru a le face mai puțin predictibile și mai diverse, evitând optimizările excesive pentru un set limitat de teste.

2.3 Obiectivele proiectului

Obiectivul principal al acestui proiect este dezvoltarea unei aplicații mobile care să ofere utilizatorilor o metodă robustă de testare a performanțelor dispozitivelor mobile, punând accent pe evaluarea puterii de procesare a unității centrale (CPU) și pe testarea capabilităților grafice (GPU) și a eficienței memoriei. Proiectul urmărește să realizeze

o analiză detaliată a performanțelor dispozitivelor, replicând scenarii reale de utilizare pentru a asigura acuratețea rezultatelor.

Primul pas constă în evaluarea performanței CPU-ului prin utilizarea algoritmilor de criptare și a simulărilor de navigare, care evidențiază capacitățile de calcul ale procesorului. Procesul de criptare implică operațiuni complexe din punct de vedere computațional, similare cerințelor aplicațiilor de securitate care rulează în fundal pe majoritatea dispozitivelor mobile. Suplimentar, algoritmul lui Dijkstra va fi aplicat pentru a simula navigarea, acesta fiind un exemplu eficient de algoritm utilizat în scenarii de orientare, precum aplicațiile de hărți. Acest test permite evaluarea capacității dispozitivului de a gestiona rutarea și calculul celor mai eficiente trasee în scenarii reale.

Pentru GPU, se va încerca randarea unor imagini complexe și măsurarea timpului necesar pentru generarea acestora. Acest test urmărește să observe capacitatea procesorului grafic de a gestiona sarcini vizuale intensive, întâlnite în aplicații de jocuri și design, permițând evaluarea performanței grafice moderne a dispozitivului.

În final, evaluarea performanței memoriei se va concentra pe analiza vitezei de accesare și stocare a datelor în timpul testelor anterioare, precum și pe eficiența gestionării memoriei. Aceasta reprezintă un aspect esențial pentru menținerea funcționării fluide a dispozitivelor, chiar și în condiții de utilizare intensivă.

Scopul final este dezvoltarea unei aplicații de benchmarking care să fie nu doar precisă din punct de vedere tehnic, ci și adaptată scenariilor de utilizare reală, asigurând astfel relevanța rezultatelor pentru utilizatori.

2.4 Specificațiile proiectului

Aplicația va fi gândită pentru dispozitivele iOS și implementată nativ în limbajele Swift, respectiv SwiftUI pentru interfața grafică. Apoi pentru a o testa în afara emulatorului din Xcode, o voi porta pe telefonul mobil unde voi rula testele, care vor scrie rezultatele într-un fișier, iar pe urmă interfața grafică va prelua informațiile din fișier și le va afișa într-un mod intuitiv și ușor de interpretat.

3 Studiu bibliografic

Una dintre cele mai populare aplicații de benchmarking de pe piață la ora actuală este Geekbench. Metoda prin care aceștia estimează scorul dat unui dispozitiv este prin compararea cu un procesor i7-12700 dintr-un Dell Precision 3460 care are scorul setat la 2500 și, în funcție de diferențele de performanță obținute, i se oferă un scor adecvat dispozitivului testat.

Testele pe care le face Geekbench se împart în 2 categorii, atât pentru single core, cât și pentru multicore: operații pe numere întregi, și operații pe numere în virgulă flotantă. Pentru a evita deteriorarea performanței din cauza supraîncălzirii, după fiecare test se face o pauză de 2 secunde.

De asemenea, dispozitivul este supus și unor teste care reflectă mai mult activitățile din ziua de zi cu zi, precum:

- **Comprimare de fișiere:** comprimă arhiva cu codul sursă al Ruby 3.1.2, apoi stochează fișierele în memorie criptându-le cu algoritmul SHA-1.
- **Navigare:** folosește algoritmul lui Dijkstra pentru a calcula 24 de rute pe 2 hărți de tipul OpenStreetMaps, una în Waterloo, Ontario, iar cealaltă în Toronto, Ontario.

- **Navigare web:** simulează deschiderea a 8, respectiv 32 pagini de internet populare pentru multicore, unde este testată capacitatea de a randa rext și imagini.
- **Randare de fișiere PDF:** folosind PDFium, renderul de fisiere PDF al lui Chrome, randează hărți din American National Park Service, fișiere care conțin imagini vector mari, linii și text. În acest test sunt folosite 4 fișiere pentru single core, respectiv 16 pentru multicore.
- **Procesarea imaginilor:**
  - **Recunoașterea facială:** Evaluează capacitatea dispozitivului de a identifica fețe în imagini, o funcție esențială în aplicații precum deblocarea facială sau etichetarea fotografiilor.
  - **Segmentarea imaginilor:** Împarte o imagine în regiuni semnificative, fiind utilă în aplicații de editare foto, realitate augmentată și viziune computerizată.
  - **Stil transfer:** Aplică stilul artistic al unei imagini asupra altei imagini, o tehnică populară în aplicațiile de editare foto.
- **Compilarea și interpretarea codului:**
  - **Compilarea aplicațiilor:** Măsoară timpul necesar pentru a compila aplicații scrise în diferite limbaje de programare (C++, Java, Python), evaluând performanța compilatorului și a sistemului de construcție.
  - **Interpretarea scripturilor:** Evaluează viteza de execuție a scripturilor în limbaje interpretate precum Python sau JavaScript, importante pentru dezvoltarea rapidă de aplicații și automatizare.
- **Criptarea datelor:** Măsoară viteza de criptare și decriptare a datelor folosind algoritmi criptografici precum AES, RSA, importantă pentru securitatea datelor.
- **Baze de date:** Evaluează performanța în executarea de interogări SQL pe baze de date SQLite sau MySQL, importante pentru aplicațiile mobile și web.
- **Randare 3D:** Măsoară performanța în redarea grafică 3D, importantă pentru jocuri și aplicații de realitate virtuală.

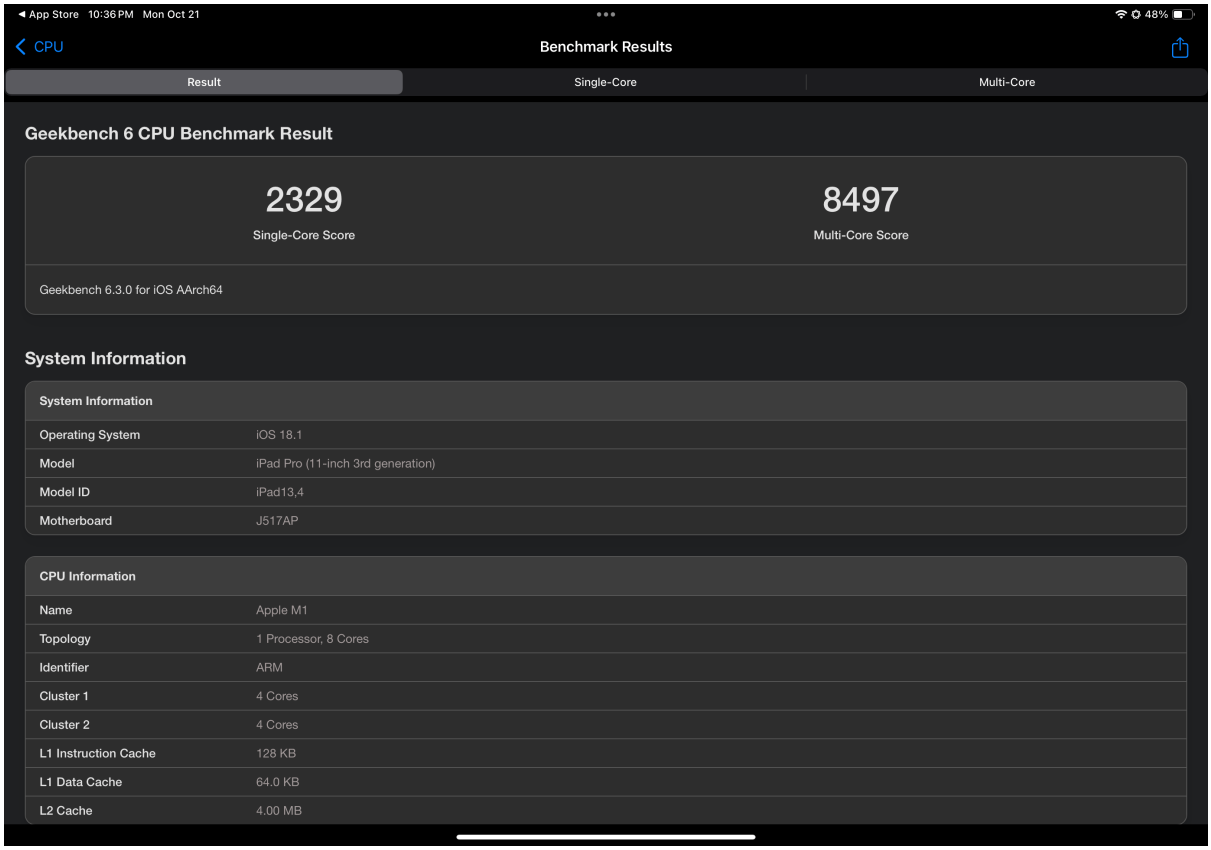

Figure 1: Geekbench 6

4 Analiză

4.1 Analiza algoritmului Argon2

Algoritmul Argon2 este un algoritm de hashing de tip *memory-hard*, proiectat pentru a utiliza intensiv resursele procesorului și ale memoriei. Acesta este recunoscut pentru eficiența sa în gestionarea sarcinilor care necesită securitate ridicată. În general, Argon2 este utilizat pentru hashing-ul parolelor și criptografie, având rolul de a împiedica atacurile cu forță brută prin cerințele sale mari de procesare și memorie. Caracteristicile sale îl fac ideal pentru o aplicație de benchmarking, unde obiectivul este evaluarea performanței procesorului și a capacității de gestionare a memoriei în sarcini intense.

Motivele alegerii Argon2 pentru benchmark-ul aplicației

Algoritmul Argon2 este esențial pentru proiectul de benchmark din mai multe motive:

- **Evaluarea performanțelor CPU**: Argon2 necesită operații de calcul complexe pentru a genera un hash, testând astfel puterea brută a procesorului și viteza de execuție a instrucțiunilor. În această aplicație, folosind Argon2 la parametri diferiți, se pot simula diverse niveluri de dificultate, ceea ce permite obținerea unor măsurători detaliate ale performanței CPU-ului dispozitivului.

- **Evaluarea performanței memoriei**: Fiind un algoritm *memory-hard*, Argon2 solicită o cantitate semnificativă de memorie, generând o sarcină semnificativă asupra dispozitivului și permițând evaluarea eficienței în utilizarea și accesarea memoriei. Astfel, folosind Argon2 în benchmark, pot verifica performanța memoriei dispozitivului, analizând cât de eficient gestionează aceasta accesările rapide și alocările mari de memorie.

- **Configurabilitatea testelor de benchmark**: Argon2 este flexibil prin parametrii săi (*time_cost*, *memory_cost*, *parallelism*), permițând ajustarea cerințelor de procesare și de memorie. Acești parametri permit scalarea complexității testelor de benchmark, simulând sarcini ușoare sau intensive în funcție de capacitățile dispozitivului testat. Astfel, pot fi testate performanțele dispozitivelor atât în scenarii moderate, cât și în scenarii de utilizare intensivă, obținând o imagine detaliată asupra puterii de calcul și a gestionării memoriei.

- **Dificultatea de a falsifica rezultatele**: Utilizarea Argon2 în benchmark permite simularea unor scenarii de securitate întâlnite frecvent în aplicațiile de criptare și de protejare a datelor. Argon2 este dificil de optimizat pentru obținerea unor rezultate false în testele sintetice, ceea ce înseamnă că rezultatele sunt reprezentative pentru performanța reală a dispozitivului în scenarii complexe de hashing și memorare.

Avantajele și provocările utilizării Argon2 în benchmark

Avantaje:

- **Performanță reprezentativă**: Argon2 este mai greu de optimizat excesiv, astfel rezultatele reflectă mai bine performanța reală a dispozitivului.

- **Configurabilitate și adaptabilitate**: Parametrii algoritmului permit simularea unei game largi de scenarii, de la sarcini mai simple la sarcini intensive de hashing.
- **Testare în contexte reale**: Fiind un algoritm de securitate recunoscut, Argon2 reprezintă un test realist al performanței în situații care necesită securitate avansată și procesare intensă.

Dezavantaje:

- **Utilizare intensivă a memoriei**: Caracterul *memory-hard* al Argon2 poate încetini considerabil dispozitivele cu memorie RAM limitată, ceea ce poate afecta validitatea benchmark-ului pentru astfel de dispozitive.
- **Necesitate de optimizare pentru dispozitive low-end**: Dispozitivele mai puțin performante pot întâmpina dificultăți în a rula testele la parametri înalți, ceea ce face testul mai puțin aplicabil pentru ele.

4.2 Operații în virgulă mobilă

Operațiile în virgulă mobilă sunt esențiale în testarea performanței dispozitivelor moderne, acestea fiind utilizate în mod extensiv în calcule științifice, grafică 3D și inteligență artificială. Aceste operații se bazează pe manipularea numerelor reale și sunt deseori folosite în calculele care implică valori precise, cum ar fi simulările fizice, procesarea semnalelor și analiza datelor.

În contextul acestei aplicații de benchmarking, operațiile în virgulă mobilă vor permite evaluarea capacității dispozitivului de a procesa rapid și precis informații complexe. Aceste operații vor testa atât puterea brută a procesorului, cât și abilitatea de a gestiona sarcini numerice complexe fără erori de precizie. Testele vor include calcule pentru funcții trigonometrice, logaritmice și exponențiale, relevante pentru majoritatea aplicațiilor grafice și științifice.

Astfel, utilizarea operațiilor în virgulă mobilă în cadrul aplicației va oferi o măsură obiectivă a capacității procesorului de a efectua calcule complexe, oferind utilizatorului o imagine clară asupra performanței dispozitivului în scenarii de calcul intens.

4.3 Utilizarea algoritmului LZFSE pentru testarea comprimării fișierelor

Algoritmul LZFSE (Lempel-Ziv Finite State Entropy) este o metodă de compresie dezvoltată de Apple, proiectată pentru a oferi un echilibru optim între rata de compresie și viteza de procesare. Este cunoscut pentru performanțele sale superioare în comparație cu alte algoritmi, cum ar fi `zlib`, mai ales pe platformele Apple, datorită integrării eficiente cu arhitecturile hardware specifice.

Motivele pentru alegerea algoritmului LZFSE

- **Eficiența în Compresie**: LZFSE oferă o comprimare rapidă și eficientă, ceea ce îl face ideal pentru teste de performanță unde viteza este esențială. Această caracteristică permite aplicației să simuleze

scenarii de compresie și decomprimare a fișierelor mari fără a compromite performanța generală.

- **Integrarea cu Platformele Apple**: Algoritmul este optimizat pentru a funcționa eficient pe dispozitivele care rulează macOS și iOS, beneficiind de suportul sistemului de operare și de instrucțiunile hardware specifice.
- **Sursă Deschisă și Documentație**: LZFSE este open-source și bine documentat, permițând implementarea și adaptarea sa cu ușurință în diferite aplicații și scenarii de testare.

Implementarea algoritmului LZFSE în aplicație

În cadrul aplicației, LZFSE va fi utilizat pentru a evalua performanța comprimării și decomprimării fișierelor, un aspect esențial în testele de benchmark ale procesorului. Procedura de testare va include următoarele etape:

1. **Pregătirea Datelor**: Selectarea unor seturi de fișiere cu dimensiuni și tipuri variate pentru a simula scenarii de utilizare din viața reală.
2. **Compresia Fișierelor**: Utilizarea algoritmului LZFSE pentru a comprima fiecare fișier și măsurarea timpului necesar pentru această operațiune.
3. **Decompresia Fișierelor**: Măsurarea performanței în procesul de decomprimare pentru a analiza cât de rapid pot fi recuperate fișierele comprimate.

Prin includerea algoritmului LZFSE în testele de performanță, aplicația va putea oferi o evaluare completă a capacităților de compresie și decomprimare ale dispozitivelor testate. Aceasta va contribui la înțelegerea modului în care diferite arhitecturi hardware gestionează sarcinile intensive de compresie, fiind o componentă importantă în evaluarea generală a performanței CPU.

4.4 Testarea Performanței GPU prin Randarea Obiectelor 3D

Aplicația de benchmarking va evalua performanța GPU-ului dispozitivelor mobile prin randarea unor obiecte 3D complexe. Această metodă testează capacitatea GPU-ului de a gestiona eficient sarcinile grafice și de a menține o rată constantă de cadre sub încărcări ridicate. Structura testelor de randare include:

- **Randarea Obiectelor 3D Complexe**: Aplicația va randa modele 3D cu detalii ridicate, pentru a evalua capacitatea GPU-ului de a procesa eficient structuri geometrice complexe.
- **Rezoluții Înalte**: Testarea va include randarea la rezoluții mari, precum 4K, pentru a observa modul în care GPU-ul gestionează memoria video și resursele necesare procesării grafice de înaltă fidelitate.

Prin aceste teste, aplicația va oferi o imagine clară asupra performanței GPU-ului, concentrându-se pe viteza de procesare și eficiența grafică în redarea de obiecte 3D complexe. Acest tip de testare este relevant pentru a evalua capacitățile dispozitivelor în aplicații grafice și jocuri.

4.5 Testarea memoriei

Memoria este o componentă vitală a oricărui dispozitiv mobil, a cărei viteză influențează mult timpii de încărcare a aplicațiilor.

Pentru a testa performanța accesării memoriei, e nevoie să testăm operațiile de scriere și de citire din memoria principală. Pentru acest test se va genera un șir aleator de numere care se vor scrie într-un array stocat pe heap, iar apoi vor fi citite pe rând toate.

4.6 Calcularea scorului final

Pe parcursul rulării fiecărui test, dispozitivului îi va fi alocat un punctaj pentru acel test. La final, scorul total va fi calculat ca fiind media geometrică a punctajelor tuturor testelor.

4.7 Folosirea aplicației

Aplicația va cuprinde 2 use case-uri principale: rularea efectivă a testelor pentru CPU, GPU, memorie sau toate împreună în funcție de selecția utilizatorului, respectiv vizualizarea topurilor rezultatelor.

În momentul în care utilizatorul selectează una dintre modalitățile de testare, aceasta va începe să ruleze, interfața grafică afișând un ecran de încărcare. Când testarea a luat sfârșit, rezultatele sunt scrise într-un fișier, iar interfața va prelua aceste informații și le va prezenta într-un mod ușor de interpretat.

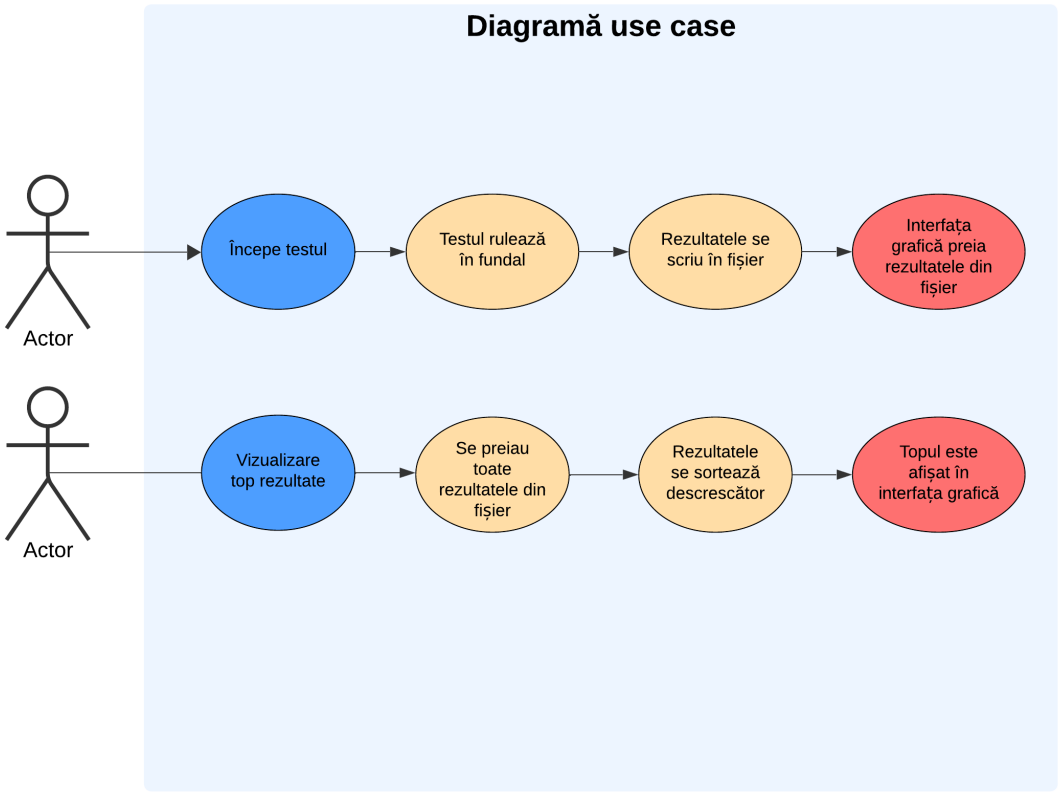

Figure 2: Diagramă use case

Dacă utilizatorul alege să vizualizeze topul rezultatelor, acestea vor fi preluate tot dintr-un fișier care conține tot istoricul testelor și vor fi afișate în ordine descrescătoare într-un tabel.

5 Proiectarea aplicației

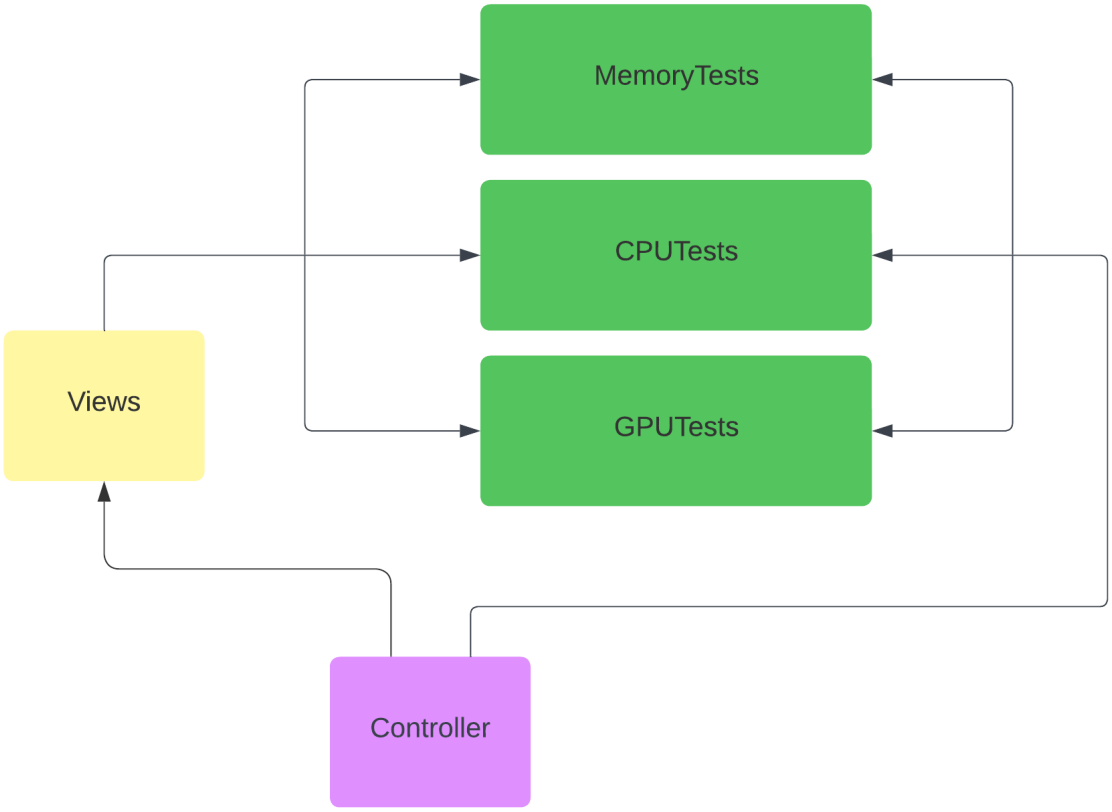

Figure 3: Diagramă de pachete

Pentru a construi această aplicație am ales să urmez ca design pattern Model-View-Controller pentru a separa clar întrebuințele fiecărui modul. În diagrama prezentată în figura 3 am decis să separ modelul în cele 3 componente mari, MemoryTests, CPUTests și GPUTests pentru claritate.

Pachetul CPUTests cuprinde 3 teste folosind algoritmul Argon2 cu parametri variabili de paralelizare, memorie utilizată și număr de iterații pentru a măsura performanța în situații taxante pentru procesor, dar și mai relaxate. De asemenea, tot aici se regăsesc și testele care conțin operații în virgulă mobilă, un test pentru adunări și scăderi, unul pentru înmulțiri și împărțiri, dar și 2 teste echivalente folosind mai multe fire de execuție.

În pachetul GPUTests regăsim un test care randează constant o sferă timp de 5 secunde folosind shadere Metal și calculează câte cadre a reușit să randeze în medie pe secundă.

Testele de memorie se află în pachetul denumit sugestiv MemoryTests. Pentru a putea să aloc un array în memoria heap, am creat o clasă wrapper care conține array-ul respectiv. Apoi acesta este umplut cu valori aleatoare, astfel se testează viteza de scriere a memoriei. Într-un final, acel array este parcurs element cu element și copiat într-un array static pentru a simula citirea datelor din memorie.

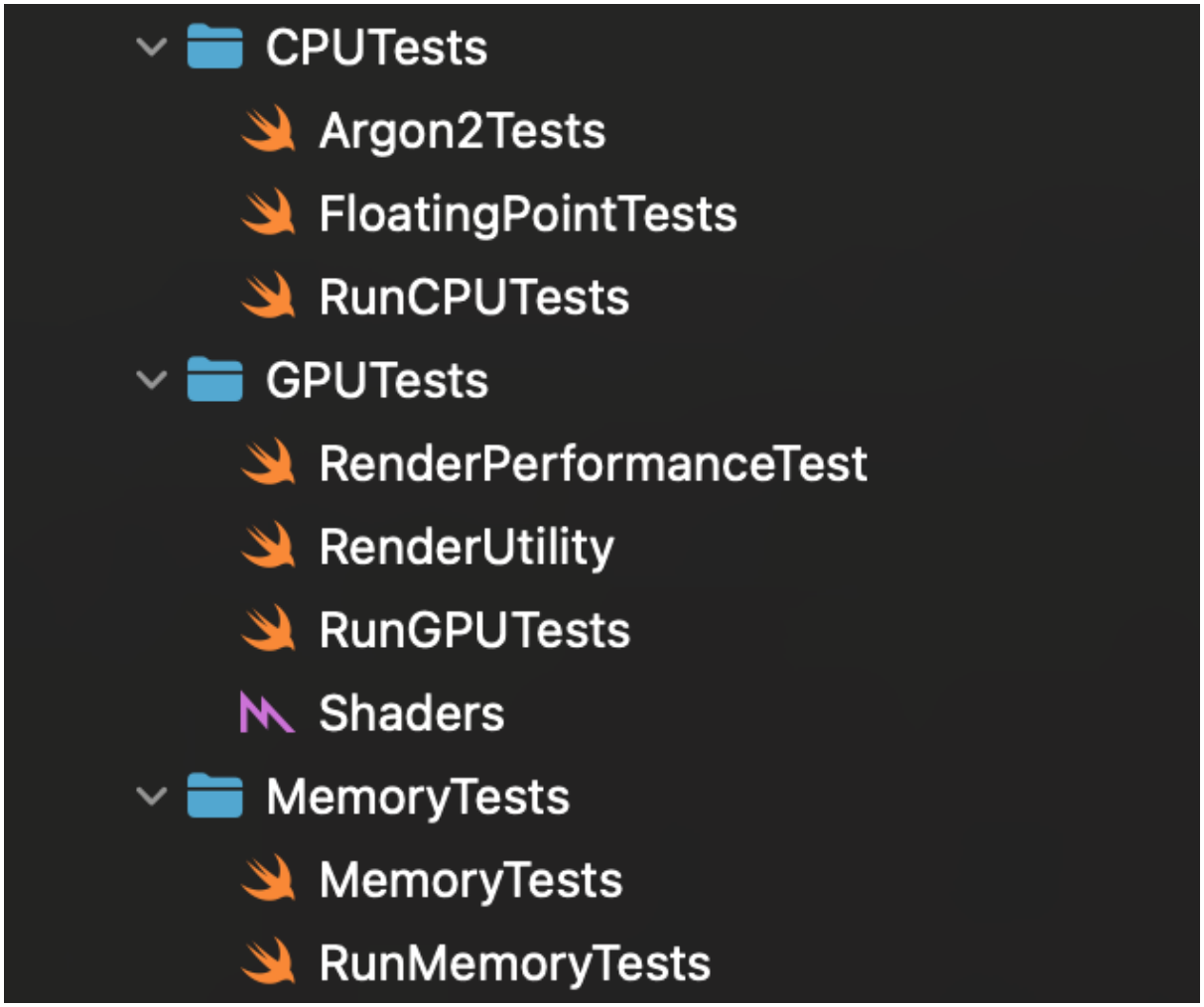

Figure 4: Organizarea testelor

Pe lângă pachetele menționate mai sus, am adăugat un fișier unde creez în mod static structurile utilitare de care e nevoie în diverse părți ale aplicației astfel încât să fie respectat principiul Singleton.

```swift
import Foundation

public struct Singleton {
    static let serialTestQueue = DispatchQueue(label: "com.bogdan.benchmark.serialTestQueue")
    static let argonUtil = Argon2Utility()
    static let fileWriter = FileWriter()
    static let cpuInfo = CPUInfo()
    static let gpuInfo = GPUInfo()
    static let systemInfo = SystemInfo()
    static var memoryWrapper = Wrapper(length: 10_000_000)
}
```

Figure 5: Singleton

Deoarece e nevoie ca testele să fie rulate în fundal pe un fir de execuție diferit de cel principal, astfel încât interfața grafică să nu fie blocată pe parcursul lor, am creat acel DispatchQueue care este folosit pentru a programa testele unul după altul, iar rezultatele sunt preluate la final de către firul de execuție principal pentru a fi afișate în interfață.

6 Bibliografie

[1] Benchmarking - Wikipedia
Accesat: 20.10.2024 - 14:50
https://en.wikipedia.org/wiki/Benchmarking

[2] Misunderstood Mobile Benchmarks Are Hurting The Industry and Consumers
Autor: Patrick Moorhead
Accesat: 20.10.2024 - 20:25
https://www.forbes.com/sites/patrickmoorhead/2015/06/12/misunderstood-or-inappropriate-mobile-benchmarks-are-hurting-the-industry-and-consumers/

[3] Geekbench: How it actually works
Autor: Adam Conway
Accesat: 20.10.2024 - 20:57
https://www.xda-developers.com/geekbench/

[4] What is Argon2?
Autor: Hynek Schlawack
Accesat: 30.10.2024 - 22:14
https://argon2-cffi.readthedocs.io/en/stable/argon2.html

[5] Geekbench 5 CPU workloads
Accesat: 30.10.2024 - 22:29
https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf

[6] Apple Open-Sources its New Compression Algorithm LZFSE
Autor: Sergio De Simone
Accesat: 2.11.2024 - 17:49
https://www.infoq.com/news/2016/07/apple-lzfse-lossless-opensource/

[7] Compressing and decompressing files with stream compression
Accesat: 2.11.2024 - 17:54
https://developer.apple.com/documentation/accelerate/compressing_and_decomp

[8] Gathering system information in Swift with sysctl
Autor: Matt Gallagher Accesat: 8.11.2024 - 00:25
https://www.cocoawithlove.com/blog/2016/03/08/swift-wrapper-for-sysctl.html

[**9**] Multithreading in Swift
Autor: Ajaya Mati Accesat: 20.11.2024 - 19:11
https://ajayamati.medium.com/multi-threading-in-swift-3f4c9c4769c5

[**10**] Metal Tutorial
Accesat: 18.11.2024 - 17:53
https://metaltutorial.com/