МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра дискретного аналізу та інтелектуальних систем
(повна назва кафедри)

Дипломна робота

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ТЕСТУВАННЯ ПРОДУКТИВНОСТІ ПЕРСОНАЛЬНИХ КОМП'ЮТЕРІВ

Виконав: студент групи ПМі-43с
спеціальності
122 «Комп'ютерні науки»
(шифр і назва спеціальності)

_____ Тороній С.С.
(підпис) (прізвище та ініціали)

Керівник _____ проф. Притула М.М.
_____ конс. ас. Прядко О.Я.
(підпис) (прізвище та ініціали)

Рецензент _____
(підпис) (прізвище та ініціали)

2023

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики

Кафедра Дискретного аналізу та інтелектуальних систем

Спеціальність 122 «Комп'ютерні науки»

(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри **Притула М.М.**

"31 "серпня 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНУ У РОБОТУ СТУДЕНТУ

Торонію Сергію Сергійовичу

(прізвище, ім'я, по батькові)

1 .Тема роботи «Розробка програмного забезпечення для тестування продуктивності персональних комп'ютерів»

керівник роботи _____,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **" 13" вересня 2022 року № 15.**

2. Строк подання студентом роботи **13.06.2023р.**

3. Вихідні дані до роботи

Документація по по мові програмування C#. .NET, Dropbox API, GunaFramework, OpenCL, Windows Forms, ColorMania інтернет-ресурси, комп'ютері на базі ОС Windows, середовище розробки Microsoft Visual Studio

4. Зміст дипломної роботи (перелік питань, які потрібно розробити)

Огляд існуючих аналогів , перегляд видів тестувань, створення бенчмарку з різними видами тестувань, дослідження тонкощів алгоритмів тестування компонентів різних видів за продуктивністю систем

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Ілюстрація швидкодії мов програмування, графіки втрати продуктивності з часом під час стрес-тестування

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **31 серпня 2022 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Аналіз та постановка задачі	10.10.2022	
2.	Вивчення літератури по темі	15.11.2022	
3.	Огляд існуючих аналогів додатку	12.12.2022	
4.	Аналіз видів тестування компонентів системи	09.01.2023	
5.	Проектування та вдосконалення бенчмарку	19.02.2023	
6.	Розробка додаткових можливостей застосунку	30.03.2023	
7.	Удосконалення існуючого функціоналу	12.04.2023	
8.	Оформлення дипломної роботи	29.05.2023	

Студент _____ Тороній С.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Прядко О.Я.
(підпис) (прізвище та ініціали)

Реферат

Робота складається із вступу, трьох розділів, висновків та списку використаної літератури. Обсяг основної частини дипломної роботи: 46 сторінок тексту, 2 таблиць та 26 рисунків. Список використаних джерел складається з 11 найменувань.

Мета роботи полягає у створенні одного з таких продуктів для тестування продуктивності персональних комп’ютерів швидким та простим способом, який би додатково містив в собі онлайн рейтинг для порівняння результатів з результатами будь-кого іншого, дослідити значну частину цієї галузі. Основною ціллю є створення та розробка безкоштовного, швидкого бенчмарку з швидким та простим тестуванням продуктивності ПК на базі ОС Windows. На сьогоднішній день бенчмарки є популярними здебільшого в тих сферах, користувацькому досвіду яких притаманна пряма залежність від швидкодії ПК.

Новизна роботи полягає у важливому значенні бенчмарків в розробці та тестуванні програмного забезпечення, зокрема в області комп'ютерної графіки та інших високопродуктивних додатків. Розробники програмного забезпечення використовують бенчмарки для оцінки ефективності та оптимізації своїх продуктів на різних конфігураціях обладнання.

ЗМІСТ

ВСТУП

Завдяки появі комп'ютерної техніки, життя людства зазнало кардинальних змін. Процес керування даними значно спростився, починаючи з часів виникнення друкарських машинок. Комп'ютери почали виконувати все більше розрахунків, що пришвидшило технічний прогрес. У кінці XX століття деякі сфери життєдіяльності, де раніше людина відігравала основну роль, значно зменшилися.

На сьогоднішній день, зі зростанням продуктивності комп'ютерів, збільшилася складність та кількість задач, які необхідно виконувати. Проте не всі мають можливість мати найновіші технологічні досягнення. Це призводить до проблеми недостатньої продуктивності для виконання поставлених задач.

Щоб з'ясувати взаємозв'язок між приростом та потребами в швидкості персональних комп'ютерів у сфері інформаційних технологій, були розроблені програми для тестування продуктивності за високого навантаження - бенчмарки. Перед тим як придбати програмне забезпечення, багато людей звертають увагу на мінімальні та рекомендовані системні вимоги, які встановлені в результаті тестування бенчмарками. Кожен, хто працює в галузі інформаційних технологій, будь то програміст, музикант, геймер чи відеомонтажер, цікавиться системою, що задовольняє вимоги необхідних продуктів.

Метою цієї роботи є створення продукту для швидкого та простого тестування продуктивності персональних комп'ютерів, який також містить онлайн рейтинг для порівняння результатів з іншими. Таким чином, буде досліджено значну частину цієї галузі.

Програмне забезпечення має наступні можливості:

- Тестування швидкості виконання арифметичних, стрічкових та інших операцій ЦП.

- Тестування швидкості виконання операцій, пов'язаних з відеокартою (ГПУ).
- Тестування температурних показників для виявлення троттлингу під великими навантаженнями.
- Тестування швидкості інтернет-з’єднання.
- Тестування швидкості запису та читання диску.
- Реалізація онлайн рейтингу продуктивності ПК.
- Реалізація сучасного інтерфейсу програми.
- Автоматичне оновлення програми.

Отже, головною метою цієї роботи є створення безкоштовного та швидкого бенчмарку для простого тестування продуктивності ПК під управлінням операційної системи Windows.

1. Основні поняття про бенчмарки

Бенчмарк є інструментом для оцінки продуктивності електронних пристроїв, зокрема комп'ютерів. Використовуючи програму комп'ютерного порівняльного аналізу, проводяться серії тестів на ПК для вимірювання його продуктивності.

Бенчмарки надають оцінку, яку можна використовувати для порівняння ПК систем. Вищий бал вказує на кращу продуктивність. За допомогою бенчмарків легше порівнювати контрольні показники, ніж складні технічні характеристики. Вони допомагають визначити швидкодію окремих компонентів і системи в цілому. Деякі бенчмарки також містять функціонал стрес-тестування, який виявляє можливий тротлінг ПК. Бенчмарки часто надають інформацію про компоненти системи, які тестуються.

Крім того, бенчмарки можуть використовуватися для тестування продуктивності програмного забезпечення, такого як відеоігри або програми для обробки фото та відео. Ці тести допомагають визначити, як програмне забезпечення працює на конкретній системі і порівняти продуктивність різних програм на однакових комп'ютерах. Деякі бенчмарки також проводять тестування енергоефективності, щоб визначити ефективність роботи комп'ютера в різних режимах. Бенчмарки є корисним інструментом для оцінки продуктивності комп'ютерної системи і можуть дати підставу для рішення про оновлення або заміну компонентів.

1.1 Види та способи тестувань

Зазвичай бенчмарки поділяються на мобільні та настільні (призначені для ПК), але існують також кросплатформені види тестувань. Вони надають оцінку або бал, який розраховується на основі відношення між складністю та тривалістю виконання системи, компонентів, операцій

та алгоритмів. У ПК основними компонентами для тестування є центральний процесор (CPU) та відеокарта (GPU), які безпосередньо тестуються на складні та тривалі за часом алгоритми. Вимірюючи швидкість окремо CPU та GPU, можна оцінити їхню продуктивність в ПК. Деякі види тестувань, наприклад стрес-тестування, навантажують систему повністю шляхом одночасного виконання різних складних операцій на всіх компонентах.

Стрес-тестування дозволяє не лише максимально навантажити комп'ютер, але й виявити тротлінг - зниження частоти компонентів, яке призводить до загального зменшення продуктивності. Під час стрес-тестування також вимірюються температури, що є одним із підвидів бенчмарків. Недостатня ефективність систем охолодження може призвести до підвищення температур компонентів і тротлінгу. Бенчмарки можуть вказати на такі проблеми та сприяти вирішенню їх для виробників комп'ютерів або ноутбуків.

Крім CPU та GPU, бенчмарки також можуть тестувати інші компоненти, наприклад оперативну пам'ять, жорсткий диск або SSD, а також мережеві інтерфейси. Тестування оперативної пам'яті допомагає виявити її повну потужність, а тестування дискового простору - швидкість передачі даних.

Бенчмарки можуть бути використані для порівняння комп'ютерних систем або оцінки ефективності оновлень. Проведення бенчмарків до і після оновлення може показати покращення продуктивності системи після оновлення.

Важливо мати на увазі, що бенчмарки надають орієнтовну оцінку продуктивності комп'ютера, і їхні результати можуть відрізнятися залежно від контексту використання. Також варто звернути увагу на якість виконання бенчмарку, його налаштування та тестовий сценарій, оскільки ці фактори можуть вплинути на точність результатів.

1.2 Сучасний стан та актуальність

На сьогоднішній день бенчмарки є популярними у різних сферах, особливо в тих, де користувацький досвід прямо залежить від швидкодії ПК. Розглянемо деякі з них:

1. Сфера геймінгу, обробки відео та фото: Геймери та фахівці з обробки медіа звертають увагу на результати бенчмарків, оскільки швидкість роботи системи безпосередньо впливає на їхнє задоволення. Вони особливо приділяють увагу результатам тестування графічних процесорів, оскільки саме вони виявляють свій потенціал у відеоіграх та графічних застосунках. Також фахівці, які працюють з обробкою відео та фото, цікавляться продуктивністю відеокарт, оскільки вона безпосередньо впливає на швидкість редагування та створення графічних елементів.

2. Сфера IT: Більшість фахівців, пов'язаних з IT, так чи інакше залежать від продуктивності ПК, з яким вони працюють. Швидкодія системи має прямий вплив на їхню роботу та ефективність.

3. Сфера криптомайнингу: Криптомайнери залежать від продуктивності графічних процесорів, оскільки саме вони використовуються для майнінгу криптовалют. Результати бенчмарків допомагають їм обрати оптимальні комплектуючі для максимізації прибутку.

Бенчмарки також мають важливе значення в розробці та тестуванні програмного забезпечення. Розробники використовують бенчмарки для оцінки ефективності та оптимізації своїх продуктів на різних конфігураціях обладнання. Вони також допомагають порівняти різні архітектури та технології, що використовуються в галузі комп'ютерної техніки та електроніки.

Отже, бенчмарки є важливим інструментом для різних галузей технологій та для кожного користувача, який бажає отримати максимальну продуктивність свого ПК. Вони дозволяють відстежувати тенденції в

розвитку технологій та порівнювати різні системи на ринку, надаючи об'єктивну оцінку їх продуктивності.

2. Практична реалізація застосунку

Torobench - це розроблений програмний продукт, який містить в собі онлайн рейтинг для порівняння результатів користувачів. Головною метою Torobench є створення безкоштовного та швидкого бенчмарку для тестування продуктивності ПК під управлінням операційної системи Windows.

Основні технології та бібліотеки, які використовуються в Torobench:

1. Windows Forms та .NET: Torobench реалізований з використанням Windows Forms та мови програмування C#. Ці технології обрані через їхню підтримку операційної системи Windows.
2. Dropbox API: Для зберігання даних рейтингу користувачів використовується хмарне сховище Dropbox, доступ до якого здійснюється за допомогою Dropbox API.
3. Guna Framework: Для створення інтерфейсу програми використовується Guna Framework. Це обрано через обмеження стандартних інструментів Windows Forms щодо дизайну.
4. AutoUpdater.NET: Для оновлення застосунку використовується відкрита бібліотека AutoUpdater.NET. Це дозволяє автоматично оновлювати програму і підтримувати її в актуальному стані.
5. OpenCL: Алгоритми для графічного процесора реалізовані на низькорівневій мові програмування OpenCL. Це робить алгоритми незалежними від виробників графічних процесорів.

Torobench тестує продуктивність основних комплектуючих будь-якого ПК. Для цього використовуються алгоритми пошуку простих чисел, арифметичні та тригонометричні операції, піднесення до степеня, визначення швидкості хешування та інші паралельні алгоритми, такі як множення великих матриць. Також проводяться операції з великою

кількістю знаків після коми та операції із стрічковими даними, наприклад, конкатенація.

Програма також надає можливість запуску стрес-тестування ПК з моніторингом температур, навантажень та показників троттлингу в реальному часі. Результати тестів користувачів автоматично додаються до рейтингу, який містить найкращі результати тестувань та інформацію про ПК з можливістю пошуку та сортування записів.

Також в Torobench є функція оновлення застосунку, яка дозволяє пропустити версії додатку або нагадати про оновлення. Програма має можливість кастомізації вигляду, зокрема вибір світлої, темної або системної теми.

Torobench є багатопотоковим застосунком, що дозволяє проводити тестування без блокування інтерфейсу для користувача.

2.1 Технології .NET Framework та Windows Forms

Програма була реалізована з використанням .NET Framework 4.6.1, який є останньою версією на момент створення застосунку. Для розробки інтерфейсу користувача використовувався Windows Forms, що надає зручні можливості для налаштування вигляду програми. Однак для реалізації дизайну інтерфейсу був використаний Guna Framework, що дозволило освіжити зовнішній вигляд програми і надати йому сучасний стиль. Guna Framework надає розширений набір графічних елементів та стилізацію, що допомагає створювати привабливі та сучасні користувацькі інтерфейси.

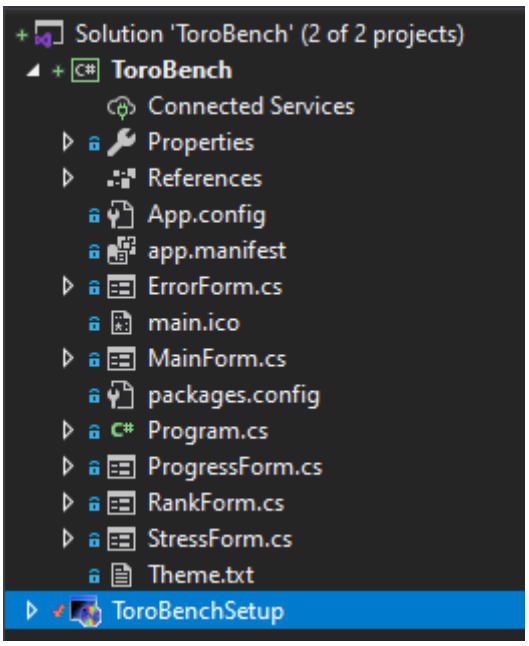

Рисунок 2.1 Коренева папка програми

В папці, що є кореневою для застосунку (див. Рис. 2.1), було створено кілька класів-форм, які реалізують різні функціональні можливості програми. Крім самого проекту застосунку, був створений окремий проект, спрямований на встановлення бенчмарку.

2.2 Тестування CPU

Тестування продуктивності центрального процесора (CPU) ПК можна розділити на дві основні частини: перше - це вимірювання швидкодії одного логічного ядра CPU (Single-Core), а друге - вимірювання швидкодії всіх логічних ядер CPU (Multi-Core). Це пояснюється тим, що не всі типи програмного забезпечення, що виконуються на ПК, підтримують багатопотоковість, оскільки деякі операції не можуть бути розпаралелені. Під час тестування процесор виконує різні операції, і час, необхідний для їх виконання, пропорційний до визначення балу результату тестування. Цей бал обчислюється шляхом відношення виконавчого часу до певної константи. Чим швидше система виконує завдання, тим вище буде бал.

У цьому коді, що входить до програми, проводиться тестування продуктивності обчислень на процесорі (CPU). Після ініціалізації списків з даними, код використовує два підходи до обчислень: послідовне виконання операцій на одному ядрі процесора та паралельне виконання операцій на кількох ядрах процесора за допомогою бібліотеки Parallel.

У цьому коді виконуються наступні дії:

1. Встановлюється маска процесорного афініту, щоб гарантувати виконання операцій на одному ядрі процесора.

2. Обчислення виконуються на числах зі списків floatList та intList, які містять по 2000 елементів. Для кожного елементу списку проводиться послідовність арифметичних операцій та викликів математичних функцій.

3. Для списку stringList виконується додавання рядків та заміна символів.

4. Код використовує клас Stopwatch для вимірювання часу виконання операцій.

5. Результати продуктивності різних методів вимірюються та зберігаються в об'єкті rS.

6. Результати виводяться на екран у форматі кількості операцій на мільйон мілісекунд виконання.

Загальна мета цього коду полягає в тому, щоб дослідити, як різні методи обчислень впливають на продуктивність процесора та порівняти їх результати. Код використовує бібліотеку Parallel для паралельного виконання операцій на кількох ядрах процесора та клас Stopwatch для точного вимірювання часу виконання операцій.

2.2.1 Тестування Single-Core

Для проведення тестування швидкодії процесора в однопотоковому режимі, програма встановлюється в операційну систему з профілем, що обмежує його виконання до одного з ядер ЦП. Це дозволяє центральному процесору виконувати будь-які операції на вибраному ядрі і виміряти його швидкодію.

```
long AffinityMask = (long)Proc.ProcessorAffinity;
AffinityMask &= 0x0001;
Proc.ProcessorAffinity = (IntPtr)AffinityMask;
```

Рисунок 2.2.1 Встановлення процесу в однопотоковий режим

За допомогою класу System.Diagnostics.Process можна налаштувати "маску афінітету", яка визначає порядковий номер логічного ядра ЦП, на якому буде виконуватися наступний код програми (див. Рис. 2.2.1). З відкритих джерел [7] встановлено, що операції, які вимагають значних ресурсів ЦП, включають піднесення до степеня великих чисел, обчислення тригонометричних функцій та операції з рядками, такі як конкатенація. Ці операції можуть повністю використати потенціал центрального процесора, завантажуючи його на 100%. Вимірювання використання процесора оцінюється різними способами, наприклад, шляхом вимірювання часу виконання процесу протягом однієї секунди. Значення завантаження визначається за допомогою бібліотеки OpenHardwareMonitor, яка також використовується в програмі для моніторингу сенсорних показників системи, таких як CPUID HWMonitor, який є популярним серед розробників у галузі створення бенчмарків. Під час розробки було прийнято рішення включити тестування інших класичних операцій, які також вимагають значних ресурсів ЦП, включаючи арифметичні операції з числами з великою кількістю знаків після коми, пошук простих чисел та сортування великих списків даних.

2.2.2 Тестування Multi-Core

Для проведення тестування швидкодії центрального процесора в багатопотоковому режимі, значення профілю процесу програми в операційній системі встановлюється на відповідну кількість доступних логічних ядер.

```
Proc.ProcessorAffinity = (IntPtr)((1 << Environment.ProcessorCount) - 1);
using (Process p = Process.GetCurrentProcess())
    p.PriorityClass = ProcessPriorityClass.Normal;
```

Рисунок 2.2.2 Встановлення процесу в багатопотоковий режим

Для встановлення значення використовується клас System.Environment, що надає можливість отримати кількість доступних логічних ядер процесора (див. Рис. 2.2.2). Крім вищезгаданих операцій, описаних у пункті 2.2.1, для тестування продуктивності ЦП в режимі Multi-Core також були використані різні алгоритми для паралельного програмування за допомогою класу System.Threading.Tasks.Parallel. Цей клас містить інструменти для створення паралельних циклів, що дозволяють ефективно використовувати потенціал багатьох ядер процесора.

2.3 Тестування GPU. OpenCL та CUDAfy.NET

Для оцінки продуктивності графічного процесора (GPU) в технологіях .NET Framework вбудованих інструментів не існує. Тому були використані сторонні інструменти, зокрема:

1. OpenCL: Це низькорівнева мова програмування, яка дозволяє прямий доступ до графічного ядра відеокарти та виконання різних алгоритмів та операцій на відеокарті.

2. CUDAfy.NET: Це стороння бібліотека, яка також дозволяє використовувати GPU для обчислень та виконання завдань. Вона пропонує зручні інструменти для розробників, оскільки використовує мову програмування C#.

Ці інструменти є кросплатформними, що означає, що програма може тестувати продуктивність будь-якої відеокарти незалежно від її виробника.

Алгоритми для тестування відеокарт включають всі типи тестів для ЦП, які були описані в пунктах 2.2.1 та 2.2.2. Крім того, під час тестування відеокарти виконуються різні паралельні операції, такі як множення та ділення матриць. Виявлено, що саме ці операції найбільш навантажують відеокарту на 100%, що є важливим завданням для таких застосунків.

Для виконання завдань була використана бібліотека OpenCL, яка містить кілька класів для GPU-програмування:

- OpenCL: для безпосереднього застосування GPU.

- MultiCL: для виконання тестування з використанням усіх ресурсів системи.

- EasyCL: спрощений варіант OpenCL.

Оцінка продуктивності відбувається таким же способом, як і при тестуванні ЦП, який описаний у пункті 2.2. Однак, константа була змінена для отримання більш точних результатів, оскільки відеокарти значно швидше виконують завдання.

У наведеному коді виконуються обчислення на відеокарті (GPU) з використанням технології OpenCL. Код містить функцію, яка містить ядро обчислення ("kernel"), яке приймає два масиви дійсних та цілих чисел, а також їх розміри. Внутрішність цього ядра складається з циклу, який обходить елементи масивів і застосовує до них арифметичні операції.

Далі налаштовуються параметри для використання OpenCL, передаються масиви та їх розміри в ядро. Потім створюється новий потік, в якому виконується цикл обчислення ядра за допомогою методу Execute

класу OpenCL. Цикл виконується протягом певної кількості ітерацій, і стан виконання операцій регулярно оновлюється за допомогою ProgressChangedEventHandler.

Якщо виконання процесу було перервано ззовні, то виконання коду припиняється. Після завершення обчислень на відеокарті обчислюється їх продуктивність, і результат записується у змінну rS.GpuScore.

2.4 Стрес-тестування системи

Для реалізації функціоналу стрес-тестування, який включає тестування одночасно центрального процесора (CPU) та відеокарти (GPU), а також моніторинг показників навантаження в реальному часі, можна використовувати асинхронні функції для розділення роботи на окремі потоки.

Наведений код містить метод "WhileCPU", який представляє асинхронну операцію, що виконується в окремому потоці для операцій на CPU. Цей метод містить безкінечний цикл `while`, який містить паралельний код з вкладеними циклами `Parallel.ForEach` та `Parallel.For`.

У першому циклі `Parallel.ForEach` виконуються операції паралельно для кожного елемента списку L. В коді не вказано, які саме операції виконуються з кожним елементом, але можна припустити, що це додаткові дії, які виконуються в кожному потоці.

Другий цикл `Parallel.For` також виконується паралельно та повторюється 1000000 разів для кожного елемента списку L. В цьому циклі також відсутній конкретний код операцій, які виконуються, але можна припустити, що це операції, які потребують значних обчислювальних ресурсів і виконуються в кожному потоці.

У коді зазначається наявність змінної `stop`, яка може використовуватись для зупинки виконання циклу. Крім того, метод

повертає цілочисельне значення 0, що вказує на успішне завершення операції.

Такий підхід дозволяє завантажити основні компоненти системи на повну потужність та виявляти наявність троттлінгу шляхом моніторингу показників навантаження в реальному часі.

2.5 Тестування швидкості інтернету

У цьому розділі буде описано тестування швидкості інтернет-з'єднання, щоб визначити наявну пропускну здатність мережі на комп’ютерах та показати певні результати користувачу після тестування. Для цього буде використано спеціальний інструмент, який відправляє запити в обидві сторони і повертає результати, що містять інформацію про швидкість завантаження та відвантаження даних. Крім того відбувається безпосередньо визначення ще таких важливих параметрів швидкості мережі як jitter та ping.

Після тестування остаточні результати показуються користувачу програми, а також під час виконання самого тестування користувач має змогу бачити прогрес етапу завантаження та відвантаження даних.
Тестування швидкості інтернету допоможе зробити висновки про ефективність мережі та необхідність проведення її модернізації.

2.5.1 Основні поняття про параметри тестування інтернету

Швидкість завантаження - це швидкість з якою через мережу Інтернет приходять дані на комп’ютер користувача. Ця швидкість є значущою для повсякденного користування інтернетом, адже вона відіграє значну роль користування ним. Висока швидкість завантажень зазвичай супроводжується виском комфортом користування в інтернеті. Чим швидша швидкість завантаження тим швидше файли в інтернеті досягають користувача. Швидкість завантаження - це кількість даних, які

можуть бути передані з сервера в пристрій користувача за одиницю часу, зазвичай вимірювано в мегабітах на секунду (Мбіт/с). Швидкість завантаження є одним з ключових параметрів для визначення якості підключення до Інтернету. Чим вища швидкість завантаження, тим швидше веб-сторінки завантажуються, аудіо та відео відтворюються без буферизації, і інші онлайн-дії виконуються швидше. Однак, швидкість завантаження може бути обмежена різними факторами, такими як швидкість підключення до Інтернету, якість мережі, наявність вірусів або шкідливих програм, що впливають на пропускну здатність мережі, інтенсивність використання мережі в даному регіоні, і т.д.

Швидкість відвантаження - це швидкість з якою через мережу Інтернет проходять дані та надсилаються від користувача у власне мережу. Поняття такої швидкості є протилежною до швидкості завантаження. Чим швидша швидкість відвантаження даних в мережі, тим швидше користувач може ділитись своїми даними з іншими користувачами. Швидкість відвантаження відноситься до швидкості передачі даних з локального пристрою (наприклад, комп'ютера або телефону) до віддаленого сервера. Швидкість відвантаження є важливою метрикою як для домашніх, так і для корпоративних користувачів, оскільки вона визначає, наскільки ефективно можна завантажувати файли, докладати роботу в хмару, відправляти електронну пошту з великими вкладеннями, стрімінгові відео та інші важливі функції, які потребують передачі даних у мережу Інтернет. Швидкість відвантаження може бути обмежена провайдером Інтернету, обладнанням та іншими факторами, тому важливо вимірювати швидкість відвантаження, щоб зрозуміти, наскільки добре функціонує ваша мережа.

Ping - це інтернет-протокол, що використовується для вимірювання часу, необхідного для передачі пакету даних з одного пристрою до іншого через мережу.

Пінг може бути використаний для визначення стану доступності веб-сайтів або серверів, а також для визначення якості з'єднання з Інтернетом. Пінг вимірюється у мілісекундах (мс), і чим менше значення пінгу, тим краще якість з'єднання. Велике значення пінгу може вказувати на проблеми з мережею, такі як перевантаження або зниження пропускної здатності.

Jitter - це вимірювання варіацій (змінності) часу відправки і отримання пакетів даних в мережі. Іншими словами, це може бути різниця між часом відправки і часом отримання пакету. Відсутність або мінімізація Jitter дуже важлива для якісного відеозв'язку та онлайн-ігор, де кожна мілісекунда може вирішувати результат матчу. Jitter може бути виміряно в мілісекундах (ms) або в процентах (%) від часу затримки пінгу. Зазвичай чим менше значення Jitter, тим краще якість підключення.

Для тестування показників була створена окрема форма в застосунку для вимірювання цих показників, ця форма містить дані про сам комплектуючий пристрій (Wi-Fi модуль, Ethernet і т.д.), крім того на формі користувачу доступні компоненти, які відповідають за швидкість завантаження та відвантаження даних, а також текстові дані про jitter та ping.

2.5.2 Тестування швидкості завантаження

Швидкість завантаження тестується таким чином, що програма намагається з віддаленого серверу в інтернеті завантажити файл розміром 100 мегабайт. Цей розмір було вибрано для оптимізованого тестування, адже в різних користувачів швидкість завантаження може дуже сильно відрізнятись, тому, вибравши оптимальний на мою думку розмір файлу на сьогоднішній день, це дозволить збільшити комфорт тестування для користувача. Програмна реалізація тестування побудована за схемою завантаження файлу за сталий проміжок часу, але якщо швидкість у користувача значно більша за розмір файлу, то сам тест виконається

швидше за цей проміжок і користувач побачить результати швидше, ніж користувач у якого швидкість інтернету дуже низька. Для отримання даних про назву та модель девайсу, який відповідає за інтернет-з’єднання на комп’ютері користувача написана окрема функція в коді програми, яка використовуючи такі класи як NetworkInterface та GatewayIPAddressInformation дозволяє отримати модель саме того девайсу, який зараз має доступ в мережу Інтернет. Тобто програма проходячи по всіх доступних адаптерах в системі, пробує відправляти запити на віддалений сервер і таким чином є можливість тестування саме того девайсу, який зараз виконує функцію підключення до інтернету. Код отримує назву пристрою, з якого він виконується.

Для отримання назви пристрою використовується .NET-клас NetworkInterface, який надає доступ до мережевих інтерфейсів. Метод GetAllNetworkInterfaces повертає масив інтерфейсів, які доступні на даному пристрої.

Цикл foreach обходить кожен інтерфейс та перевіряє, чи має він статус OperationalStatus.Up та чи не є типом мережевого інтерфейсу NetworkInterfaceType.Loopback. Якщо ці умови виконуються, то отримуються властивості IP-інтерфейсу методом GetIPProperties().

Далі перевіряється наявність хоча б однієї шлюзової адреси (GatewayIPAddressInformation) за допомогою властивості GatewayAddresses. Якщо така адреса є, то перевіряється, чи належить вона до IPv4-адресної сім'ї. Якщо ця умова виконується, то назва пристрою зберігається в змінній deviceName та цикл foreach переривається за допомогою оператора break.

Нарешті, метод повертає назву пристрою, яку він зберіг у змінній deviceName.

Для того щоб протестувати швидкість завантаження було створено асинхронну функцію, яка дозволяє не блокувати інтерфейс програми і

виконувати тестування незалежно від нього. В цій функції власне ініціалізується таймер на певний проміжок часу тестування, щоб користувач із високою та низькою швидкістю проводили тестування за один і той самий проміжок часу. Це було впроваджено для зручності користування програмою. Також крім основного тестування швидкості, після проведення якого за певними формулами визначаються остаточні дані тестування швидкості завантаження, користувачу через подію DownloadProgressChanged показується прогрес тестування, а також швидкість завантаження в даний момент часу. Це дозволяє отримати користувачу ясніше розуміння про стабільність його з'єднання з інтернетом, а також є інформативним, тому що користувач явно бачить, що зараз робить програма.

Якщо швидкість інтернету є повільною або дуже швидкою, то одиниці вимірювання змінюються автоматично для наглядного інформування користувача. Крім того на інтерфейсі програми передбачено спеціальні компоненти, які сповіщають візуально про прогрес виконання тестування. Після того як відбулось тестування швидкості завантаження даних відразу починається тестування швидкості відвантаження.

Програмна реалізація цього типу тестування представляє собою метод C#, який використовує клас WebClient для вимірювання швидкості завантаження даних з інтернету. Метод повертає значення подвійного типу, яке відображає швидкість завантаження даних у мегабітах за секунду (Mbps).

Код використовує таймер, щоб виміряти час завантаження даних і прогрес-бар для відображення прогресу завантаження. Якщо завантаження займає більше 10 секунд, то завантаження переривається і відображається значення швидкості, яка була досягнута до того моменту.

Цей код також містить розрахунки для відображення швидкості завантаження відповідним чином в залежності від її значення. Наприклад,

якщо швидкість менше 1 Мбіт/с, то вона відображається в кілобітах на секунду (Kbps), якщо вона більше 1 Мбіт/с, але менше 1 Гбіт/с, то вона відображається в мегабітах на секунду (Mbps), а якщо вона більше 1 Гбіт/с, то вона відображається в гігабітах на секунду (Gbps).

Цей код може бути корисним для вимірювання швидкості завантаження даних в різних програмах або для порівняння швидкості завантаження даних на різних комп'ютерах або в різних мережах.

Крім того, якщо швидкість завантаження дуже повільна або відсутня, то метод встановлює значення downloadSpeed на -1. Це дозволяє виявляти проблеми зі з'єднанням, такі як втрату мережевого з'єднання або низьку швидкість інтернету.

Загалом, цей код є досить ефективним тестом швидкості завантаження даних, який може бути використаний для вимірювання швидкості інтернету та виявлення проблем з мережею. Однак, важливо пам'ятати, що швидкість залежить від багатьох факторів, таких як відстань до сервера, тип мережі та трафік в мережі в цей час, тому результати можуть відрізнятися в різні моменти часу.

2.5.3 Тестування швидкості відвантаження

Для тестування швидкості відвантаження виконується подібна форма тестування така, як для швидкості завантаження. Основна різниця полягає в тому, що файл, який буде відвантажуватись на віддалений сервер, створюється локально на стороні користувача з фіксованим розміром для всіх користувачів.

Для вимірювання швидкості програма використовує клієнт WebClient, який надсилає випадкові дані розміром 10 МБ на веб-сервер.

Код програми складається з двох методів: GetUploadSpeed() і UploadDataAsync(). Метод GetUploadSpeed() відповідає за визначення

швидкості відвантаження даних, а метод UploadDataAsync() відповідає за відвантаження даних на веб-сервер та відстеження прогресу.

У методі GetUploadSpeed() спочатку створюється новий екземпляр WebClient. Потім генерується випадковий буфер розміром 10 МБ, і таймер встановлюється на 10 секунд. Також встановлюється мінімальна і максимальна швидкість анімації для відображення прогресу завантаження.

Після цього розпочинається відвантаження даних за допомогою методу UploadDataAsync(). Він відстежує кількість переданих байтів і виводить прогрес відвантаження даних на графічний інтерфейс користувача. Крім того, метод визначає швидкість передачі даних та відображає її на графічному інтерфейсі користувача.

Після завершення відвантаження даних метод GetUploadSpeed() обчислює швидкість відвантаження даних та повертає її.

2.5.4 Тестування ping та jitter

Програмна реалізація тестування ping представляє собою метод "GetPingTime()", який дозволяє виміряти час, необхідний для відправки пакетів інформації до серверу і повернення відповіді від нього, використовуючи протокол Ping.

У тілі методу використовується об'єкт "Ping", який дозволяє відправляти запити до вказаного серверу і отримувати відповідь. Об'єкт Ping знаходиться в блоці "using", що дозволяє автоматично закривати його після виконання методу, щоб уникнути витоків пам'яті.

У цьому методі встановлюються параметри Ping, такі як "DontFragment" та "Ttl". Перший параметр запобігає фрагментації пакетів при передачі, а другий обмежує максимальну кількість маршрутизаторів, які можуть бути пройдені перед поверненням відповіді.

Для відправки запиту використовується метод "SendPingAsync()", який передає адресу сервера ("www.google.com"), таймаут ("5000"

мілісекунд), буфер даних ("buffer") та параметри Ping. Після відправки запиту, очікується відповідь з сервера, яка зберігається в об'єкті "PingReply".

Якщо відповідь від сервера успішна (статус "IPStatus.Success"), то зберігається час, необхідний для відправки запиту та отримання відповіді (reply.RoundtripTime), у змінну "pingTime". В іншому випадку, "pingTime" залишається рівним "-1".

Код програмної реалізації для тестування jitter є функцією, яка отримує середнє значення затримки (ping) та "jitter" (розкид) для п'яти послідовних пінгів до веб-сайту www.google.com. Для цього використовується клас Ping, який надається в стандартній бібліотеці .NET.

Функція використовує асинхронність за допомогою ключового слова "async" та методу "await" для відправки пінгів та очікуванн результатів пінгу перед обчисленням середнього значення та розкиду. Для зменшення ризику втрати даних через переповнення буфера, встановлюється параметр DontFragment.

Середнє значення затримки обчислюється за допомогою методу "Average()", який використовується для обчислення середнього значення зі списку часів, де часи більші за 0. Розкид обчислюється шляхом обчислення суми різниць між кожним часом пінгу та середнім значенням затримки та ділення на (кількість пінгів - 1).

На виході, функція повертає значення "jitter" у мілісекундах. Оскільки це асинхронна функція, вона може бути викликана з іншого асинхронного методу або звичайної функції за допомогою ключового слова "await".

2.6 Тестування диску

Тестування швидкості запису та читання диску комп'ютера є важливим етапом в оцінці продуктивності комп'ютера та його складових частин. Це дозволяє оцінити, наскільки швидко комп'ютер може зчитувати та записувати дані на диск.

Тестування швидкості диску зазвичай виконується за допомогою послідовного читання та запису даних на диск. Під час тестування швидкості читання диску програма зчитує дані з диску та визначає швидкість, з якою вони були прочитані. Під час тестування швидкості запису диску програма записує дані на диск та визначає швидкість, з якою вони були записані.

Основним показником швидкості диску є швидкість передачі даних. Швидкість передачі даних визначає, яка кількість даних може бути передана за одиницю часу.

Оцінка швидкості диску є важливою для багатьох завдань, таких як відтворення відео, обробка зображень, графічний дизайн, розробка програмного забезпечення та багато інших завдань. Також важливо враховувати, що швидкість диску може змінюватися залежно від того, які дані записуються та зчитуються, тому тестування повинно проводитися з урахуванням реальних умов використання комп'ютера.

Тестування швидкості диску також може допомогти виявити проблеми з його працездатністю, наприклад, якщо швидкість зчитування та запису даних на диск є дуже низькою, це може свідчити про те, що диск вже зношений та потребує заміни.

Крім того, тестування швидкості диску може допомогти визначити, які компоненти комп'ютера можуть бути в узгоджені з диском. Наприклад, якщо швидкість диску дуже висока, але швидкість передачі даних на інших компонентах комп'ютера є низькою, це може означати, що інші компоненти комп'ютера потребують оновлення.

Загалом, тестування швидкості диску є важливою частиною в оцінці продуктивності комп'ютера та може допомогти виявити проблеми з його працездатністю. Для досягнення максимальної продуктивності комп'ютера варто періодично проводити тестування швидкості диску та інших компонентів комп'ютера.

2.6.1 Імплементація тестування диску

Форма для тестування диску відображає дані про швидкість запису та читання, показує модель диску, який тестується, а також користувачу доступна інформація про прогрес тестування швидкості запису та читання диску комп’ютера.

Код, який відповідає за отримання моделі диску, має на меті отримання назви пристрою (наприклад, жорсткого диску) з системи, на якій він виконується.

Для досягнення цієї мети, використовується клас ManagementObjectSearcher з бібліотеки System.Management. Цей клас дозволяє отримати колекцію об'єктів, що задовольняють певний запит, у даному випадку - запит на отримання всіх об'єктів Win32_DiskDrive, тобто всіх пристроїв зберігання даних в системі.

Далі, за допомогою циклу foreach, відбувається ітерація по кожному з отриманих об'єктів, і зберігається значення назви пристрою (властивість "Model") у змінну deviceName. Після цього цикл зупиняється за допомогою оператора break, оскільки для отримання назви потрібен лише перший пристрій зі списку.

Нарешті, змінна deviceName повертається як результат роботи функції GetDeviceName(). Це дозволяє іншим частинам програми отримувати назву пристрою та використовувати її для подальшої роботи з пристроєм.

Програмна реалізація для тестування швидкості запису та читання диску є частиною програми, яка тестує швидкість читання та запису даних на диск. Він містить дві функції, які виконуються паралельно: Worker_DoWork та Process_OutputDataReceived.

Worker_DoWork є головною функцією, яка виконується в окремому потоці. Вона виконує запуск консольної програми DiskTestConsoleApp.exe за допомогою об'єкта Process. Цей об'єкт містить інформацію про процес запущеної програми та його параметри. Після того, як програма запущена, відбувається обробка виведення результатів цієї програми у функції Process_OutputDataReceived.

Process_OutputDataReceived викликається кожен раз, коли програма виводить дані в консоль. Функція обробляє ці дані та оновлює графічний інтерфейс користувача з результатами тестування. Зокрема, функція перевіряє, чи змінилися значення показників швидкості читання та запису даних на диск. Якщо значення змінилися, вони оновлюються на графічному інтерфейсі. Крім того, функція встановлює флаг IsReadNow, який вказує, чи відбувається в даний момент читання з диска.

Загалом, цей код є досить складним та використовує багато концепцій мови програмування C#. Він демонструє, як можна взаємодіяти з консольною програмою та оновлювати графічний інтерфейс на основі результатів цієї програми.

Код, який є окремим консольним проектом, є програмою на мові C# для тестування швидкодії диска. Вона містить функції для зчитування та запису файлів асинхронно з використанням багатьох потоків, а також функцію для моніторингу завантаження фізичного диска, що забезпечує максимальне навантаження на будь-який диск.

Програма використовує буфер розміром 10 Мбайт і чергу з глибиною 100, щоб створити файли для зчитування та запису. Загальний розмір файлів розраховується як добуток глибини черги та розміру блоку. Потім

програма запускає функції запису та зчитування одночасно з використанням багатьох потоків, щоб перевірити швидкодію диска.

Після завершення запису та зчитування програма видаляє файли та зупиняє моніторинг завантаження диска. Моніторинг завантаження диска реалізований за допомогою функції, яка кожну секунду зчитує показник "Відсоток вільного часу" з лічильника продуктивності "PhysicalDisk" і виводить його на екран.

Програму можна змінювати, наприклад, змінювати розмір блоку та глибину черги, щоб вивчати вплив цих параметрів на швидкодію диска.

Також код містить метод `GetDiskLoading()`, який запускається в окремому потоці і виводить на екран навантаження на жорсткий диск раз на секунду. Цей метод зупиняється, коли змінна `stopMonitoring` стає істинною.

Метод `WriteAsync()` виконує запис даних у файли за допомогою `FileStream`. Запис відбувається `queueDepth` разів, кожного разу в файл записується блок даних розміром `blockSizeInKiB`. Запис відбувається паралельно в `threadCount` потоках. Кожен потік записує дані у свій файл з унікальним ім'ям. Під час запису на екран виводиться процент виконання запису кожного з потоків.

Метод `ReadAsync()` виконує зчитування даних з файлів, які були записані раніше методом `WriteAsync()`. Зчитування відбувається `queueDepth` разів, кожного разу зчитується блок даних розміром `blockSizeInKiB`. Зчитування відбувається паралельно в `threadCount` потоках. Кожен потік зчитує дані зі свого файлу. Під час зчитування на екран виводиться процент виконання зчитування кожного з потоків.

На кінець виконання програми всі файли, які були створені методом `WriteAsync()`, видаляються за допомогою методу `File.Delete()`.

Описуючи код методу, який повертає завантаження диску, можна зазначити наступне. Цей код містить метод `GetDiskLoading()`, який

повертає значення типу `Task<int>`. У цьому методі використовується цикл, щоб постійно отримувати дані про навантаження диску. Якщо змінна `stopMonitoring` має значення `true`, то цикл припиняється.

У циклі методу `GetDiskLoading()` використовується метод `Thread.Sleep(1000)`, щоб призупинити виконання цього циклу на 1 секунду. Після цього використовується об'єкт `PerformanceCounter`, щоб отримати значення відсотка вільного часу диску. Це значення використовується для обчислення відсотка завантаження диску, яке потім виводиться на консоль за допомогою методу `Console.WriteLine()`.

Коли змінна `stopMonitoring` отримує значення `true`, цикл завершується, і метод повертає значення 0 у вигляді об'єкта `Task<int>`. Основна мета цього методу полягає в відстеженні навантаження диску під час виконання інших задач у програмі.

Детальніше описавши код тестування швидкості запису на диск можна відзначити, що цей код представляє метод WriteAsync(), який є асинхронним та повертає цілочисельне значення. Метод приймає рядок, який відповідає за назву файлу, в який будуть записуватись дані.

В методі створюється буфер розміром blockSizeInKiB, який заповнюється випадковими даними за допомогою об'єкту Random.

Далі використовується блокування з метою запобігання змінам змісту файлу одночасно кількома потоками.

За допомогою об'єкту Stopwatch вимірюється час запису в файл. Використовується об'єкт FileStream для створення файлу з назвою, що передана в метод.

У циклі for проходиться через queueDepth блоків даних, що генеруються випадковим чином та записуються в файл за допомогою методу fs.Write() з використанням буфера даних.

Також умова i % 10 == 0 дозволяє виводити на екран відсоткову кількість записаних даних з метою відслідковування прогресу.

Детальна інформація про реалізацію тестування швидкості читання диску представляє те, що цей код містить метод `ReadAsync`, який здійснює асинхронне читання даних з файлу з заданої назвою. Метод використовує ключове слово `async`, щоб позначити, що він містить асинхронний код.

Метод приймає рядок `filename` як параметр, який вказує назву файлу для читання. В методі створюється буфер розміром `blockSizeInKiB` байтів, який буде використовуватися для зчитування даних з файлу блоками. Далі використовується блокування `lock` для запобігання одночасного доступу до файлу з боку різних потоків.

Для читання використовується об'єкт `FileStream`, який відкривається з використанням методу `File.OpenRead`, передаючи в нього назву файлу для читання. У циклі `for` розміром `queueDepth` ітерацій зчитуються дані з файлу у буфер методом `fs.Read`. Умова `if (i % 10 == 0)` перевіряє, чи потрібно виводити в консоль повідомлення про прогрес читання, що відбувається кожних 10 ітерацій.

Метод повертає ціле число 0, що є ознакою успішного завершення читання даних з файлу. Код методу обгорнутий в конструкцію `Task.Run`, щоб він міг бути виконаний асинхронно. Ключове слово `await` перед викликом методу `Task.Run` означає, що виклик буде здійснено асинхронно, і виконання методу буде призупинене до того моменту, поки результат не стане доступним.

Цей код може бути корисним для тестування пропускної здатності дискової системи. Він дозволяє одночасно виконувати паралельно запис та зчитування даних з диску, і виводить на екран навантаження на жорсткий диск. Крім того, він дозволяє виміряти пропускну здатність дискової системи за допомогою розрахунку кількості записаних або зчитаних даних на одиницю часу.

2.7 Аналіз показників температур, навантажень та оцінки

Для порівняння показників, тестування проводилось на двох системах (див. Табл. 2.7). Всі тестування проводились на однаковій версії програми та в однаковий час.

Таблиця 2.7 Системи, використані при тестуванні

Назва системи	CPU	GPU	RAM	OS
Система 1	AMD E2-3000M	AMD Radeon HD 6380G	8GB	Windows 10, Home
Система 2	AMD Ryzen 5 4600H	NVIDIA RTX 2060	8GB	Windows 10, Pro

Системи сильно відрізняються між собою за продуктивністю та виробниками комплектуючих, що дозволить тим самим протестувати програму для користування широким колом користувачів та їхніх систем.

Таблиця 2.7.1 Порівняння результатів тестування CPU

Найменування системи	Single-Core	Multi-Core
Система 1	770	1443
Система 2	2920	18325

Таблиця 2.7.2 Порівняння результатів тестування GPU

Найменування системи	Відсоткове відношення результатів, %
Система 1	1.9
Система 2	100

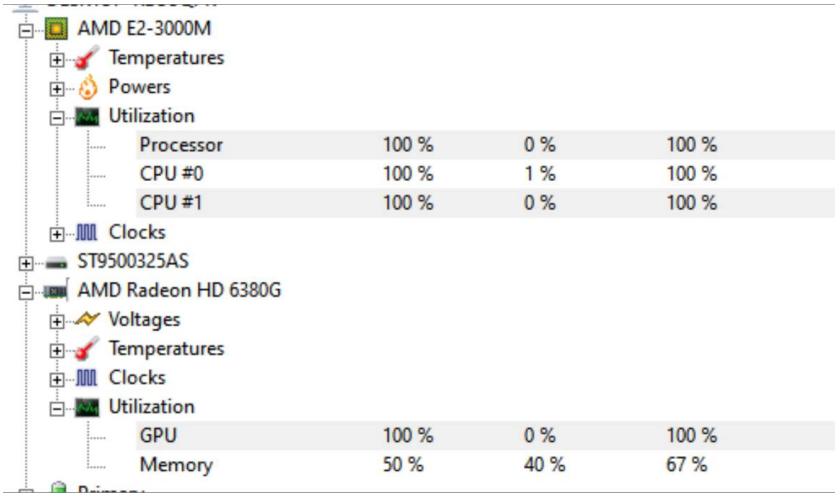

Рисунок 2.7 Показники навантаження Системи 1 під час стрес-тестування

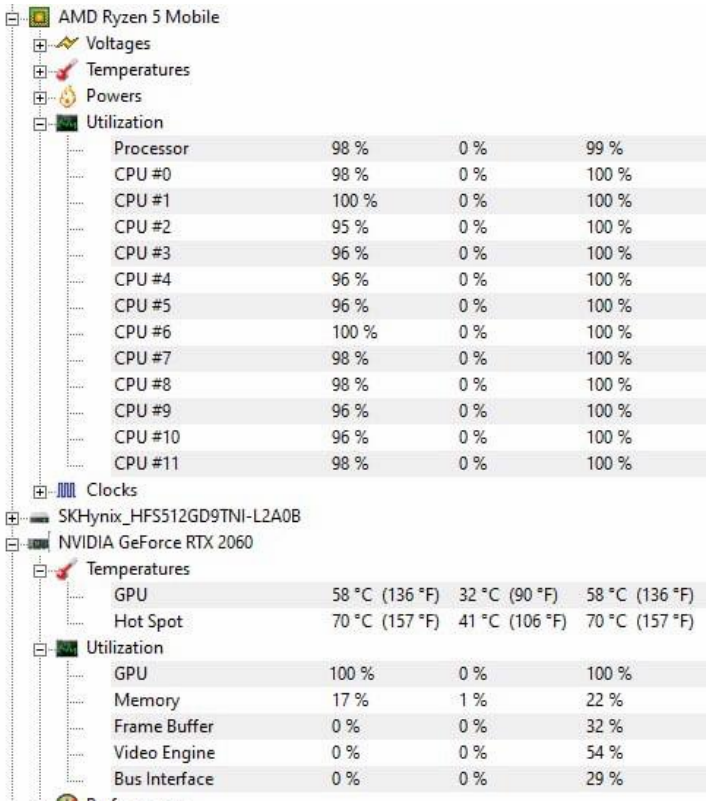

Рисунок 2.7.1 Показники навантаження Системи 2 під час стрес-тестування

Процесор Системи 2 набирає, очевидно, набагато більші бали (див. Табл. 2.7.1). Однак, ще більшою різниця є між продуктивність відеокарт цих двох систем (див. Табл. 2.7.2). Також можна переконатися у ефективності функції стрес-тестування обох систем (див. Рис. 2.7 та Рис. 2.7.1).

2.8 Рейтинг продуктивності систем

Для реалізації рейтингу за продуктивністю, можна використовувати вбудовану таблицю в застосунку, яка містить інформацію про тестовані системи та їх результати тестувань. Також можна реалізувати можливість сортування та пошуку в цій таблиці. Для цього можна використовувати System.LINQ, а дані можна відображати за допомогою віджета DataGridView.

Таблиця рейтингу містить наступні дані:

- позиція системи в рейтингу
- назву CPU
- назву GPU
- ємність RAM
- найменування OS
- кількість ядер
- швидкість процесора
- результати тестування Single-Core
- результати тестування Multi-Core
- результати тестування GPU

За допомогою System.LINQ можна виконувати пошук та сортування в цій таблиці, що дозволяє знаходити певні системи за заданими критеріями або сортувати їх за певними параметрами. Віджет DataGridView дозволяє відображати дані з таблиці у зручному табличному форматі.

2.8.1 Dropbox API

За допомогою технології Dropbox API розробники з відповідним акаунтом можуть керувати та зберігати дані своїх застосунків у хмарному сховищі Dropbox (див. Рис. 2.8.1). Для реалізації системи рейтингу

продуктивності використовується саме ця технологія. Після успішного тестування процесора (CPU) та відеокарти (GPU), дані результатів системи автоматично завантажуються в хмарне сховище у форматі JSON-файлу. Оскільки цей застосунок вимагає зберігання лише однієї таблиці і метою є зробити його безкоштовним та використовувати платні хостинги, оптимальним рішенням є використання легко серіалізованого JSON-файлу замість бази даних. Ще однією перевагою цього підходу є невеликий обсяг даних, які потрібно зберігати. Навіть при тисячах записів, розмір файлу не перевищує кількох мегабайтів. Таким чином, ми отримуємо систему рейтингу, в якій результати зберігаються на віддаленому хмарному сховищі.

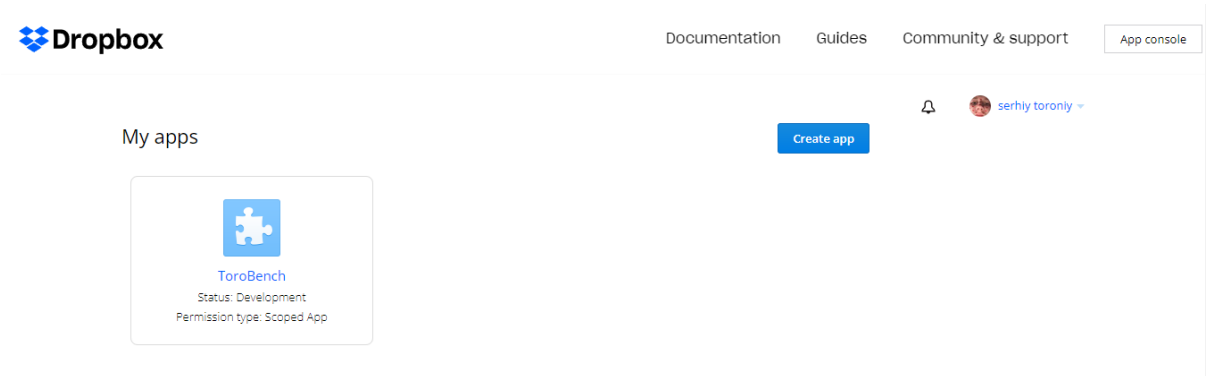

Рисунок 2.8.1 Кабінет розробника Dropbox

2.9 Функціонал автооновлення застосунку

Автооновлення застосунку можна поділити на два типи. Перший тип використовує оновлення за допомогою GitHub, що передбачає ручне автооновлення шляхом клонування репозиторію GitHub. Цей підхід відповідає застосункам з відкритим вихідним кодом і найбільше підходить для досвідчених користувачів.

Другий тип автооновлення реалізовано за допомогою сторонньої бібліотеки AutoUpdater.NET. В цьому випадку користувачу достатньо натиснути одну кнопку, і його програма оновиться до останньої версії всього за кілька секунд. Для реалізації цього функціоналу, крім бібліотеки AutoUpdater.NET, використовується також хмарне сховище Dropbox, про яке згадувалося в розділі 2.8.1. У хмарному сховищі Dropbox зберігається архів з новим пакетом оновлення, а також файл-конфігуратор (див. Рис. 2.9), який керує поведінкою оновлення. За допомогою редагування цього файлу можна налаштувати пропуск певних версій оновлення, задати дату встановлення майбутніх оновлень та налаштувати самі версії оновлень.

Бенчмарк здійснює запит до хмарного середовища та перевіряє конфігурацію файлу. В результаті користувач отримує сповіщення про наявність нового оновлення і може виконати певні дії на свій розсуд, або користувач буде повідомлений про відсутність нового пакету оновлень.

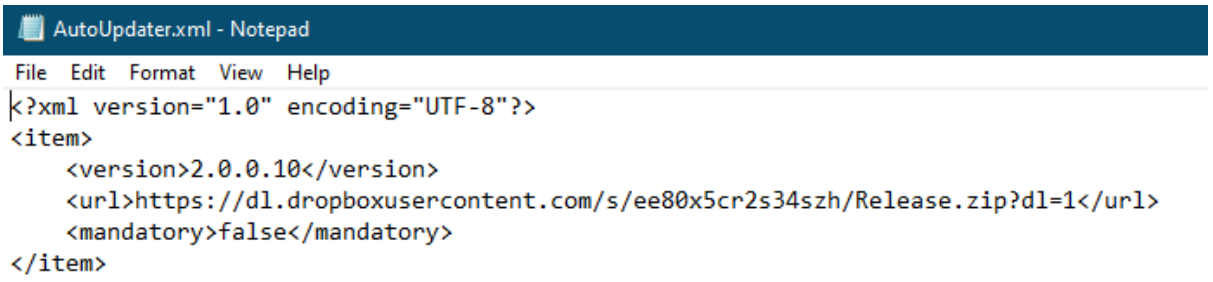

Рисунок 2.9 Файл-конфігуратор автооновлень

2.10 Кастомізація вигляду програми

У програмі бенчмарків, крім основного функціоналу, також присутні додаткові можливості для зміни зовнішнього вигляду застосунку. Користувач має доступ до трьох тем: світлої, темної та системної.

Системна тема автоматично встановлюється відповідно до налаштувань операційної системи, буде використана світла або темна тема. Програма звертається до системного реєстру та перевіряє значення відповідного ключа, щоб визначити поточну тему пристрою. Ця функція особливо зручна для користувачів, які працюють в різний час доби. У реалізації цієї функції використовується бібліотека Guna Framework, згадана в підрозділі 2.1, яка дозволяє змінювати кольори віджетів відповідно до вибраної теми. В підрозділі 3.6 наведено зразки вигляду окремих вікон програми при різних темах.

Крім системної теми, користувач може вибрати світлу або темну тему. Кольорова схема віджетів програми розроблена з орієнтацією на простоту та зручність для користувача. У світлій темі використовуються світлі та приємні кольори, що роблять програму зрозумілою та легкою використовувати. У темній темі використовуються темні та приємні кольори, що забезпечують комфортну роботу в умовах обмеженого освітлення.

Усі зміни, внесені до зовнішнього вигляду програми, можуть бути збережені та застосовані під час наступного запуску. Це дозволяє користувачам зручно налаштовувати програму відповідно до своїх потреб та уподобань.

Загалом, можливість налаштування зовнішнього вигляду програми дозволяє користувачам зручно та комфортно працювати з програмою, поліпшує їх досвід користування та забезпечує вищу продуктивність та ефективність.

2.11 Інсталятор для програми

Для реалізація інсталятора було використано вбудований Setup-проект від .NET Framework (див. Рис. 2.11). В проекті містяться

файли програми та проект сконфігуровано для встановлення однієї версії інсталятора. Інсталятор містить наступні етапи встановлення програми:

- вибір папки для інсталяції
- вибір користувача системи

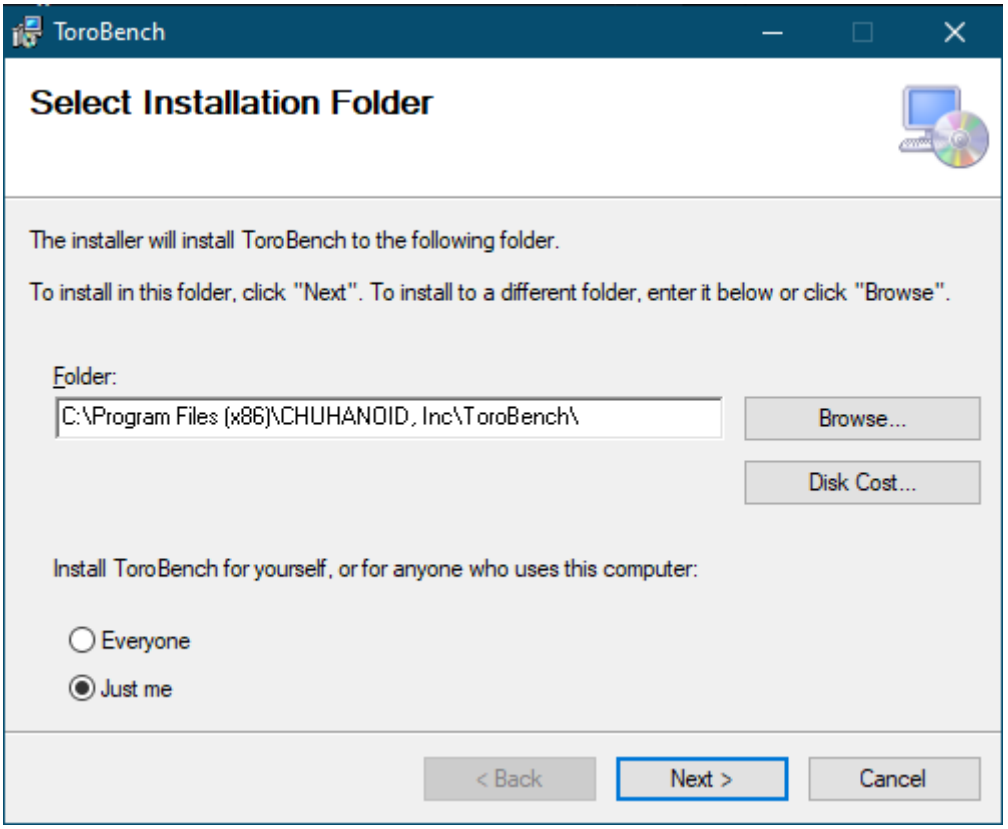

Рисунок 2.11 Вікно інсталятора програми

Інсталятор створює ярлики на робочому столі та в меню Пуск для швидкого доступу до програми.

2.11.1 Файлова система інсталятора програми

Для того, щоб проект інсталятора програми міг в результати побудови повернути файл інсталювання типу .msi потрібно правильно сконфігурувати файлову систему проекту, а саме такі папки як “Application Folder”, “User’s Desktop” та “User’s Programs Menu”.

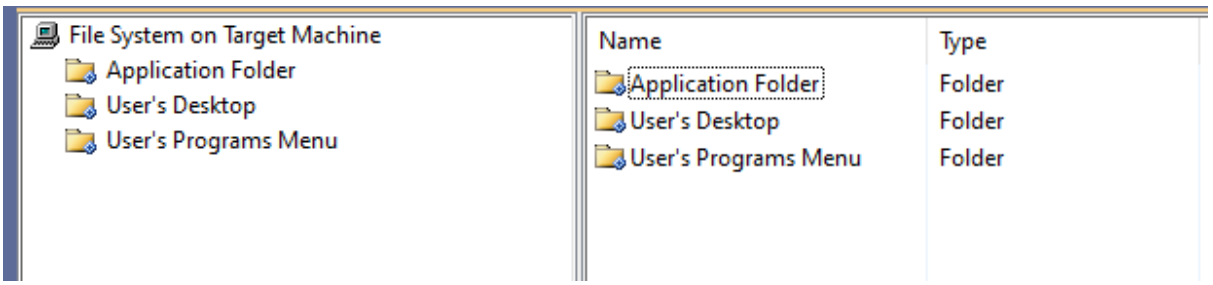

Рисунок 2.11.1.1 Файлова система проекту інсталятора

Папка “Application Folder” відповідає за вміст папки самого застосунку після встановлення користувачем. Всі файли в цій папці будуть розташовані в основній папці програми в такому ж стані як і в цій папці. Тут розташовані файли самого проекту, додаткові бібліотека та фреймворки, а також основний файл ToroBench.exe. Крім того ця папка містить окремі підпапки такі як ‘img’ та ‘updates’. Кожна з цих папок містить ще деякі окремі файли: програмні зображення та інформацію про інсталятори оновлення відповідно.Також тут міститься підпроект який відповідає за консольне середовище тестування швидкості запису та читання диску (див. 2.6).

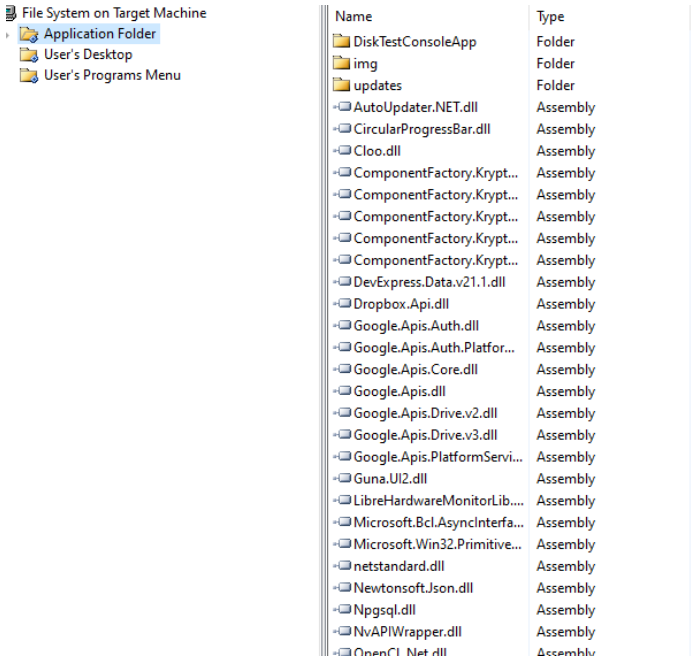

Рисунок 2.11.1.2 Вміст папки ‘Application Folder’

Папка “User’s Desktop” містить лише ярлик на програмний вивід, який знаходиться в основній папці програми. Ця папка відповідає за робочий стіл користувача після інсталювання програми, тобто, що саме буде знаходитись на робочому столі. В даному випадку після встановлення користувач буде мати швидкий доступ до відкриття програми на своєму робочому столі у вигляді ярлика із назвою програм.

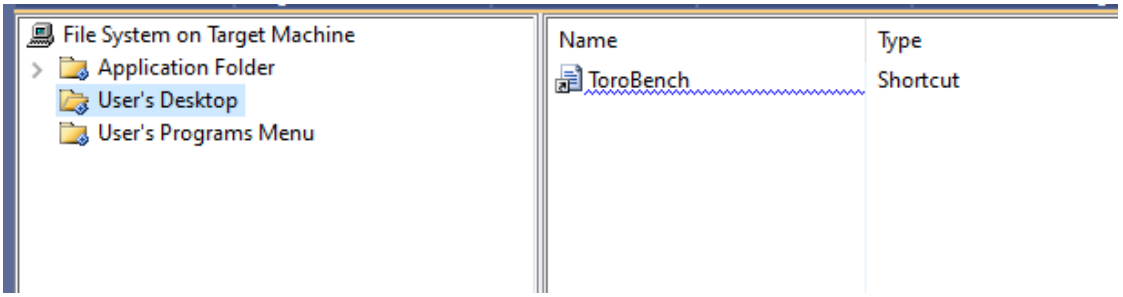

Рисунок 2.11.1.3 Вміст папки ‘User’s Desktop’

Папка “User’s Programs Menu” також містить тільки ярлик на основний файл програми, і дозволяє швидко доступитися до нього через кілька кліків. Також ця папка відповідає за розташування та перелік файлів в меню Пуск операційної системи. В даному випадку після інсталювання програми користувач матиме в меню Пуск ярлик на основну програму і зможе швидко та зручно запустити її в будь-який момент.

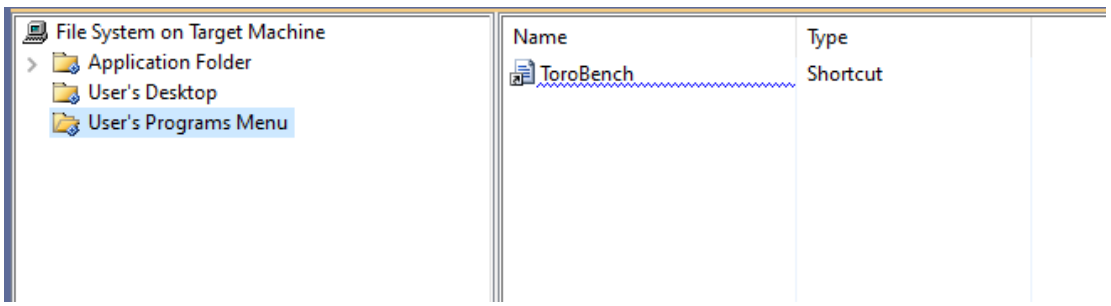

Рисунок 2.11.1.4 Вміст папки ‘User’s Programs Menu’

3. Демонстрація роботи програмного забезпечення

В наступних підрозділах буде наведено опис та зображення зовнішнього вигляду програми з позиції користувача. Також буде проведено порівняння результатів тестування з іншими бенчмарками-аналогами.

3.1 Головне вікно

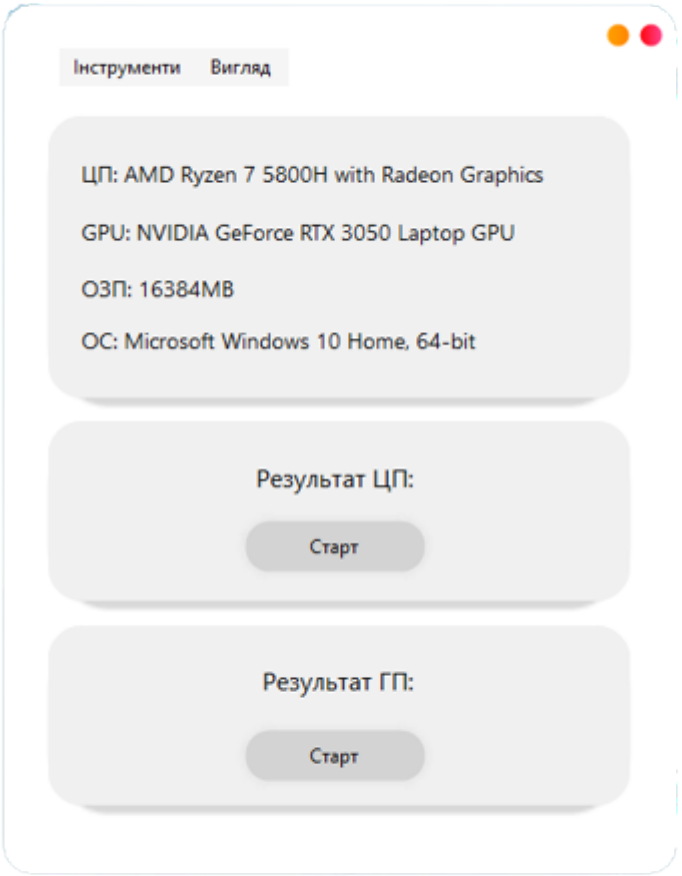

Рисунок 3.1 Вигляд головного вікна програми

На головному вікні програми міститься інформація про комплектуючі системи, також верхнє меню (див. Рис. 3.1). В цьому вікні є кнопки тестування CPU та GPU відповідно.

3.2 Тестування CPU та GPU

Тестування центрального та графічного процесорів відбувається у вікні ProgressForm, яке має однаковий зовнішній вигляд для обох тестів (див. Рис. 3.2). У цьому вікні також проводиться тестування режимів Single-Core та Multi-Core. Спочатку виконується тестування в однопотоковому режимі, а потім в багатопотоковому. Користувач має можливість в будь-який момент припинити виконання тестування, а також відображається прогрес виконання, що дозволяє користувачеві відстежувати поточний стан процесу.

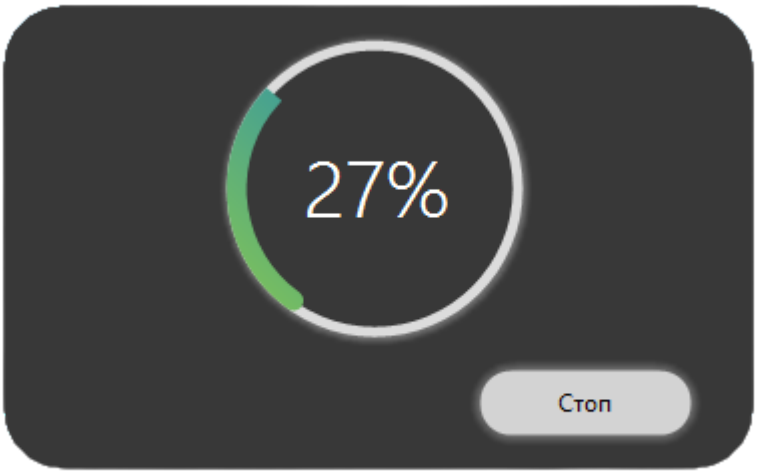

Рисунок 3.2 Вікно тестування CPU та GPU

3.3 Стрес-тестування

Вікно тестування надає користувачеві докладну інформацію про CPU та GPU. В ньому відображаються дані щодо відсоткового навантаження, температурних показників та оцінки продуктивності кожного компонента, які оновлюються в реальному часі (див. Рис. 3.3). Також програма показує граничні значення цих параметрів для виявлення потенційних аномалій. У вікні також присутня інформація про час, протягом якого виконувалося

тестування. Користувач може в будь-який момент припинити виконання тестування, якщо потрібно.

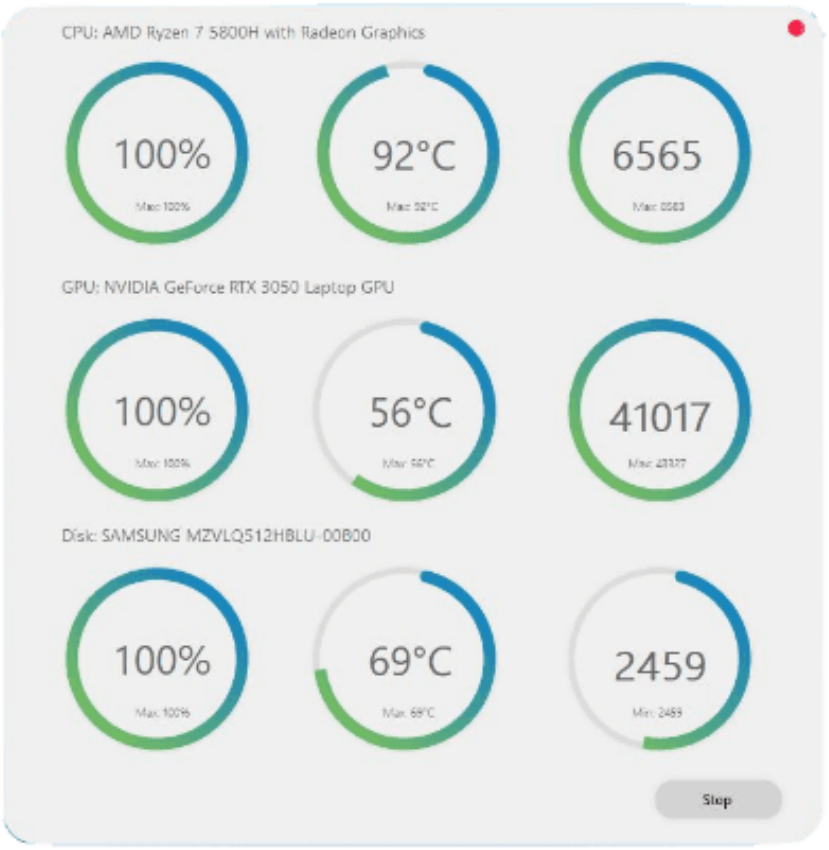

Рисунок 3.3 Вікно стрес-тестування

У вікні тестування також присутня інформація про диск комп'ютера. Завантаження диску відображається у відсотках і оновлюється в реальному часі прямо в програмі. Температура диску вимірюється у градусах Цельсія. Бал тестування, що знаходиться в останньому секторі сенсорів і показників диску, показує поточну швидкість послідовного читання даних з диска. Кожен сенсор диску також має граничні значення: мінімальні для швидкості читання та максимальні для завантаження диску і температури диску.

3.4 Рейтинг

У вікні рейтингу представлена інформація про найкращі результати систем з відповідними комплектуючими. Користувач має можливість сортувати дані в таблиці за різними колонками, а також виконувати пошук за певними значеннями. Це дозволяє зручно організувати та знайти потрібні результати в рейтингу (див. Рис. 3.4).

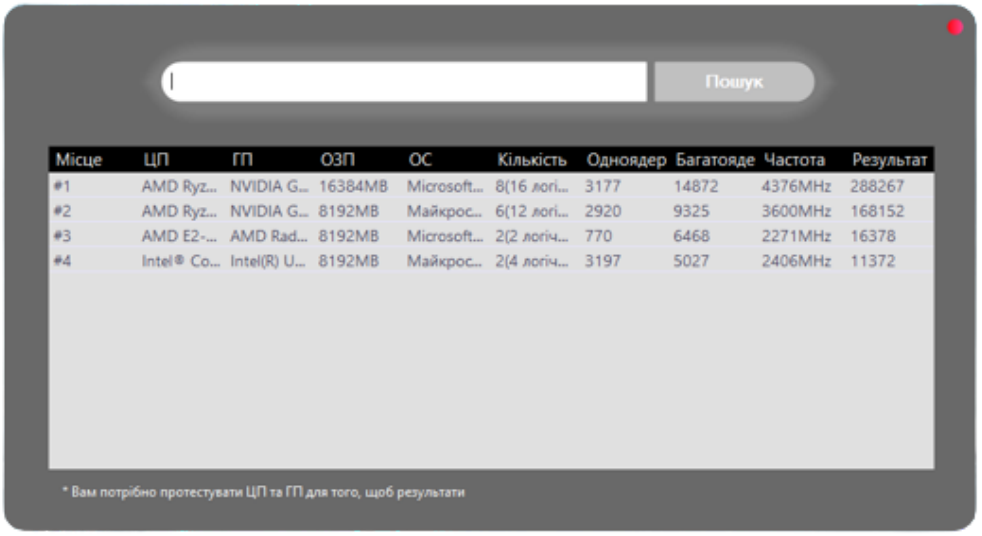

Рисунок 3.4 Вікно рейтингу

На вікні програми доступний пошук інформації про результати тестування. Користувач може ввести пошуковий запит в текстове поле і розпочати пошук, натиснувши кнопку "Find". Це дозволяє знайти потрібну інформацію в таблиці.

Проте, варто враховувати, що дані будуть доступні в рейтингу лише після проведення тестування процесора і відеокарти. Якщо було протестовано лише один з цих компонентів, дані не будуть занесені до таблиці рейтингу. Це робиться для того, щоб уникнути наявності порожніх

значень у таблиці і дозволяє отримати більш детальну інформацію про збірку, на якій був протестований конкретний компонент системи.

3.5 Тестування швидкості диску

Вікно тестування диску містить інформацію про швидкість запису та читання з диску, модель якого зображена вверху над компонентами, які відображають прогрес виконання, кожного підтипу цього тестування.

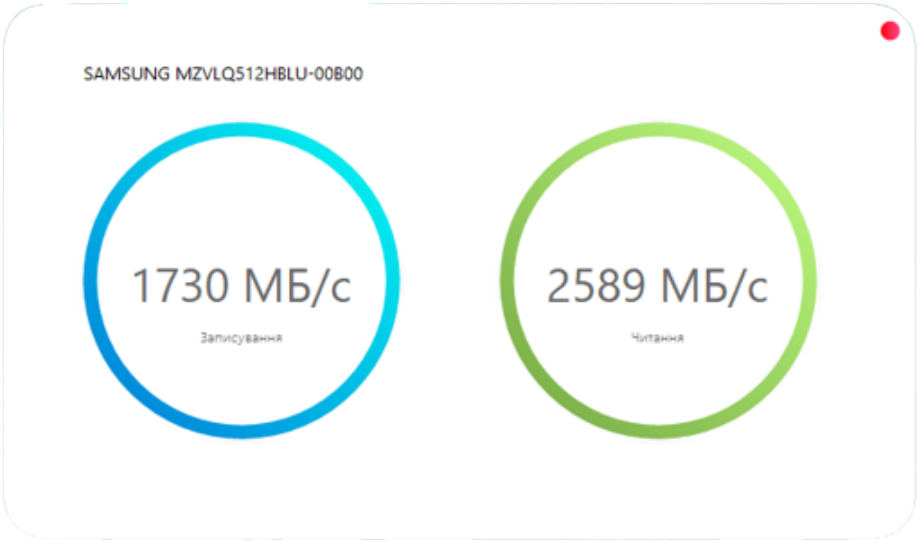

Рис. 3.5 Вікно тестування диску

Компоненти прогресу відображаються різними кольорами для більш зручного користування та розуміння для користувача. Крім того, користувач під час тестування бачить відсоткове завершення тестування до його закінчення і в кінці остаточні результати тестування. Користувач не може зупинити виконання тестування оскільки програмна реалізація передбачає виконання тестування в іншому процесі (див. 2.6).

3.6 Тестування швидкості інтернету

Вікно тестування швидкості інтернету виглядає подібно до вікна тестування швидкості запису та читання диску (див. 3.5). На самому вікні зображені два компоненти, які відображенні різними кольорами для кращого сприйняття інформації користувачем. Ці компоненти містять інформацію про швидкість завантаження та відвантаження мережі. Модель девайсу, який відповідає за доступ в мережу Інтернет системи користувача знаходиться зверху у вікні тестування.

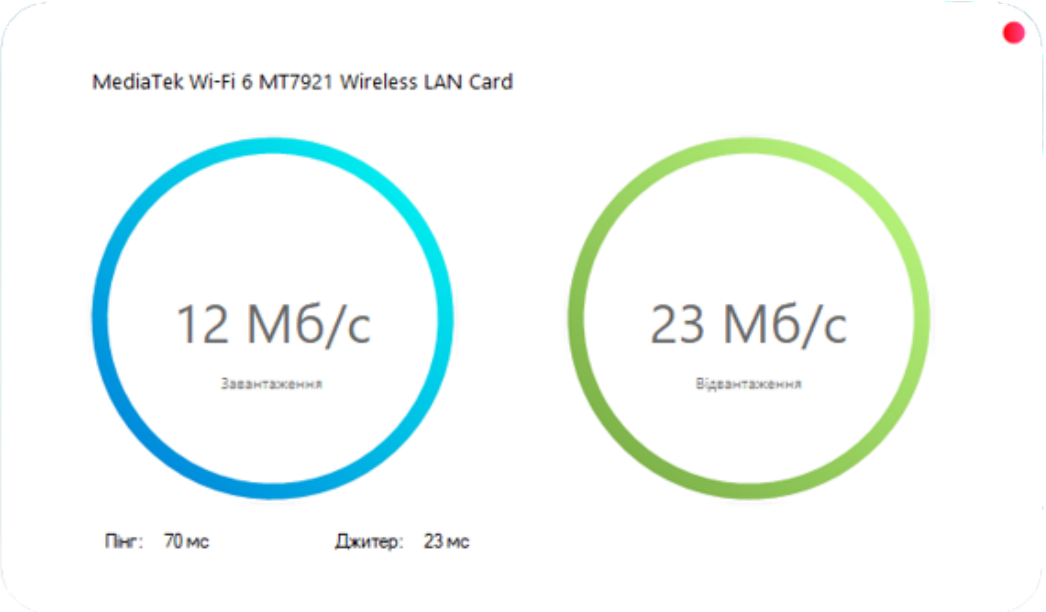

Рис. 3.6 Вікно тестування швидкості інтернету

Швидкість завантаження та відвантаження даних в інтернет оновлюється в реальному часі і виконується послідовно. Крім того слід зазначити, що анімація самого компонента, який відповідає за відображення прогресу тестування пришвидшується чи сповільнюється залежно від швидкості самого з’єднання на комп’ютері.

Такі показники як ping та jitter зображуються в самому низу вікна

тестування та показують інформацію про ці параметри мережі першими та перед самим тестуванням швидкості завантаження та відвантаження даних.

3.7 Автооновлення

В автооновленні програми вікно містить інформацію про поточну версію оновлення та функціонал, пов'язаний з плануванням майбутніх оновлень, можливістю пропуску або встановлення оновлень, а також додатковий функціонал бібліотеки AutoUpdater.NET (див. Рис. 3.7.1).

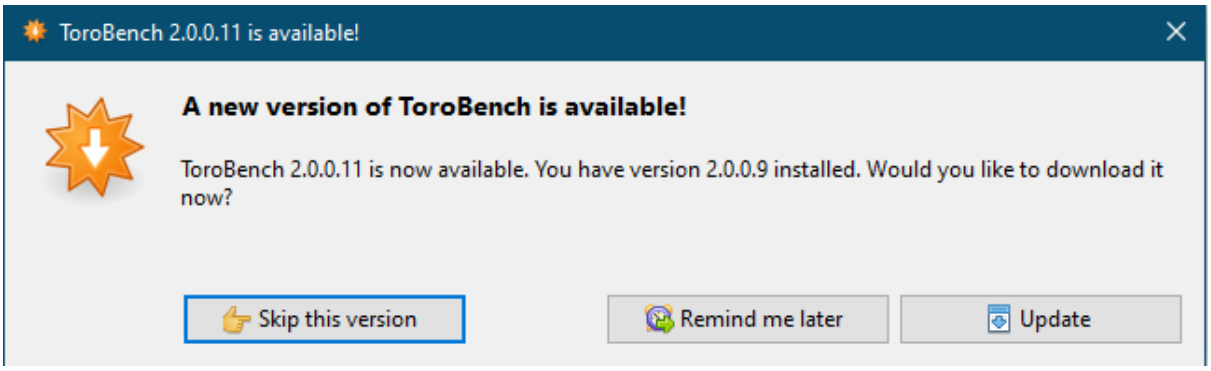

Рисунок 3.7.1 функціонал бібліотеки AutoUpdater.NET

Якщо оновлення недоступне і версія програми, яка вже присутня на комп'ютері користувача, є актуальною та останньою, програма повідомить про це користувача у відокремленому діалоговому вікні, яке відображається після завершення перевірки на наявність оновлень основною програмою.

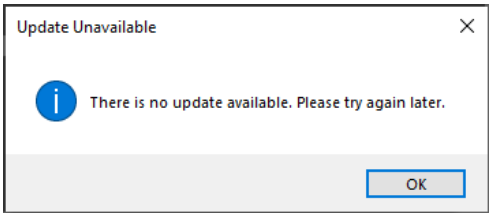

Рисунок 3.7.2 Діалогове вікно про відсутність оновлень

3.8 Кастомізація

Деталі про кастомізація описано в підрозділі 2.10. Тут наведено вигляд головного вікна в світлій та темній теми (див. Рис. 3.8).

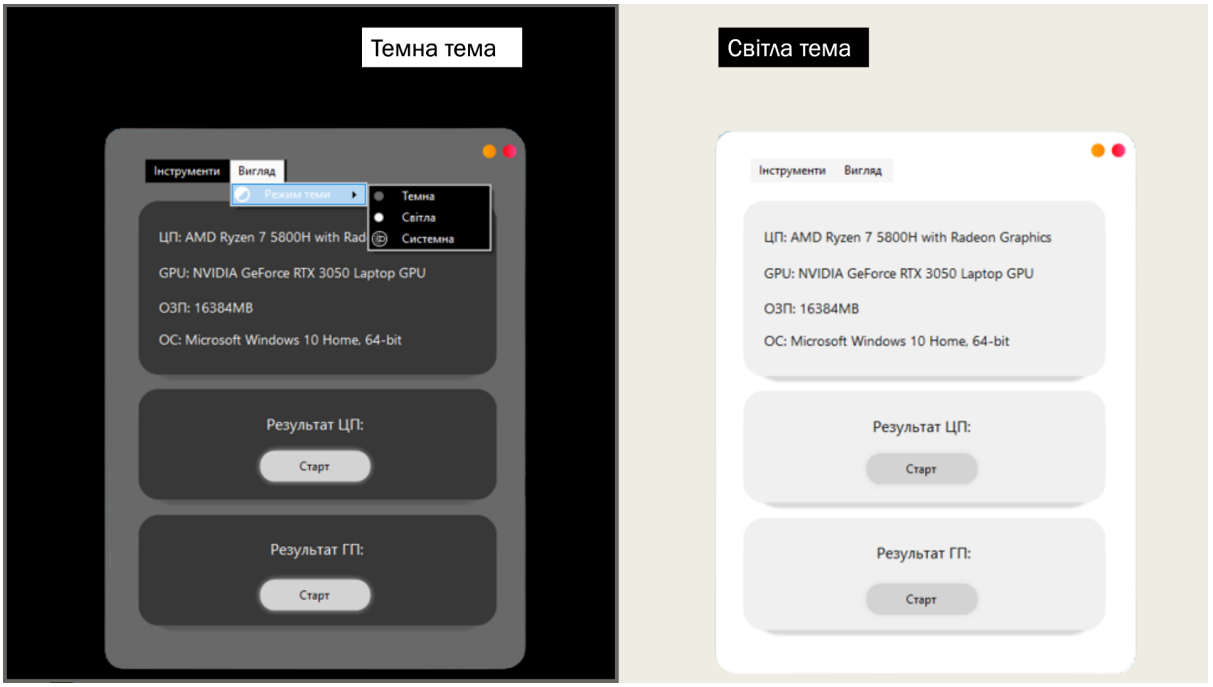

Рисунок 3.8 Вигляд головного вікна світлої та темної теми

Зміна теми вигляду програми призводить до значних змін кольорів, які були підібрані для поліпшення користувацького досвіду. Вибір теми залежить від умов використання програми. Наприклад, якщо користувач тестує свій пристрій у добре освітленому приміщенні або на вулиці, рекомендовано використовувати світлу тему для кращої чіткості та відображення програми. У випадку, коли приміщення є темним або тестування відбувається в темний час доби, доцільно переключитись на темну тему для зручності користування та збереження здоров'я очей.

У сучасному світі існують сторонні програми, які можуть автоматично змінювати вигляд програми в залежності від пори доби або користувацького графіка. У такому випадку можна вибрати опцію "System", яка використовує тему операційної системи користувача і автоматично змінює тему програми відповідно до системної теми.

3.9 Порівняння з аналогами

В наступних підрозділах наведено порівняння з відомими у світі аналогами. Порівняно їхні результати тестування та загальний функціонал.

3.9.1 Geekbench 5

Geekbench 5 — це один із найпопулярніших бенчмарків у світі.

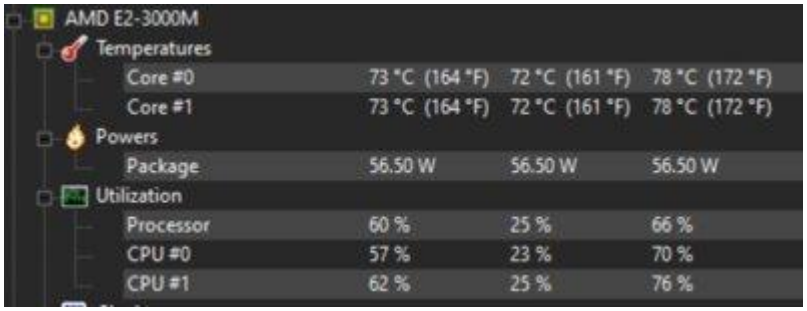

Рисунок 3.9.1 Навантаження ЦП при тестуванні Geekbench 5

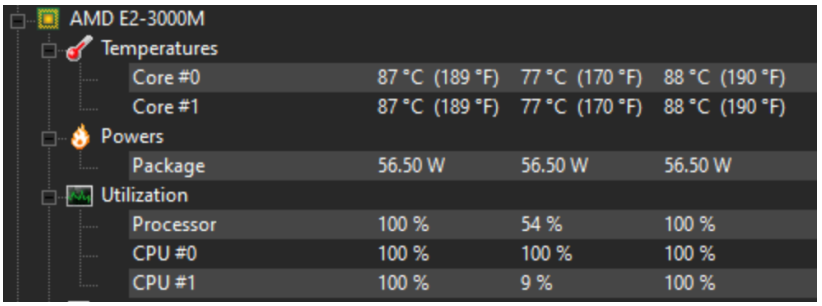

Рисунок 3.9.2 Навантаження ЦП при тестуванні ToroBench

Порівняючи продукт з відомим аналогом - Geekbench 5, можна зробити наступні висновки та порівняння. По-перше, більшість аналогів ToroBench не є повністю безкоштовними, включаючи Geekbench 5. ТороBench надає більш широкий функціонал, такий як стрес-тестування CPU/GPU та вбудований рейтинг, що не присутні у Geekbench 5.

По-друге, показники навантаження та температур комплектуючих під час тестування в ToroBench (див. Рис. 3.9.2) є більшими, ніж у Geekbench 5 (див. Рис. 3.9.1). Це є ключовою особливістю програмного забезпечення ToroBench, оскільки воно надає більш інтенсивні навантаження на компоненти системи, що дозволяє виявляти їхню максимальну швидкодію та стійкість.

Отже, ToroBench видається привабливим варіантом для тестування компонентів системи, зокрема CPU та GPU, з врахуванням його більш широкого функціоналу та більш вимогливих навантажень, що можуть забезпечити більш детальну та точну оцінку продуктивності та стійкості компонентів.

3.9.2 CPU-Z та GPU-Z

Порівнюючи з також не менш відомими бенчмарками CPU-Z та GPU-Z можна дійти висновку, що в CPU-Z та GPU-Z не вбудовані додаткові функції (онлайн рейтинг та вимірювання швидкості хешування GPU) та й взагалі ці програми поділені між собою за функціоналом, що зрозуміло за назвами застосунків (див. Рис. 3.9.2.1 та Рис. 3.9.2.2).

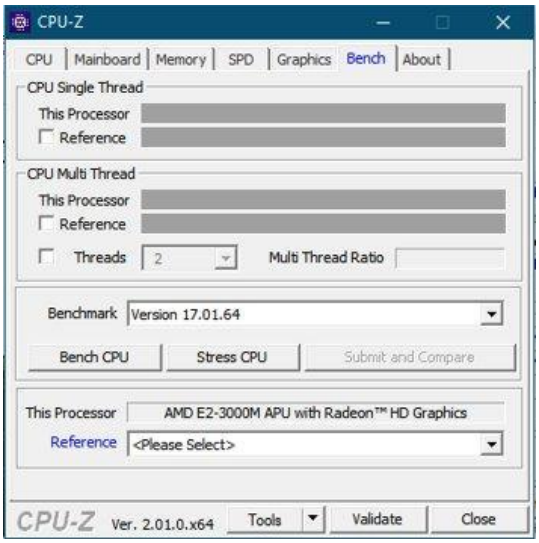

Рисунок 3.9.2.1 Вікно бенчмарка CPU-Z

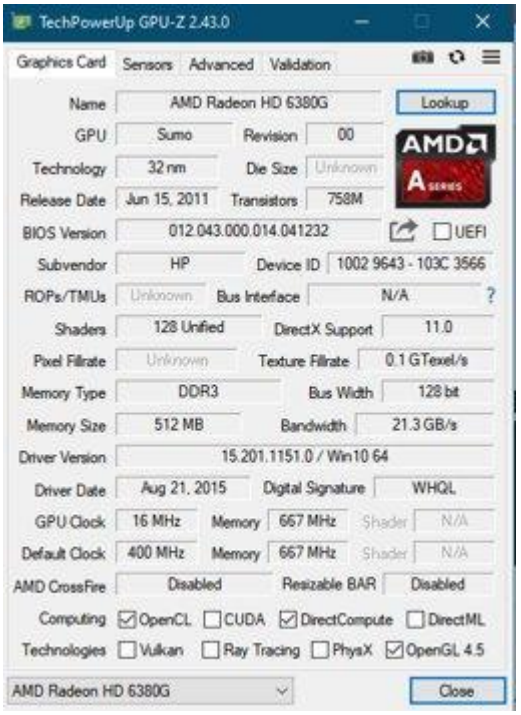

Рисунок 3.9.2.2 Вікно бенчмарка GPU-Z

3.9.3 CrystalDiskMark

CrystalDiskMark - це безкоштовна програма для тестування швидкості читання та запису на накопичувачах даних. Вона дозволяє визначити швидкість передачі даних на різних типах накопичувачів, таких як жорсткі диски (HDD), твердотільні накопичувачі (SSD) та флеш-накопичувачі (USB-накопичувачі).

Програма має декілька режимів тестування, включаючи послідовний та випадковий доступ до даних. Вона також дозволяє налаштувати розмір блоку даних для тестування та кількість повторень тестування, що дає можливість проводити більш детальний аналіз продуктивності накопичувача.

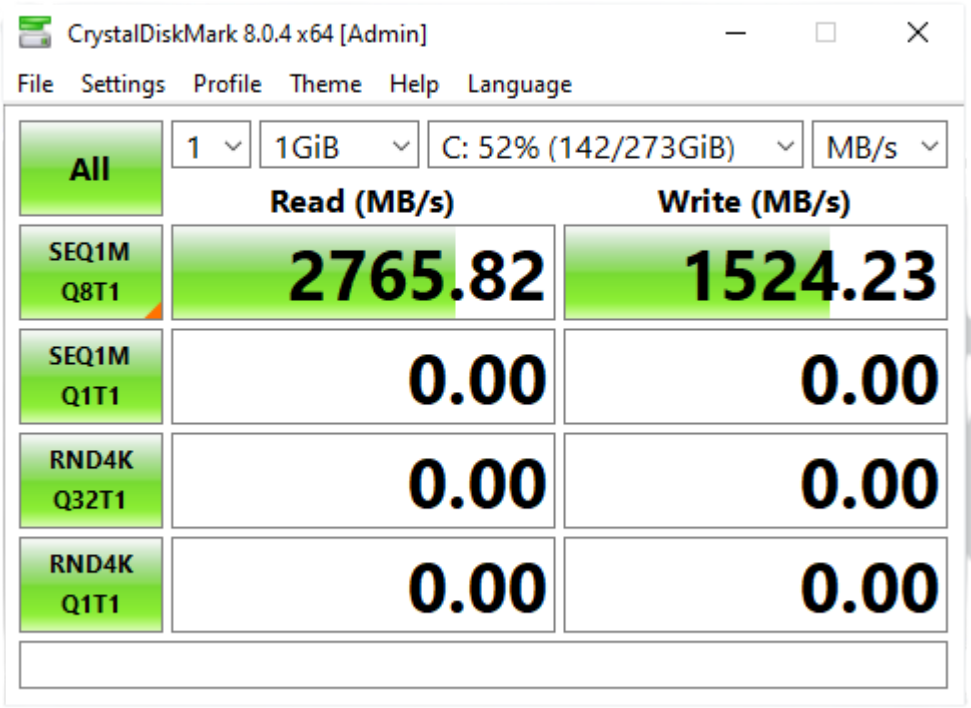

Рисунок 3.9.3.1 Вікно програми CrystalDiskMark.

CrystalDiskMark є популярним інструментом для визначення продуктивності накопичувачів, що дозволяє вибрати найбільш оптимальну модель для задач, що потребують великої швидкості передачі даних, таких як відеомонтаж, ігри та робота з великими об'ємами даних.

Основною перевагою тестування швидкості читання та запису на диск, яке імплементовано в застосунку ToroBench, полягає в тому, що воно займає більше часу та містить кращу інтерфейсну взаємодію користувача під час тестування самого диску. Отримання майже однакових результатів, порівнюючи з цим аналогом, але за менший проміжок часу та із зручним інформуванням в реальному часі позначає велику перевагу цього типу тестування в ToroBench, порівняно з аналогом CrystalDiskMark.

ВИСНОВКИ

Було створено десктопний застосунок з низкою додаткових функцій для тестування продуктивності персональних комп'ютерів. Процес розробки включав освоєння десктопної розробки, роботу з хмарними сховищами та використання інших важливих бібліотек, специфічних для бенчмарків. Також було проведено дослідження роботи комплектуючих ПК та їхньої програмної взаємодії.

На сьогоднішній день програма повністю підтримується розробником, випускаються регулярні оновлення, а вихідний код доступний на GitHub. Бенчмарк можна встановити як звичайну програму на будь-якому ПК. Програмне забезпечення пропонує швидкий і простий спосіб тестування продуктивності ПК та містить в собі онлайн-рейтинг для порівняння результатів з іншими користувачами.

Додаток надає такі можливості:

- Тестування швидкодії арифметичних, стрічкових та інших операцій процесора (CPU).

- Тестування швидкодії операцій, пов'язаних з відео, на графічному процесорі (GPU).

- Тестування швидкості читання та запису на диск.

- Тестування швидкості Інтернет-з'єднання.

- Тестування температурних показників та виявлення троттлингу під час тривалих та важких навантажень.

- Реалізація онлайн-рейтингу продуктивності ПК.

- Автоматичне оновлення додатку.

Крім того, додаток дозволяє зберігати результати тестування для подальшого аналізу та порівняння з результатами інших ПК. Користувачі можуть переглядати свої результати на своєму комп'ютері і порівнювати їх з іншими результатами. Також є можливість автоматичного запуску

тестування, що дозволяє проводити тривалі тести без участі користувача, що є корисним для оцінки стабільності ПК.

Загалом, додаток є потужним і корисним інструментом для тестування продуктивності ПК з багатьма функціями та можливостями. Він доступний для завантаження та використання на операційній системі Windows і постійно оновлюється розробником з новими функціями та виправленнями помилок. Основна мета створення безкоштовного та простого у використанні бенчмарку для оцінки продуктивності ПК під ОС Windows була досягнута.

До додатку можна доступитися за посиланням на GitHub-репозиторій за одинадцятим[11] посиланням в списку використаної літератури та інсталювати собі на комп’ютер, використовуючи msi-файл.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Windows Forms documentation. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/?view=netdesktop-7.0&viewFallbackFrom=netdesktop-5.0

2. .NET Documentation. [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/?view=net-5.0

3. Developers - Dropbox. [Електронний ресурс]. – Режим доступу: https://www.dropbox.com/developers

4. Guna Framework. [Електронний ресурс]. – Режим доступу: https://gunaui.com/

5. OpenCL Overview. [Електронний ресурс]. – Режим доступу: https://www.khronos.org/opencl/

6. CUDAfy.NET. [Електронний ресурс]. – Режим доступу: https://github.com/lepoco/CUDAfy.NET

7. CrystalDiskMark Official Website. [Електронний ресурс]. – Режим доступу: https://crystalmark.info/en/software/crystaldiskmark/

8. PassMark Software. [Електронний ресурс]. – Режим доступу: https://www.passmark.com/

9. NetworkInterface.Speed Property. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.net.networkinformation.networkinterface.speed?view=net-8.0

10. How to get internet speed in C#. [Електронний ресурс]. – Режим доступу: https://www.codeproject.com/Questions/852339/How-to-get-internet-speed-in-Csharp

11. ToroBench. [Електронний ресурс]. – Режим доступу: https://github.com/SerhiyToroniy/ToroBench