НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Програмно-апаратний комплекс для векторно-матричних обчислень

Виконав: студент 4 курсу, групи ІО-13
(шифр групи)

Маруженко Іван Сергійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н., доцент Корочкін О.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас. Гончаренко О.О. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доцент Тарасенко-Клятченко О.В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Михайло НОВОТАРСЬКИЙ
(підпис)
“__” _____2025 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Маруженка Івана Сергійовича

1. Тема проєкту Програмно-апаратний комплекс для векторно-матричних обчислень
керівник проєкту доцент, к.т.н., доцент Корочкін О.В.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року № 1705.с
2. Строк подання студентом проєкту *12 червня 2025 р.*
3. Вихідні дані до проєкту *технічна докуменація, статичтияні дані*
4. Зміст пояснювальної записки (перелік завдань, які потрібно розробити)
Розділ 1. Огляд існуючих рішень побудови програмно-апаратного комплексу.
Розділ 2. Розробка апаратної складової програмно-апаратного комплексу.
Розділ 3. Розробка і тестування програмного забезпечення комплексу.
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень) *структурна схема програмно-апаратного комплексу, структурна схема обраного процесора програмно-апаратного комплексу, блок-схеми алгоритмів потоків.*

6. Консультанти розділів проєкту

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	ас. Гончаренко О.О.		

7. Дата видачі завдання 14.04.2025

Календарний план

№ п/п	Назва етапу дипломного проєкту	Строк виконання етапу проєкту	Примітка
1.	*Затвердження теми проєкту*	*14.04.2025*	
2.	*Вивчення та аналіз завдання*	*14.04.2025 – 21.04.2025*	
3.	*Розробка архітектури та загальної структури системи*	*21.04.2025 – 05.05.2025*	
4.	*Розробка структур окремих підсистем*	*05.05.2025 – 12.05.2025*	
5.	*Програмна реалізація системи*	*12.05.2025 – 18.05.2025*	
6.	*Оформлення пояснювальної записки*	*18.05.2025 – 28.05.2025*	
7.	*Захист програмного продукту*	*28.05.2025*	
8.	*Передзахист*	*02.06.2025*	
9.	*Захист*	*16.06.2025*	

Студент _____
(підпис)

Керівник проєкту _____
(підпис)

АНОТАЦІЯ

Розроблено програмно-апаратний комплекс для виконання векторно-матричних обчислень. Актуальність роботи зумовлена зростанням обсягів обробки даних у технічних і наукових задачах, що потребують ефективного розв’язання векторно-матричних виразів.

У процесі роботи проведено аналіз існуючих програмних і апаратних засобів для виконання матричних обчислень, обґрунтовано вибір програмного середовища (Java з використанням фреймворку ForkJoinPool та графічного інтерфейсу на платформі JavaFX), а також апаратної платформи з використанням багатоядерного процесору. Спроєктовано архітектуру системи, реалізовано ключові алгоритми, проведено тестування на різних розмірах вхідних даних та при різних кількостях ядер.

Результатом роботи є створення прототипу програмно-апаратного комплексу, здатного виконувати векторно-матричні обчислення з високою швидкодією, що може бути використано як навчальний інструмент, дослідницький модуль або вбудований компонент у спеціалізованих інформаційних системах.

ANNOTATION

A hardware-software system for performing vector-matrix computations has been developed. The relevance of this work is driven by the increasing volume of data processing in technical and scientific tasks that require efficient evaluation of vector-matrix expressions.

During the project, an analysis of existing software and hardware tools for matrix computations was conducted. The choice of the software environment (Java using the ForkJoinPool framework and the JavaFX graphical interface) and the hardware platform (utilizing a multi-core processor) was substantiated. The system architecture was designed, key algorithms were implemented, and testing was carried out with varying input data sizes and different numbers of processor cores.

The result of the work is a prototype of a hardware-software system capable of performing high-speed vector-matrix computations. It can be used as an educational tool, a research module, or an embedded component in specialized information systems.

Довідки	Формат	Значення	Найменування	Кіл. листів	№ екземп-ляра	Додаток
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.466500.002 ТЗ***	Технічне завдання	3		
	А4	***ІАЛЦ.466500.003 ПЗ***	Пояснювальна записка	67		
	А4	***ІАЛЦ.466500.004 Д1***	Структурна схема	1		
			програмно-апаратного			
			комплексу			
			(Структурна схема)			
	А4	***ІАЛЦ.466500.005 Д2***	Схема процесора	1		
			програмно-апаратного			
			комплексу			
			(Функціональна схема)			
	А4	***ІАЛЦ.466500.006 Д3***	Блок-схеми алгоритмів	2		
			(Принципова схема)			
	А4	***ІАЛЦ.466500.007 Д4***	Лістинг програми	21		

Зм	*Арк.*	*№ докум*	*Підп.*	*Дата*	
Розроб		*Маруженко І.С.*			
Перев		*Корочкін О.В.*			

ІАЛЦ.466500.001 ОА

Програмно-апаратний комплекс для векторно-матричних

Опис альбому

Літ.	Арк.	Аркушів
	1	1

НТУУ «КПІ» ФІОТ
ІО-13

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Програмно-апаратний комплекс для векторно-матричних обчислень*»

Київ – 2025

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання включає в себе розробку програмно-апаратного комплексу для векторно-матричних обчислень.

Область застосування: проведення векторно-матричних обчислень для заданих виразів з великим об'ємом даних.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки дипломного проєкту є завдання на розробку програмно-апаратного комплексу для векторно-матричних обчислень, з метою виконання бакалаврського проєкту за освітньо-професійною програмою «Комп'ютерні системи та мережі» спеціальності 123 «Комп'ютерна інженерія», що затверджений факультетом «Інформатики та обчислювальної техніки» Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням розробки даного проєкту є створення програмно-апаратного комплексу для вирішення заданих векторно-матричних виразів з великим об'ємом даних.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки для даного дипломного проєкту слугують технічна документація, науково-технічна література, наукові статті та публікації в мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Вимогами до розробленого продукту в ході виконання роботи є:

- Забезпечення високої швидкості виконання векторно-матричних обчислень.
- Мінімізація часу обробки виразів із великими обсягами даних.
- Оптимальне використання системних ресурсів.
- Забезпечення ефективної взаємодії між програмною та апаратними частинами.

5.2. Вимоги до програмного забезпечення

- Операційна система Windows 10 і вище.
- Java Development Kit 24 і вище.

5.3. Вимоги до апаратної частини

- Багатоядерний процесор із підтримкою інструкцій паралельної обробки даних.
- Об’єм оперативної пам’яті від 8ГБ.
- SSD-накопичувач.

6. ЕТАПИ РОЗРОБКИ

Назва етапу дипломного проєкту	Термін виконання
Затвердження теми проєкту	*14.04.2025*
Вивчення та аналіз завдання	*14.04.2025 – 21.04.2025*
Розробка апаратної частини	*21.04.2025 – 05.05.2025*
Розробка програмної частини	*05.05.2025 – 18.05.2025*
Оформлення пояснювальної записки	*18.05.2025 – 28.05.2025*

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Програмно-апаратний комплекс для векторно-матричних обчислень*»

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЦП – Центральний процесор (можна писати як ЦПУ або CPU)

ГП – Графічний процесор (ГПУ або GPU)

ПЗ – Програмне забезпечення

ПА – Програмно-апаратний

ПАК – Програмно-апаратний комплекс

JDK – Java Development Kit

JIT – Just-In-Time компілятор

API – Application Programming Interface (інтерфейс взаємодії)

GUI – Graphical User Interface (графічний інтерфейс)

RAM – Random Access Memory (оперативна пам’ять)

TXT – Формати файлів

ООП – Об’єктно-орієнтоване програмування

ОС – Операційна система

RDIMM – Registered Dual Inline Memory Module

SMT – Simultaneous Multithreading

ВСТУП

Поява персональних комп’ютерів і мережі Інтернет у другій половині XX століття стала важливим етапом у розвитку людства, що ознаменував початок інформаційної епохи. В новий період історії головним ресурсом і рушієм розвитку стала інформація. Такі зміни у пріоритетах призвели до стрімкого збільшення об’ємів накопичених даних та подальшого розвитку інформаційних технологій. В свою чергу збільшення кількості інформації створило нові вимоги до її зберігання, структуризації та обробки. Для цих цілей широко застосовуються такі структури даних, як вектори та матриці, а їх ефективна обробка вимагає застосування векторно-матричних обчислень. Такі обчислення є критично важливими для обробки інформації в багатьох галузях науки, техніки та бізнесу, адже однією з вимог сучасності є необхідність оптимізації складних операцій над даними для збільшення загальної продуктивності.

Векторно-матричні обчислення добре масштабуються завдяки можливості ефективного розпаралелювання, що дає змогу істотно підвищити швидкодію при використанні багатоядерних процесорів і багатопотокових обчислювальних систем.

Наведена аргументація підкреслює актуальність розробки програмно-апаратного комплексу для прискорення векторно-матричних обчислень. В цьому проєкті буде розглянуто типи комп’ютерних систем, проведено аналіз сучасних підходів до розробки апаратних та програмних складових. Вимогами до ПАК являється забезпечення належної обчислювальної потужності для роботи з великими обсягами інформації та мінімізація часу, необхідного для її обробки. Архітектура ПАК повинна забезпечувати можливість майбутніх модифікацій та покращень, враховуючи швидке зростання вимог до подібних систем.

РОЗДІЛ 1.
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ПОБУДОВИ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

1.1 Апаратні складові ПАК

1.1.1 Типи комп`ютерних систем

У сучасних інформаційних технологіях комп’ютерні системи класифікуються за різними ознаками: рівнем продуктивності, структурою обробки даних, масштабами використання, а також характером виконуваних завдань. Розуміння функціонального поділу комп’ютерних систем дозволяє ефективно обирати апаратну основу для конкретних задач, зокрема тих, що вимагають інтенсивного використання ресурсів процесора, пам’яті або паралельної обробки даних[1].

Поширеним типом є персональні комп’ютери, які становлять основу для повсякденного користування в офісних, навчальних, творчих і домашніх середовищах. Типова конфігурація сучасного ПК включає багатоядерний процесор (від 4 до 16 ядер), 8–64 ГБ оперативної пам’яті DDR4 або DDR5, SSD-накопичувач для прискореного доступу до даних і інтегрований або дискретний графічний адаптер. Персональні комп’ютери використовують архітектуру типу x86-64 та підтримують широкий спектр операційних систем (Windows, Linux, macOS), що робить їх гнучким інструментом для базових та середніх обчислювальних потреб.

Наступним класом, що суттєво перевершує ПК за ресурсами, є робочі станції. Це високопродуктивні комп’ютери, які призначені для виконання спеціалізованих, ресурсоємних задач, зокрема моделювання, обробки тривимірної графіки, симуляцій у технічних та наукових середовищах. Такі системи оснащуються потужними багатоядерними процесорами (наприклад,

Intel Xeon або AMD Threadripper), мають обсяг оперативної пам’яті до 1 ТБ, та професійні графічні прискорювачі класу NVIDIA Quadro або AMD Radeon Pro. Важливою характеристикою робочих станцій є стабільність, довговічність і підтримка багатозадачності в умовах інтенсивної експлуатації.

Для централізованого обслуговування користувачів і програмних сервісів використовуються серверні системи. На відміну від ПК і робочих станцій, сервери оптимізовані для тривалої безперервної роботи (24/7) та здатні обробляти одночасно велику кількість запитів. У типовій серверній конфігурації застосовуються багатопроцесорні плати з підтримкою 2–4 CPU (кожен з 24–64 ядрами), оперативна пам’ять типу ECC з обсягом понад 256 ГБ, RAID-масиви для підвищення надійності зберігання, а також високошвидкісні мережеві адаптери (10 Гбіт/с і вище). Сервери поділяються за призначенням на файлові, веб-, поштові, баз даних, віртуалізації, і використовуються як у локальних дата-центрах, так і в хмарних середовищах.

Окрему категорію представляють суперкомп’ютери - системи, спроєктовані для виконання найскладніших обчислювальних задач, зокрема моделювання фізичних процесів, біоінформатики, аналізу великих масивів даних. У своїй архітектурі суперкомп’ютери складаються з тисяч вузлів, об’єднаних у паралельну інфраструктуру через високошвидкісні інтерфейси, такі як InfiniBand. Обчислювальна потужність таких систем може перевищувати 100 PFLOPS (10^{17} операцій за секунду). Вузли можуть включати як центральні процесори, так і графічні прискорювачі (наприклад, NVIDIA A100), що дозволяє реалізовувати гібридні моделі обчислень. Прикладом такої архітектури є система Fugaku (Японія) або Summit (США).

Поряд із суперкомп’ютерами тривалий час залишаються актуальними мейнфрейми – великі обчислювальні системи з високим рівнем надійності та стійкості до відмов. Вони розраховані на обробку сотень мільйонів транзакцій на добу, підтримують велику кількість віртуальних машин та модульну архітектуру. Мейнфрейми активно використовуються в банківській сфері,

телекомунікаціях, урядових структурах. Обчислювальні вузли таких систем мають спеціалізовані процесори (наприклад, IBM z15) з апаратною підтримкою шифрування, апаратним контролем пам’яті та логічним розділенням навантажень.

Іншим напрямом є вбудовані (embedded) системи, які реалізуються як частина складніших технічних пристроїв і виконують обмежений, але чітко визначений набір функцій. До таких систем належать мікроконтролери, плати з ARM-процесорами (наприклад, STM32, Raspberry Pi), що забезпечують обробку сенсорних сигналів, керування механізмами, зв’язок між пристроями. Частота таких процесорів зазвичай становить 100–1500 МГц, обсяг пам’яті - від кількох сотень КБ до кількох ГБ. Особливістю є висока надійність, енергоефективність та робота в режимі реального часу.

Поширеним сьогодні рішенням у корпоративному та споживчому секторі стали хмарні обчислювальні системи, які базуються на віртуалізованій інфраструктурі, розміщеній у віддалених дата-центрах. Користувачі отримують доступ до обчислювальних ресурсів через інтернет за моделлю IaaS (інфраструктура як сервіс), PaaS (платформа як сервіс) або SaaS (програмне забезпечення як сервіс). Хмарні обчислення використовують автоматичне масштабування, балансування навантаження, захищене резервне зберігання даних та інтеграцію з сервісами обробки штучного інтелекту. Типовими провайдерами є Amazon Web Services, Microsoft Azure, Google Cloud.

У деяких задачах критичним є використання розподілених систем, у яких обробка інформації виконується не на одній фізичній машині, а одночасно на десятках чи сотнях вузлів. У цьому випадку завдання поділяється на підзадачі, що обробляються паралельно, з подальшою агрегацією результатів. Розподілені обчислення застосовуються в таких платформах, як Apache Hadoop, Spark. Основні переваги – масштабованість, гнучкість, підвищена відмовостійкість[3].

Особливе місце посідають мобільні комп’ютерні системи, до яких належать смартфони, планшети, носимі пристрої та спеціалізовані мобільні контролери. Завдяки поєднанню компактності та енергоефективності, мобільні системи оснащуються процесорами ARM-архітектури (наприклад, Apple M-серія або Qualcomm Snapdragon), оперативною пам’яттю LPDDR4/LPDDR5, графічними блоками, модулями GPS, Wi-Fi та Bluetooth. Вони забезпечують обробку мультимедіа, навігацію, зв’язок і підтримку програмного забезпечення на базі Android або iOS.

Останнім типом, що активно досліджується, є квантові комп’ютери, які використовують принципи квантової механіки для виконання обчислень. Основною одиницею інформації є кубіт - квантовий аналог біта, який може перебувати у стані 0, 1 або суперпозиції. Квантові комп’ютери (IBM Q System One, D-Wave Advantage) виконують обчислення за допомогою квантових вентилів, потребують наднизьких температур (до 0,015 К) і складних систем стабілізації. Незважаючи на експериментальний характер, подібні системи потенційно здатні прискорити вирішення задач пошуку, оптимізації та моделювання хімічних процесів у десятки або сотні разів порівняно з класичними обчисленнями.

1.1.2 Багатоядерні процесори

У сучасних комп’ютерних системах центральні процесори (CPU) відіграють ключову роль у забезпеченні продуктивності, особливо у задачах, які вимагають обробки великих обсягів даних або високої швидкодії при багатозадачності. Однією з найважливіших еволюційних змін у розвитку процесорів стало впровадження багатоядерної архітектури. Якщо раніше процесор містив лише одне ядро і міг виконувати лише один потік інструкцій одночасно, то тепер багатоядерні процесори мають два, чотири, вісім або навіть кілька десятків ядер, кожне з яких є повноцінним обчислювальним блоком.

Багатоядерний процесор – це мікросхема, що містить кілька незалежних ядер, які можуть одночасно обробляти інструкції. Це дозволяє розподіляти навантаження між ядрами, підвищувати ефективність виконання паралельних алгоритмів і забезпечувати кращу реакцію системи при багатозадачній роботі. У більшості сучасних операційних систем підтримується багатопотокове програмування, що забезпечує оптимальне використання багатоядерних ресурсів.

Кожне ядро має власний блок виконання інструкцій, кеш-пам’ять першого та другого рівнів, а також спільний або розподілений кеш третього рівня (L3). У багатьох сучасних процесорах реалізовано підтримку технології Simultaneous Multithreading (SMT) або Hyper-Threading (у Intel), що дозволяє одному ядру одночасно обробляти два потоки, імітуючи два логічних ядра.

Загалом, збільшення кількості ядер не гарантує лінійного приросту продуктивності, однак у задачах, що добре паралелізуються (моделювання, рендеринг, кодування відео, обчислення в науці та техніці), виграш може бути дуже значним. З іншого боку, системи з меншою кількістю ядер, але вищою тактовою частотою, можуть краще показати себе у задачах з послідовними інструкціями. В таблиці 1.1 наведено порівняння двох актуальних моделей багатоядерних процесорів: AMD Ryzen Threadripper PRO 3995WX[2, 13, 14] та Intel Core i9-13900K[4], які належать до різних класів і орієнтовані на різні сфери застосування.

Таблиця 1.1 - Порівняльна таблиця багатоядерних процесорів

Характеристика	**AMD Ryzen Threadripper PRO 3995WX**	**Intel Core i9-13900K**
Архітектура	Zen 2	Raptor Lake (гібридна: P- та E-ядра)
Техпроцес	7 нм TSMC	Intel 7 (10 нм)
Кількість ядер	64	24 (8 продуктивних + 16 енергоефективних)

Кінець таблиці 1.1

Характеристика	**AMD Ryzen Threadripper PRO 3995WX**	**Intel Core i9-13900K**
Кількість потоків	128	32
Базова частота	2.7 ГГц	3.0 ГГц
Максимальна частота Boost	До 4.2 ГГц	До 5.8 ГГц
Кеш L3	256 МБ	36 МБ
Типова потужність (TDP)	280 Вт	125 Вт (до 253 Вт у режимі Turbo)
Підтримка оперативної пам’яті	DDR4 ECC до 2 ТБ	DDR5/DDR4 до 128 ГБ
Роз’єм CPU	sWRX8	LGA1700
Призначення	Високопродуктивні обчислення, робочі станції	Потужні ПК, геймінг, креативні завдання
Рік випуску	2020	2022

Процесор AMD Ryzen Threadripper PRO 3995WX – це рішення з сегменту високопродуктивних робочих станцій, орієнтоване на задачі промислового моделювання, візуалізації, рендерингу та інженерних обчислень. Його 64 фізичні ядра та 128 потоків забезпечують виняткову паралельну продуктивність у задачах, де важлива багатопоточність. Він підтримує до 2 ТБ оперативної пам’яті ECC, що робить його придатним для найскладніших сценаріїв, включаючи віртуалізацію та обробку великих даних.

У свою чергу, Intel Core i9-13900K - це високопродуктивний процесор споживчого класу, який поєднує гібридну архітектуру з 8 продуктивними (P) та 16 енергоефективними (E) ядрами. Такий підхід дозволяє збалансовано розподіляти навантаження між фоновими процесами та важкими обчисленнями. Його тактова частота до 5.8 ГГц робить його ідеальним для ігор, роботи зі звуком і відео, а також багатозадачних креативних проєктів.

Незважаючи на меншу кількість ядер, ніж у Threadripper, цей процесор часто демонструє вищу продуктивність в одноядерних та частково багатопоточних задачах.

Обидві моделі показують потенціал багатоядерної архітектури, але в абсолютно різних сегментах. Threadripper є прикладом максимальної паралельної потужності, тоді як Core i9 – демонстрація балансу між продуктивністю, частотою та енергоефективністю.

1.1.3 Кластерні системи

У сфері високопродуктивних обчислень, коли можливості навіть найсучасніших багатоядерних процесорів не дозволяють ефективно вирішувати задачі з великим обчислювальним навантаженням, застосовуються кластерні комп’ютерні системи. Основу таких систем становлять кластерні процесори, що функціонують як частина розподіленої обчислювальної інфраструктури. Їхня головна перевага полягає у здатності масштабовано об'єднувати обчислювальні ресурси великої кількості вузлів для спільного розв’язання єдиної задачі.

Кластерна система складається з окремих обчислювальних вузлів (нодів), кожен з яких має власний багатоядерний процесор, оперативну пам’ять, накопичувач і засоби мережевого з’єднання. Усі вузли з’єднуються між собою через високошвидкісні інтерфейси, зокрема Gigabit Ethernet, InfiniBand або Cray Slingshot, що дозволяє передавати дані з мінімальною затримкою між процесорами в процесі паралельного виконання.

У кластерних середовищах застосовуються серверні процесори з високою кількістю ядер та великою пропускною здатністю пам’яті. Серед найбільш поширених - Intel Xeon Scalable, AMD EPYC, Fujitsu A64FX, а також новітні рішення на базі ARM (наприклад, Amazon Graviton, NVIDIA Grace). Типові технічні характеристики одного кластерного вузла можуть включати:

- Кількість фізичних ядер CPU: 32, 64 або 128 ядер (наприклад, AMD EPYC 9654 має 96 ядер, 192 потоки);
- Кількість потоків: залежить від підтримки SMT/Hyper-Threading, зазвичай подвоєна кількість ядер;
- Обсяг кеш-пам’яті L3: від 64 МБ до 384 МБ (наприклад, AMD EPYC 9654 - 384 МБ L3);
- Частота ядра: базова 2.0–2.5 ГГц, Boost до 3.7–4.0 ГГц;
- Пропускна здатність пам’яті: до 500–800 ГБ/с;
- Тип пам’яті: DDR5/DDR4 ECC, з підтримкою 6–12 каналів на сокет;
- TDP (потужність розсіювання тепла): 200–400 Вт залежно від моделі;
- Підтримка інтерфейсів: PCIe Gen4/Gen5, CXL, NVMe, HBM2 у гібридних рішеннях;
- Мережева інфраструктура: інтеграція з мережами 100/200/400 Гбіт/с.

Наприклад, процесор AMD EPYC 9654 (архітектура Genoa), який використовується в сучасних кластерних системах, має 96 ядер, 192 потоки, виготовлений за 5-нм техпроцесом, оснащений 384 МБ L3 кешу та підтримує 12-канальну DDR5 пам’ять з пропускною здатністю до 460 ГБ/с. Така конфігурація забезпечує надзвичайно високу щільність обчислень при оптимальному енергоспоживанні. При з’єднанні десятків або сотень таких вузлів у єдину кластерну систему досягається пікова продуктивність у сотні петафлопсів.

Функціонування кластерних процесорів забезпечується за допомогою програмних середовищ паралельних обчислень, зокрема MPI (Message Passing Interface), OpenMP, OpenCL, а також засобів управління чергами задач (SLURM, PBS, LSF). Завдяки цьому процесори у вузлах кластеру синхронізуються, розподіляють підзадачі, обмінюються результатами обчислень і забезпечують стійке масштабування незалежно від кількості обчислювальних одиниць.

Кластерні процесори застосовуються у задачах, що потребують масивної обробки даних: чисельне моделювання фізичних систем, біоінформатика, кліматичні симуляції, квантова хімія, розпізнавання зображень, обробка супутникових даних, штучний інтелект. Обчислювальні кластери часто використовуються в національних суперкомп'ютерних центрах, дослідницьких інститутах, урядових установах та великих промислових корпораціях.

До прикладу, кластерна система LUMI, що розташована у Фінляндії, базується на процесорах AMD EPYC 7003 та графічних прискорювачах Instinct MI250X. Вона містить понад 3600 вузлів та забезпечує пікову продуктивність понад 550 PFLOPS, що ставить її серед найпотужніших кластерних систем світу. Обчислювальна мережа об'єднана через інтерфейс Slingshot 11 зі швидкістю понад 200 Гбіт/с на вузол, що забезпечує мінімальні затримки під час передачі даних між процесорами.

У такий спосіб кластерні процесори формують обчислювальний фундамент суперкомп'ютерних систем нового покоління, орієнтованих на рішення задач критичного значення в галузях науки, техніки, медицини, економіки й національної безпеки.

1.2 Програмні складові ПАК

1.2.1 Мова програмування C++

Мова програмування C++ традиційно розглядається як одна з основних у сфері наукових та інженерних обчислень, насамперед завдяки своїй здатності забезпечувати високу продуктивність, точний контроль над пам'яттю та безпосередній доступ до апаратних ресурсів. При створенні обчислювальних систем, зокрема таких, що реалізують математичні моделі у вигляді великих матриць чи складних обчислень, C++ часто сприймається як логічний вибір через наявність ефективних бібліотек (таких як Eigen,

Armadillo, Boost uBlas), які дозволяють реалізовувати лінійну алгебру на низькому рівні з мінімальними накладними витратами.

З погляду багатопоточності C++ також надає розробнику широкий спектр можливостей: починаючи від базових засобів роботи з потоками у стандартній бібліотеці (std::thread, std::mutex, std::lock_guard), до використання високопродуктивних рішень типу OpenMP або Intel TBB, які орієнтовані на паралельну обробку даних. Такий рівень контролю дає змогу гнучко керувати розподілом обчислювального навантаження, адаптуючи алгоритми до особливостей конкретного процесора чи платформи.

Однак саме цей контроль, який на перший погляд виглядає перевагою, у практичній реалізації часто обертається на істотну складність. Реалізація багатопотокових обчислень у C++ передбачає ручне керування синхронізацією, що створює передумови для типових помилок, таких як гонки даних, дедлоки або несподівані стани пам’яті. У великих проєктах, особливо коли йдеться про підтримку й масштабування програмного коду, це створює суттєві виклики. Крім того, відсутність автоматизованих механізмів управління потоками, як і складність тестування паралельного виконання, ускладнюють розробку стабільної та передбачуваної системи, здатної функціонувати на різних апаратних платформах без додаткової адаптації.

Використання C++ у контексті кросплатформенності потребує додаткових зусиль: хоча сама мова є універсальною, практична реалізація проєкту з використанням специфічних бібліотек або API часто вимагає окремої компіляції та налаштування під кожну цільову платформу. Це, в свою чергу, підвищує загальні витрати на підтримку та оновлення коду, що є критичним чинником у проєктах із тривалим життєвим циклом.

C++ є технічно потужною мовою з великим потенціалом у сфері високопродуктивних обчислень, її використання в задачах багатопотокової обробки даних пов’язане з низкою труднощів. Високий поріг входження, потреба в ручному управлінні потоками та складність у тестуванні й підтримці

коду створюють додаткове навантаження на розробника, що не завжди є доцільним у контексті сучасних вимог до гнучкості, масштабованості та стабільності систем. Саме тому, попри теоретичну ефективність, C++ на практиці може поступатися іншим мовам, які забезпечують вищий рівень абстракції та вбудовані механізми безпечної багатопоточності без значних накладних витрат на підтримку.

1.2.2 Мова програмування Java

Java є універсальною, об'єктно-орієнтованою мовою програмування, яка з моменту свого створення була спроєктована з урахуванням принципів багатопоточності, портативності та безпеки. У контексті розробки програмного забезпечення для високопродуктивних обчислень, зокрема розрахунку матриць опору на багатоядерних системах, Java виявляється одним із найбільш збалансованих рішень, що поєднує в собі ефективність, простоту розробки та високу надійність при виконанні паралельних задач.

Однією з ключових переваг Java у сфері багатопоточних обчислень є її вбудована модель потоків, яка підтримується на рівні платформи Java Virtual Machine (JVM). Завдяки цьому Java забезпечує повну ізоляцію потоків один від одного, надаючи розробнику зручні й безпечні механізми керування, серед яких - класи Thread, Runnable, Callable, а також високорівневі API ExecutorService, ForkJoinPool та CompletableFuture.

Особливої уваги заслуговує те, що Java надає готову інфраструктуру для реалізації паралельної обробки великих обсягів даних. Наявність стрім-API (Stream та parallelStream) дає змогу перетворити обробку великих колекцій на ефективний багатоядерний процес без необхідності ручного створення потоків чи синхронізації. Це суттєво знижує поріг входження до багатопотокового програмування та мінімізує ризик типових помилок, властивих низькорівневим мовам. Java забезпечує автоматизоване управління пам’яттю за допомогою вбудованого збирача сміття, що унеможливлює витоки пам’яті

при неправильному управлінні ресурсами, як це часто трапляється в мовах із ручним керуванням пам’яттю.

Крім функціональних переваг, важливою є також стабільність і кросплатформенність рішень, реалізованих на Java. Завдяки JVM, програма, написана один раз, може бути виконана на будь-якій системі, де встановлено віртуальну машину, без потреби в перекомпіляції чи адаптації коду. Це створює унікальні умови для масштабування програмного комплексу, а також для інтеграції з іншими системами та технологіями в межах гетерогенних середовищ.

Не менш важливим аспектом є розвинутість екосистеми Java, зокрема наявність широкого спектру інструментів для тестування, профілювання та моніторингу багатопотокових додатків. Засоби на кшталт Java Flight Recorder, VisualVM, JConsole дозволяють відстежувати навантаження на потоки, виявляти «вузькі місця» в обчислювальному процесі, а також оптимізувати використання ресурсів. У поєднанні з багаторічною стабільністю платформи це забезпечує високу передбачуваність поведінки програмного комплексу навіть під великим навантаженням.

Java поєднує у собі всі необхідні властивості для реалізації складних паралельних обчислень у стабільному, безпечному та масштабованому середовищі. У порівнянні з низькорівневими мовами, такими як C++, Java істотно знижує технічні ризики, полегшує розробку і супровід, а також забезпечує довготривалу підтримку з боку спільноти та інструментів.

1.2.3 Мова програмування Python

Python є однією з найпопулярніших мов програмування у сфері науки, освіти та швидкого прототипування. Завдяки простому синтаксису, багатій екосистемі бібліотек та великій спільноті розробників, Python широко використовується у завданнях машинного навчання, статистичної обробки даних і загальнотехнічних обчислень. Для розробки математичних алгоритмів,

зокрема пов’язаних з лінійною алгеброю, мова пропонує низку потужних інструментів: NumPy, SciPy, SymPy, Pandas тощо. Ці бібліотеки дозволяють виконувати широкий спектр операцій над матрицями, векторизацію обчислень і статистичний аналіз даних без необхідності заглиблення у складності низькорівневого програмування.

Однак, Python має низку фундаментальних обмежень у контексті багатопотокової обробки даних, що істотно впливають на його ефективність у задачах, аналогічних розрахунку матриці опору на багатоядерному процесорі. Основною технічною проблемою є наявність механізму GIL (Global Interpreter Lock), який блокує одночасне виконання декількох потоків Python-коду в одному процесі. Це означає, що навіть при створенні кількох потоків у середовищі Python, фактичне обчислення відбувається лише в одному з них у певний момент часу. Така архітектура унеможливлює повноцінне використання всіх обчислювальних ядер процесора, що суттєво знижує продуктивність при виконанні CPU-інтенсивних задач.

Для часткового обходу цього обмеження Python пропонує використання багатопроцесної обробки за допомогою модуля multiprocessing, який створює окремі процеси з власними інтерпретаторами. Хоча це дозволяє розподіляти обчислення між ядрами, така модель пов’язана з підвищеними витратами на міжпроцесну комунікацію, ускладненням обміну даними та втратою частини гнучкості, яку надають потокові моделі інших мов, зокрема Java. Крім того, робота з великими масивами даних у такій конфігурації потребує обережного проектування, оскільки серіалізація, передача структур між процесами й контроль пам’яті часто знижують загальну ефективність.

Ще одним фактором, який обмежує застосування Python у високопродуктивних обчисленнях, є його інтерпретована природа. Хоча сучасні бібліотеки активно використовують C/Fortran-розширення, сам Python-код, особливо без додаткової оптимізації (через Numba, Cython, PyPy),

виконується повільніше за аналогічний код, написаний на компільованих мовах.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі дипломного проєкту проведено всебічний аналіз існуючих апаратних і програмних рішень, що застосовуються у програмно-апаратних комплексах для векторно-матричних обчислень. Окрема увага приділена сучасним типам комп'ютерних систем – від персональних комп'ютерів до кластерних і суперкомп'ютерних архітектур – із розкриттям їх ролі та можливостей у виконанні паралельних обчислень.

Розглянуто особливості багатоядерних та кластерних процесорів як ключові елементи для апаратного прискорення. Проаналізовано низку мов програмування (C++, Java, Python, Rust) у контексті їх ефективності, багатопоточності, кросплатформенності та можливостей реалізації чисельної лінійної алгебри. Обґрунтовано переваги використання мови Java з огляду на її стабільність, потужну екосистему та зручну багатопоточну модель.

На підставі проведеного порівняльного аналізу сформульовано підхід до реалізації ПАК на основі Java з використанням багатоядерного процесору, що дозволяє досягти балансу між високою продуктивністю, зручністю розробки та масштабованістю. Такий підхід забезпечує ефективну реалізацію векторно-матричних обчислень у межах навчального, дослідницького чи прикладного застосування.

РОЗДІЛ 2.
РОЗРОБКА АПАРАТНОЇ СКЛАДОВОЇ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

2.1 Основні апаратні складові ПАК

2.1.1 Процесор

У сучасних обчислювальних системах високопродуктивні центральні процесори відіграють ключову роль у реалізації складних інженерних, наукових та мультимедійних задач. Одним із найбільш потужних процесорів, призначених для подібного типу навантаження, є AMD Ryzen Threadripper 7970X – представник новітньої лінійки HEDT (High-End Desktop) від компанії AMD. Цей процесор поєднує в собі велику кількість обчислювальних ядер, підтримку сучасних інтерфейсів та пам’яті нового покоління, що робить його придатним для використання в робочих станціях класу «Workstation» та інших високонавантажених середовищах.[5]

Threadripper 7970X побудований на мікроархітектурі Zen 4 – четвертому поколінні процесорної платформи AMD, яке характеризується оптимізованою схемою обробки інструкцій, зменшенням затримок доступу до пам’яті та збільшеною ефективністю на одиницю енергії. Процесор виготовлений за 5-нм техпроцесом (TSMC FinFET), що дозволило зменшити площу кристала, підвищити щільність транзисторів та знизити загальне енергоспоживання на тлі підвищеної продуктивності.

Загальна конфігурація обчислювальних блоків процесора складається з 32 фізичних ядер, кожне з яких підтримує технологію SMT (Simultaneous Multithreading), що дозволяє паралельно виконувати два потоки інструкцій. У сумі це дає 64 логічні потоки, що значно підвищує ефективність при виконанні багатопотокових додатків – таких як моделювання фізичних процесів,

рендеринг 3D-сцен, відеомонтаж, віртуалізація серверів, компіляція великих об’ємів коду тощо.

Тактова частота – один із ключових параметрів, що визначає швидкодію процесора. Threadripper 7970X працює на базовій частоті 4,0 ГГц, що вже є досить високим значенням. У режимі динамічного прискорення (Boost mode), який активується в залежності від навантаження, температури та енергоспоживання, частота може сягати 5,3 ГГц. Такий показник дозволяє досягати чудової продуктивності навіть у задачах, що слабо паралелізуються і критичні до швидкодії одного ядра (наприклад, деякі CAD/CAE-додатки або обробка великої кількості невеликих транзакцій).

Процесор має три рівні кеш-пам’яті, які відіграють роль буфера між ядрами та основною оперативною пам’яттю:

Кеш першого рівня (L1) – 2 МБ (по 64 КБ на кожне ядро): забезпечує надшвидкий доступ до найбільш часто використовуваних інструкцій та даних;

Кеш другого рівня (L2) – 32 МБ (по 1 МБ на ядро): зберігає локально важливі блоки даних для відповідного ядра;

Кеш третього рівня (L3) – 128 МБ (спільний між групами ядер): дозволяє ядрам ефективно обмінюватися інформацією та зменшує затримки при міжпотоковій взаємодії.

Обсяг кеш-пам’яті L3 є особливо важливим у багатоядерних архітектурах, де швидкий доступ до даних між ядрами може істотно впливати на загальну продуктивність системи.

Ще однією ключовою характеристикою є підтримка оперативної пам’яті стандарту DDR5-5200 у 4-канальному режимі, що дозволяє досягти дуже високої пропускної здатності. Максимально підтримуваний обсяг пам’яті – до 1 ТБ (1024 ГБ) у конфігурації з ECC RDIMM (реєстрова пам’ять з контролем помилок). Це особливо актуально для наукових обчислень, де робота з великими наборами даних вимагає значного буферу з високим ступенем надійності.

Важливою характеристикою для подібного класу процесорів є тепловий пакет (TDP), який у даній моделі становить 350 Вт. Це означає, що процесор потребує високопродуктивної системи охолодження. Найчастіше для таких рішень застосовують рідинні СВО (системи водяного охолодження) або великі баштові кулери з декількома тепловими трубками і вентиляторами. Правильне охолодження забезпечує стабільну роботу процесора в умовах пікового навантаження та дозволяє використовувати можливості автоматичного або ручного розгону (overclocking).

Також Threadripper 7970X сумісний із платформою Socket sTR5 – це новий тип процесорного роз'єму, спеціально розроблений для процесорів Zen 4 Threadripper. В парі з чипсетом TRX50, цей процесор підтримує до 48 ліній PCIe 5.0, що відкриває широкі можливості для підключення високошвидкісних відеокарт, накопичувачів NVMe, твердотільних RAID-масивів, а також мережевих контролерів із пропускною здатністю до 400 Гбіт/с.

Окрім технічних характеристик, процесор AMD Ryzen Threadripper 7970X вирізняється ще й набором інноваційних технологій, які значною мірою визначають його обчислювальні можливості. Вони охоплюють як внутрішню мікроархітектуру, так і системи управління продуктивністю, пам'яттю, енергоспоживанням, безпекою та взаємодією з іншими елементами комп'ютерної системи. Важливою складовою архітектури даного процесора є платформа Zen 4, що стала еволюційним розвитком попередніх поколінь Zen і втілила в собі як кількісні, так і якісні зміни. Покращення охоплюють практично всі основні підсистеми ядра: кешування, передбачення розгалужень, обробку інструкцій та управління енергоспоживанням. Зокрема, в Zen 4 було збільшено глибину мікрооперацийного кешу, розширено буфери попередньої обробки інструкцій та суттєво оптимізовано швидкодію декодера. Це дозволило досягти зростання показника IPC – кількості інструкцій, що

виконуються за один такт – приблизно на 13–19%, що забезпечує відчутне пришвидшення навіть у задачах, які не масштабуються багатопотоково.

Важливу роль у реалізації обчислювальної потужності процесора відіграє технологія SMT (Simultaneous Multithreading), яка дає змогу кожному фізичному ядру одночасно обробляти два незалежних потоки інструкцій. Таким чином, процесор із 32 фізичними ядрами забезпечує загальну кількість у 64 логічних потоки. Це створює можливість одночасної обробки великої кількості паралельних задач, що особливо цінно в умовах інтенсивної експлуатації, наприклад, при компіляції коду, рендерингу або моделюванні складних технічних систем. Завдяки такій архітектурі досягається оптимальне використання ресурсів ядра, мінімізується простоювання обчислювальних блоків, а також покращується ефективність при виконанні багатозадачних сценаріїв.

Значним удосконаленням у Zen 4 стало запровадження системи динамічного управління частотами – Precision Boost другого покоління. Ця технологія дає змогу процесору автоматично підвищувати частоту кожного ядра в залежності від теплового навантаження, поточного енергоспоживання та кількості задіяних ядер. Таким чином, при низькому навантаженні процесор працює у звичайному режимі, а при потребі – активує збільшену частоту до 5,3 ГГц, що дозволяє досягти високої продуктивності без ручного розгону. Розширенням цієї технології є Precision Boost Overdrive, яке дозволяє обійти встановлені заводом обмеження на живлення і тепловіддачу, за умови, що система охолодження здатна забезпечити стабільну роботу. Це рішення особливо актуальне для користувачів, які працюють із критично важкими обчисленнями та бажають максимально використовувати потенціал процесора без ризику втрати стабільності.

Високий рівень масштабованості системи забезпечує і так звана архітектура Infinity Fabric – спеціалізована внутрішня шина, що поєднує окремі блоки процесора в єдине обчислювальне середовище. Сам процесор

складається з кількох чиплетів – окремих обчислювальних модулів (CCD) та єдиного контролерного вузла вводу-виводу (IOD). Завдяки Infinity Fabric забезпечується швидкий обмін даними між ядрами, синхронізація кеш-пам'яті та ефективна взаємодія з оперативною пам'яттю та пристроями через шину PCIe. Важливо, що така архітектура не лише підвищує загальну швидкодію, а й дозволяє масштабувати систему при збереженні низьких затримок і високої енергоефективності.

Підтримка оперативної пам'яті DDR5 у чотириканальному режимі з максимальним обсягом до 1 ТБ відкриває нові можливості для задач, пов'язаних із великими масивами даних, зокрема – рендеринг сцени з великою кількістю об'єктів, симуляції фізичних процесів або обробка відеопотоку у високій роздільній здатності. Крім того, використання ECC-пам'яті (Error-Correcting Code) забезпечує апаратне виявлення та корекцію помилок у даних, що передаються в оперативну пам'ять і з неї. Це особливо важливо в наукових, медичних і банківських системах, де навіть одинична помилка може призвести до критичних наслідків.

У контексті взаємодії з периферійними пристроями варто відзначити підтримку інтерфейсу PCI Express 5.0. Він забезпечує подвоєну порівняно з попереднім поколінням пропускну здатність – до 64 ГБ/с для 16-лінійного з'єднання. Загалом у Threadripper 7970X доступно 48 ліній PCIe 5.0, що дозволяє одночасно підключати кілька відеокарт, NVMe-накопичувачів нового покоління, мережевих карт та спеціалізованих прискорювачів обчислень. Завдяки цьому можливо створити збалансовану систему для обробки графіки, машинного навчання або реалізації складної серверної інфраструктури.

Не менш важливим є й апаратне розширення інструкційного набору, зокрема підтримка AVX-512 – набору команд для векторних обчислень, що дозволяє обробляти великі обсяги числових даних паралельно. Це істотно пришвидшує задачі машинного навчання, криптографії, біоінформатики та

обробки зображень. Також підтримуються інструкції FMA3/4 для злиття множення та додавання в межах одного такту, що забезпечує вищу точність і швидкість при реалізації математичних моделей. Додатково реалізовано апаратне прискорення шифрування (AES-NI), що дозволяє значно скоротити час обробки зашифрованих даних.

У сучасних системах зростає потреба у захисті інформації, тому важливою перевагою процесора є підтримка технологій апаратної безпеки. Зокрема, AMD реалізувала інструменти Secure Encrypted Virtualization (SEV), які дозволяють шифрувати пам’ять окремих віртуальних машин, ізолюючи їх одна від одної навіть у межах одного фізичного процесора. Додатково, функція Memory Guard забезпечує шифрування всієї оперативної пам’яті навіть у стані сну чи очікування, що підвищує стійкість до атак на холодному старті або через апаратні експлойти. Платформна підтримка AMD-V дає змогу ефективно запускати віртуальні машини без втрати продуктивності, що критично важливо для центрів обробки даних та хмарних сервісів.

Висока продуктивність сучасних процесорів, таких як AMD Ryzen Threadripper 7970X, досягається не лише завдяки кількості ядер чи частоті, а й через продуману внутрішню структуру, яка дозволяє узгоджено взаємодіяти всім обчислювальним і допоміжним блокам. Щоб краще зрозуміти принципи функціонування цього процесора, необхідно розглянути його логічну схему на рівні структурних компонентів. Це дає можливість оцінити архітектурні рішення, що лежать в основі ефективного розподілу обчислювального навантаження, управління пам’яттю, шинами та внутрішніми інтерфейсами.

Процесор Ryzen Threadripper 7970X реалізує модульну архітектуру, що складається з кількох обчислювальних кристалів (чиплетів) типу CCD (Core Complex Die) та одного IOD (I/O Die) - модуля вводу-виводу. У кожному CCD розміщено по два блоки CCX (Core Complex), а кожен CCX містить до 8 ядер. Таким чином, повноцінна конфігурація з чотирма CCD забезпечує сумарно 32 фізичних ядра. Ядра в межах одного CCX мають доступ до спільного кешу L3,

що покращує взаємодію між потоками даних і пришвидшує обмін інформацією між ядрами.

Ключовим елементом, який забезпечує взаємозв'язок між обчислювальними модулями та контролерами пам'яті, є інтерфейсна шина AMD Infinity Fabric. Саме вона забезпечує синхронну роботу всіх частин процесора: від ядер і кеш-пам'яті до контролерів пам'яті DDR5, шин PCI Express та інтерфейсів управління. Infinity Fabric має високу пропускну здатність і низьку затримку, що дозволяє ефективно масштабувати архітектуру навіть за умов підключення великої кількості периферійних пристроїв.

Окремий блок IOD, виконаний за 6-нм техпроцесом, відповідає за взаємодію з оперативною пам'яттю (до 4 каналів DDR5), шиною PCIe 5.0 (до 48 ліній) та забезпечує реалізацію апаратної віртуалізації, контролю живлення, а також безпекових функцій. Саме завдяки IOD-підсистемі процесор може підтримувати одночасну роботу з високошвидкісними NVMe SSD, відеокартами, мережевими контролерами та іншими високопродуктивними компонентами.

Внутрішня схема також передбачає кілька рівнів кешування. Кеш L1 (на кожному ядрі) забезпечує швидкий доступ до інструкцій та даних. Кеш L2 розміщений також локально на ядрі й використовується для буферизації частих звернень до оперативної пам'яті. Кеш L3 - найбільший і спільний для групи ядер - оптимізує обмін даними між потоками, розташованими на різних ядрах одного CCX або CCD.

Така структурна організація дозволяє обчислювальним блокам функціонувати автономно при мінімальних затратах на міжмодульну синхронізацію. Це робить архітектуру Threadripper особливо ефективною в задачах, де інтенсивно використовуються всі ядра одночасно, зокрема у 3D-візуалізації, компіляції програмного забезпечення, віртуалізації та симуляціях.

2.1.2 Материнська плата

Материнська плата є одною із ключових складових апаратної частини комплексу. У науково-технічній літературі її часто метафорично називають «серцем» всієї системи, адже вона виконую низьку важливих функцій: забезпечую фізичне з’єданання всіх компонетів (процесора, ОЗП, ППЗ, GPU) та обмін даними між ними; розподіляє живлення між іншими складовими, що надходить від блоку живлення.

Оскільки при розобці даного ПАК головним елементом є процесор, то головним з крітерієв для вибору материнської плати є сумісніть із AMD Ryzen Threadripper 7970X. При цьому критерії вдалим поєднанням ціни та якості є материнська плата GIGABYTE TRX50 AERO D. Дана плата має сокет TR5, що забезпечує сумісність із обраним процесором. Підтримує необхідне покоління оперативної пам’яті, а саме DDR5, при чому максимальний об’єм такої пам’яті становить 1TB, що дає можливість серйозного апгрейду в майбутньому. Варто розуміти, що процесор часто буде працювати на максимальній потужності і щоб підтримувати його працездатність необхідне відповідне охолодження. В цьому пункті обрана плата теж має перевагу, адже має систему охолодження із посиленою загальною терморегуляцією, що спрямована на посилене охолодження потенційного гарячих зон, таких як: силові елементи живлення, слоти M.2, чіпсет і тому подібне.[6]

2.1.3 Оперативний запам’ятовуючий пристрій

Невід’ємною частиною роботи програмної складової є оперативно запам’ятовуючий пристрій. ОЗП виступає буфером між процесором і пристроями постійного зберігання даних. Використання ОЗП обмумовлене значно більшою швидкістю роботи такої пам’яті та оперативний доступ процесора до ОЗП через вбудований контролер пам’яті.

В рамках розробки даного ПАК було обрано пам’ять п’ятого покоління від відомого і роками перевіреного виробника Kingston. Для роботи із великими об’ємами даних було обрано чотири планки таких пам’яті із частотою роботи у 4800МГц та об’ємом пам’яті кожної у 32GB. Дана конфігуряція, по-перше, сумісна із вже раніше обраними компонентами, по-друге, забезпечить швидкий доступ процесора до даних і достатній об’єм для зберігання тимчасових даних.[7]

2.1.4 Пристрій постійного зберігання

Наступною важливою складовою є пристрій постійного зберігання даних. Дана складова призначена для довготривалого збереження великого об’єму цифрової інформацію навіть при відсутності живлення. У випадку даного розробляємого програмно-апаратного комплексу на ППЗ будуть зберігатись початкові дані, які за допомогою ПВВ будуть зчитуватись і в подальшому оброблятись системою, а також зберігатимуться результати виконання програми.

Оскільки доступ до початкових і швидкість запису результату напряму впливають на загальну продуктивність комплексу, то обрано швидкий накопичувач, а саме SSD Samsung 990 EVO Plus об’ємом 2 ТВ. Даний ППЗ за рахунок останньої версії високошвидкісної шини PCI-E 5.0, що забезпечую високі показники пропускної здатності. Швидкість для читання інформації становить 7250 МБ/с, для запису 6300 МБ/с. Загальний об’єм у 2ТВ забезпечить безпроблемну роботу комплексу при зберіганні великого об’єму інформації.[12]

2.1.5 Графічний процесор

Графічний процесор або GPU – це графічний блок, що, першочергово розроблений, для обробки графічної інформації. З розвитком інформаційних технологій графічний процесор перестав використовуватися лише для

виведення зображення. Графічні процесори мають сотні тисяч ядер, що надає чудові можливості для використання даного типу процесору у паралельних обчисленнях. В рамках розробляємого програмно-апаратного комплексу GPU буде використовуватись лише для роботи із графічним інтерфейсом тому задля зменшення загального бюджету розробки обраний не самий потужний процесор, а саме: Sapphire Radeon RX 6500 XT 8GB PULSE. Даний графічний процесор передає інформацію за допомгою передостаннього інтерфейсу шини PCI Express 4.0, що забезпечує непогану пропускну здатність, має непогану швидкість роботи у 2855МГц і загальним об'ємом пам'яті у 8 GB .GPU являється чудовим варіантом по відношеню ціна/якість. Даний вибір не позбавляє можливості у майбутньому встановлення більш потужного GPU і використання його потужностей для прискорення обчислень векторно-матричних і інших обчислень.[8]

2.1.6 Система охолодження

Обрані у минулих пунктах компоненти являються доволі потужними, адже здатні швидко оброблювати великі об'єми інформації , що призводить до сильного виділення тепла впродовж виконання операцій, що доволі негативно виплаває на стабільність продуктивність і взагалі може вивести комплекс із лади при перегріванні. Тому без потужної системи охолодження робота ПАК не можлива. Для виконання даної відповідальної роботи обрано систему повітряного охолодження від компанії «Artic». Arctic Freezer 4U-M створений із алюмінію і має вісім мідних трубок, що у поєднанні із кулером, з максимальною швидкістю 2300 обертів у хвилину, забезпечує розсіювану потужність у 350Вт, що і необхідно для забезпечення охолодження розробляємого ПАК.[9]

2.1.7 Блок живлення

Ще одним із критичних важливих компонентів є блок живлення. Задача даного блоку перетворювати електричну енергію змінного струму у стабілізовану напругу постійного струму. Блок живлення не тільки забезпечує необхідне забезпечення всіє системи енергією, але й виконує захисну функцію, оскільки забезпечує стабільну напругу без «просадок» при навантаженні, включає в себе захист від перенапруги, перегрівів, короткого замикання і так далі.

Для виконання описаних вище завдань обрано блок живлення Chieftec POLARIS 3.0 850W. Блок забезпечую необхідну для роботи системи потужність у 850Вт, сертифікований за стандартом «80 PLUS GOLD», що забезпечує понад 90 % ефективності та знижує тепловиділення. Важливим аспектом вибору даного блоку є підтримка PCIe Gen 5 разом із 16-контактним конектором 12VHPWR, що гарантує повну сумісність із обраними у пунктах вище компонентів.[10]

2.1.8 Корпус

Зазвичай важливість корпусу применшують і розглядають його встановлення для виконання естетичних функцій. На справді корпус виконує багато функцій. Найочевидніша фіункція – фізичний захист компонентів: захист від ударів, пилу, механічних пошкоджень, сторонніх предментів, а в деяких випадках за рахунок екраювання – від електромгнітного випромінювання. Не менш важлива функція і можлива голонвна – це організація розміщення компонентів. Правильне розміщення компонентів забезпечує ефективне охолодження, легкий доступ до компонентів при необхідності обслуговування, зручніть у побудові схеми прокладання кабелів.

При розробці ПАК осноним критерієм вибору стала сумісність і можилвість розміщення у корпусі материнської плати та інших компонентів.

Тому корпус обраний від того ж самого виробника, що й виробляє материнську плату, що використовуєть у ПАК. Корпус GIGABYTE C301 GLASS V2 має підтримку материнських плат формату E-ATX, передбачає встановлення до 7 вентиляторів, також надає можливість як горизонатльного так і вертикального встановлення відеокарти. Дана модель за рахунок передньої панелі із великою сіткою та вентильованим верхом надає суттєве зниження температури компонентів за рахунок інтенсивного повітряного потоку.[11]

2.2 Аналіз вартості апаратного забезпечення ПАК

Важливою частиною розробки апаратної частини є аналіз її вартості. Виходячи із комплектуючих, обраних у минулому підрозділі, було побудовано таблицю 2.1 із описом вартості розробленої конфігурації апаратної частини.

Таблиця 2.1 – Опис вартості апаратної частини ПАК

Назва компоненту	Кількість (шт)	Ціна за одиницю (грн)	Загальна вартість (грн)
AMD Ryzen Threadripper 7970X	1	113230	113230
GIGABYTE TRX50 AERO D	1	34723	34723
Kingston 32 GB DDR5 4800 MHz	4	9606	38424
Samsung 990 EVO Plus 2 TB	1	6864	6864
Sapphire Radeon RX 6500 XT 8GB PULSE	1	8726	8726
Arctic Freezer 4U-M	1	2336	2336
Chieftec POLARIS 3.0 850W	1	6170	6170
GIGABYTE C301 GLASS V2	1	4752	4752
Загальна вартість конфігурації (грн)			215225

Загальна вартість отриманої конфігурації – 215225 грн. Така вартість обумовлення високими вимогами до продкутинвості апаратної частини комплексу. Дана вартість є середньою для вартості серверних комп’ютерних систем (зазвичай від 2000$ до 10000$). Вартість конфігурації з часом може збільшуватись при збільшенні кількості оперативної пам’яті, заміни GPU на більш потужний та заміни системи охолодження, до прикладу, на водяне. Можливість подібних покращень закладалась при розробці апаратної частини.

ВИСНОВОК ДО РОЗДІЛУ 2

В ході розробки апаратної складової ПАК особливу увагу було приділено вибору багатоядерного процесору, а саме AMD Ryzen Threadripper 7970x, що є центральним елементом конфігурації і відіграє головну роль у досягненні обчислювальної ефективності. Обраний процесор відноситься до лінійки HEDT-платформи і побудований на архітектурі Zen 4. Має 32 фізичні ядра, максимальна частота роботи яких є 5,3 ГГц. Також за допомогою технології SMT кожне ядро може одночасно оброблювати два потоки, що пришвидшує виконання паралельних задач. Даний процесор має підтримку DDR5-пам’яті у 4-канальному режимі до 1 ТБ.

З огляду на те, що головним елементом конфігурації є багатоядерний процесор, то для всіх інших компонентів головним критерієм відбору була сумісність із AMD Ryzen Threadripper 7970x. Зважаючи на це, в якості материнської плати було обрано GIGABYTE TRX50 AERO D, тому що дана модель має необхідний для встановлення процесору сокет TR5 та посилені системи терморегуляції. Таким же чином було обрано ОЗП, а саме 4 планки Kingstone DDR5 покоління із загальним об’ємом 128 ГБ. Щодо пристрою постійної зберігання, то було обрано високошвидкісний накопичувач SSD від компанії Samsung. Вибір графічного процесору був здійснений з урахуванням сумісності із обраною материнською платою та у виконанні обрахунків участі не приймає. Передбачення можливість покращення цього в майбутньому.

Особливу увагу приділено питанням сумісності, масштабованості та теплових характеристик системи. За цим принципом було обрано систему охолодження Arctic Freezer 4U-M, яка забезпечує потужність розсіювання у 350 Вт. Блок живлення Chieftec POLARIS 3.0 850W, сертифікований за стандартом «80 PLUS GOLD», що гарантує високу працездатність та знижує тепловиділення. Корпус GIGABYTE C301 GLASS V2, що сумісний із обраною материнською платою, передбачає встановлення до 7 вентиляторів для

додаткового охолоджння та особливою структурою корпусу, що надає суттєве зниження температури компонентів за рахунок інтенсивного повітряного потоку.

Зважаючи на вищевказане, у межах розділо було побудовано апаратну платформу, що слугуватиме основою для реалізації програмної складової комплексу для забезпечення ефективного виконання поставленого завдання по прискоренню обрахунків векторно-матричних виразів із великим об’ємом інформації.

РОЗДІЛ 3.
РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМНОЇ СКЛАДОВОЇ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Програмна частина комплексу включає в себе дві програмні складові ПРГ1 та ПРГ2, що представлені векторно-матричними виразами:

1. A = B * (MB * MC) (ПРГ1)
2. MA = MB * (MC * MD) * min(Z) (ПРГ2)

Для розробки ПРГ1 та ПРГ2, для кожної із складової, необхідно розробити паралельний алгоритм обчислення заданого виразу, алгоритм та схему взаємодії потоків при обрахунку виразу та безпосередньо реалізувати алгоритм у обраній мові програмування.

3.1 Розробка ПРГ1

3.1.1 Розробка паралельного алгоритму

ПРГ1 реалізує вираз A = B * (MB * MC), для якого розроблено паралельно математичний алгоритм:

1. $A_H = B*(MB*MC_H)$

3.1.2 Розробка алгоритмів потоків

На основі розробленого паралельно-математичного алгоритму до виразу побудовано алгоритм взаємодії потоків і представлено у виді таблиці. Введення даних виконується у першому потоці, через що побудовано два алгоритми: алгоритм виконання T1 та алгоритм виконання Ti.

Алгоритми взаємодії потоків для ПРГ1:

Таблиця 3.1 – Алгоритм потоку Т1 ПРГ1

№	Задача	Точки синхронізації, КД
1	Введення B, MB, MC	
2	**Сигнал** про введення B, MB, MC потокам ТР	Si.1
3	Обчислення № 1 $A_H = B*(MB*MC_H)$	КД1
4	**Чекати** сигнал задачі Ti про завершення обчислення № 2	Wi.1
5	Виведення результату A	

Таблиця 3.2 – Алгоритм потоку Ti ПРГ1

№	Задача	Точки синхронізації, КД
1	**Чекати** сигнал про введення B, MB, MC потоком T1	W1.1
2	Обчислення № 1 $A_H = B*(MB*MC_H)$	КД1
3	**Сигнал** задачі T1 про завершення обчислення № 2	S1.1

3.1.3 Розробка схеми взаємодії потоків

В розробленому програмно-апаратному комплексі використовується модель із спільною пам’яттю та одним пристроєм введення-виведення інформації, що оброблюється першим потоком. В такому випадку постає необхідність у синхронізації потоків при введені даних та перед їх виведенням. Для цього в програмі використовуються бар’єри B1 та B2. Бар’єр B1 використовується для синхронізації потоків при введені даних. Потоки Ti очікують завершення введення даних у потоці T1, як тільки T1 закінчує введення – він сигналізує іншим потокам про завершення цієї операції і після цього починається обчислення виразу у всіх потоках. Перед виведенням

результатів використовується бар’єр B2, де ситуація протилежна. По закінченню розрахунків потік T1 очікую сигнали від потоків Ti проте, що вони закінчили свою частину обчислення. Після отримання сигналу від всіх потоків – T1 продовжує роботу і записує інформацію на диск. Схема взаємодії потоків наведена на схемі нижче (рис. 3.1):

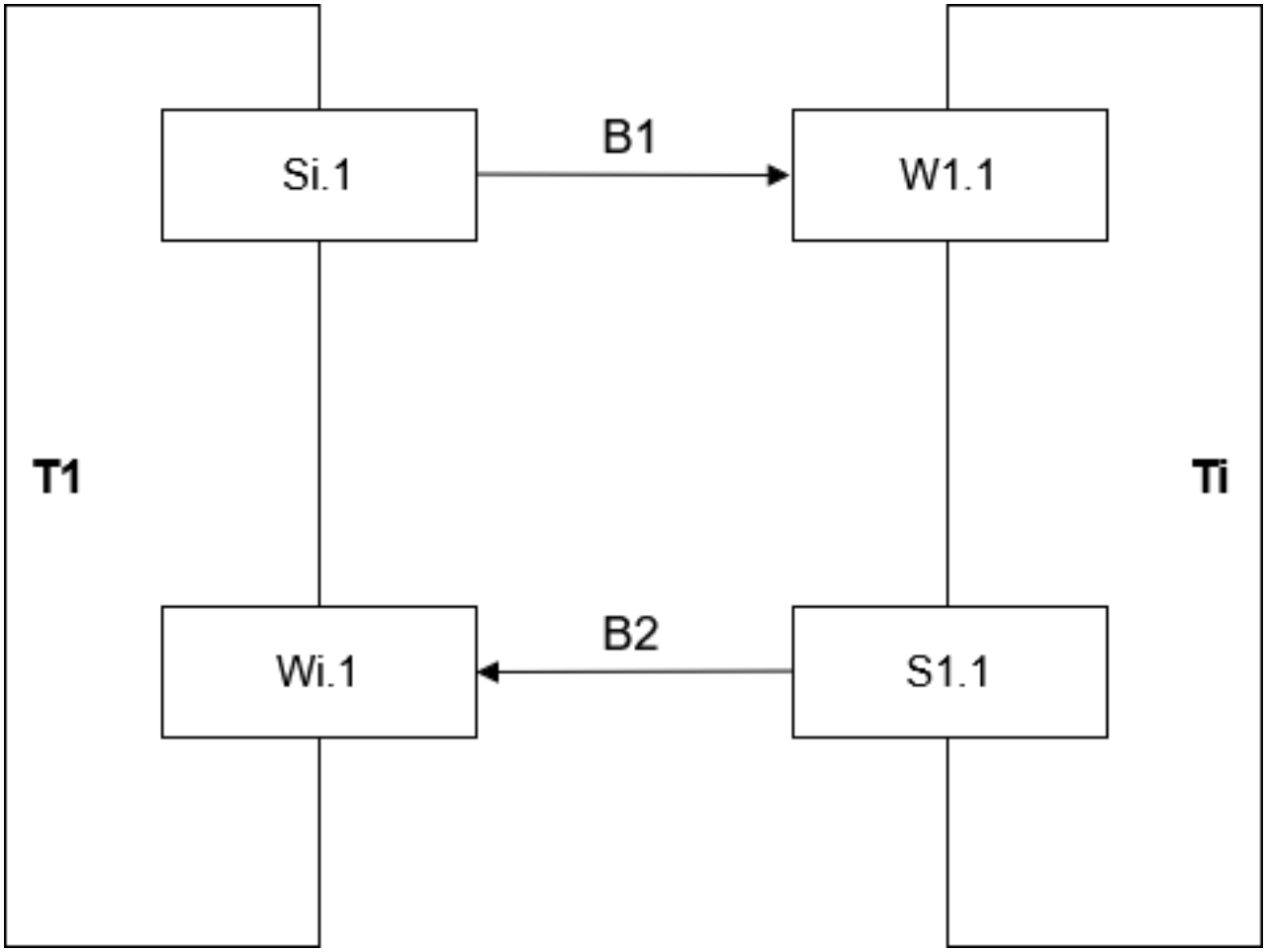


Рисунок 3.1 – Схема взаємодія потоків ПРГ1

3.1.4 Розробка програми

Для розробки програми було обрано мову програмування Java.

Загальна структура проєкту реалізована відповідно до шаблону MVC (Model–View–Controller). Модель представлена обчислювальними модулями, у яких реалізовано математичну логіку:

- подання складається з графічного інтерфейсу, розробленого у форматі FXML із відповідним оформленням;

- контролер забезпечує обробку дій користувача та передачу даних між поданням і моделлю. Дана архітектура дозволяє зберігати чітке розмежування обов'язків між компонентами системи.

У ролі точки входу в застосунок виступає клас Main.java, який відповідає за ініціалізацію графічного середовища. У ньому виконується завантаження головного вікна програми з використанням FXML-файлу (Main.fxml), підключення зовнішніх стилів оформлення та мовних ресурсів. Після запуску застосунок передає керування класу-контролеру.

Ключова логіка взаємодії з інтерфейсом зосереджена у класі MainController.java. У цьому класі реалізовано обробку подій від елементів інтерфейсу, зокрема зчитування даних з полів введення, виклик математичних операцій та виведення результатів. Контролер є посередником між візуальним рівнем програми та обчислювальними модулями, такими як MatrixUtils і VectorUtils, PRG1 та PRG2.

Для досягнення паралелізму використовується фреймворк ForkJoinPool. Алгоритм роботи даного фреймворку є наступним: спочатку проводиться розбиття основної задачі на підзадачі, що додаються в чергу на виконання, вільні потоки обирають підзадачі із черги і виконують їх одночасно, по закінченню виконання задачі потік обирає наступну підзадачу із черги. Вже виконана підзадача очікує на виконання інших задач, тобто очікує на синхронізацію перед виводом результату.

Математичні операції над матрицями реалізовані у класі MatrixUtils.java. До функціоналу цього модуля належать функції: паралельне множення матриць *multiply()*, паралельне множення матриці на скаляр *multiply_z()* та форматування в строку логування *matrixToString()*. Матричні функції можуть використовуватися як самостійно, так і в складі складніших матричних операцій.

Операції з векторами винесено в окремий модуль VectorUtils.java. Модуль містить функції для паралельного визначення мінімального значення

вектора *min_vector()* та паралельне множення матриці на вектор *multiply_matrix_vector()*. Векторні функції можуть використовуватися як самостійно, так і в складі складніших матричних операцій.

Головною функцією ПРГ1 є *prg1()*, що знаходиться у модулі PRG1.java. Дана функція реалізує головну логіку роботи програми, а саме виконує обрахунок виразу ПРГ1. Функція спочатку виконує множення матриць МВ та МС, використовуючи функцію паралельного множення із модулю MatrixUtils.java *multiply()*. Надалі функція виконує множення отриманої матриці в результаті множення МВ та МС на вектор В, використовуючи функцію паралельного множення вектора на матрицю *multiply_matrix_vector()* із модулю VectorUtils.java. По закінченню обчислень Функція виводить проміжні кроки обчислення виразу та кінцеві результати у графічний інтерфейс.

В програмі реалізована можливість збереження отриманих результатів у файл. Після виконання обчислень користувач може натиснути кнопку «Зберегти в TXT», яка знаходиться у контекстному меню «Файл» (рис. 3.2).

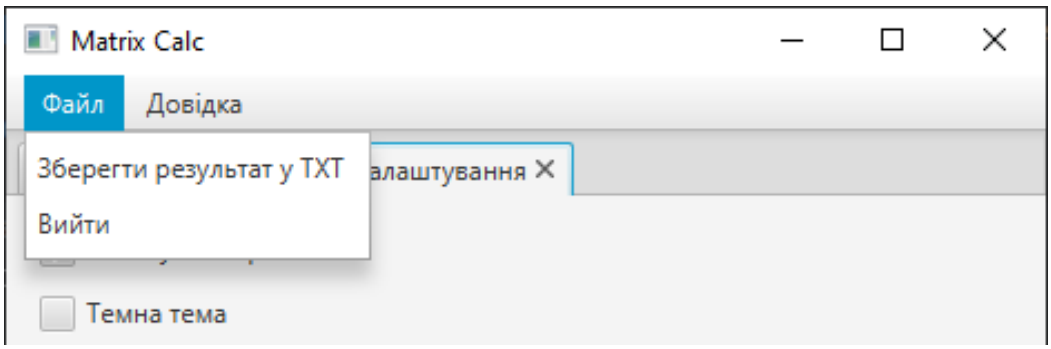


Рисунок 3.2 – Контексте меню «Файл»

Вміст вкладки «Результати», де відображається покрокове пояснення дій автоматично копіюється у текстовий файл. У збереженому документі зберігається вся інформація: задані елементи матриць та векторів, усі перетворення, проміжні значення матриць та кінцеві значення невідомих. Приклад такого текстового файлу можна побачити на рис. 3.3:

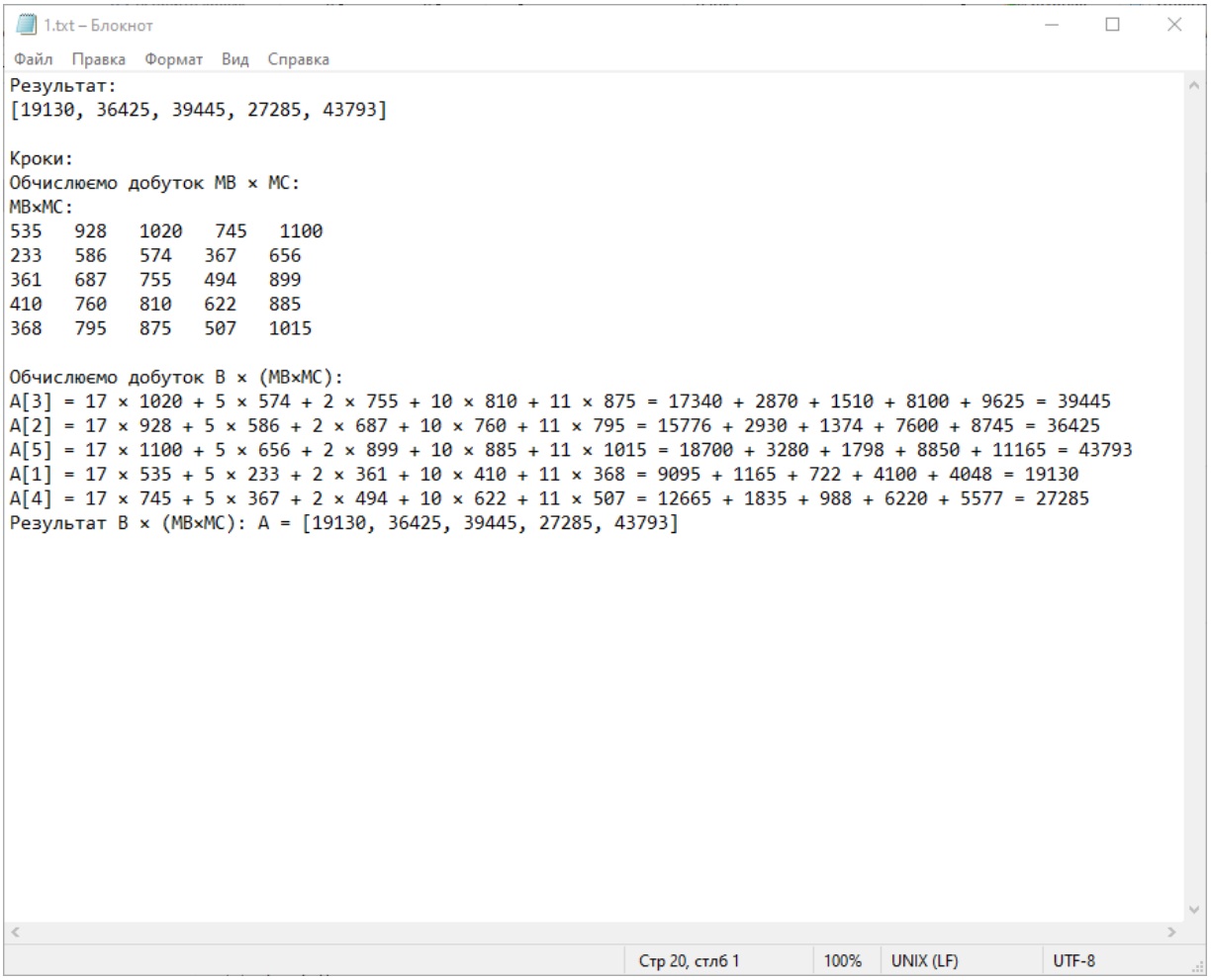


Рисунок 3.3 – Збереження у файл

Окрему функціональну роль у програмі відіграє модуль локалізації та персоналізації. Він реалізований у вигляді ресурсних файлів lang_uk.properties і lang_en.properties, у яких зберігаються мовні відповідники для всіх текстових елементів інтерфейсу. Завдяки цьому користувач має можливість змінювати мову роботи програми. Також у меню налаштувань є можливість включити звукове сповіщення про закінчення розрахунків, увімкнути чи вимкнути вікно підтвердження перед виходом із програми, встановити автоматичне зберігання результатів, вибрати кількість потоків при якій буде працювати програма, змінити розмір шрифту та вибрати темну тему, включити чи виключити відображення кроків обчислення. Вікно налаштувань представлено на рис. 3.4:

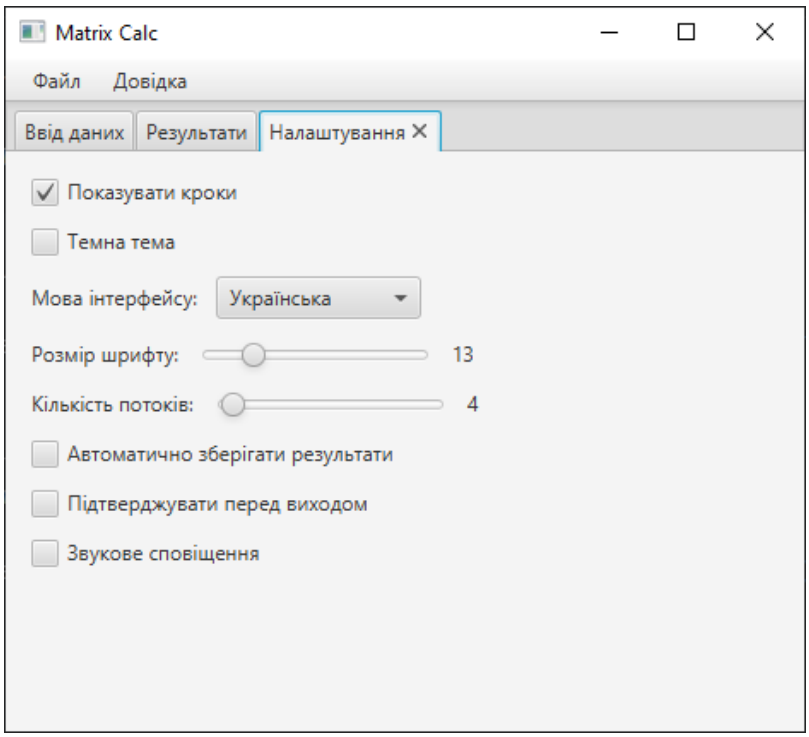


Рисунок 3.4 – Вікно налаштувань програми

3.2 Розробка ПРГ2

3.2.1 Розробка паралельного алгоритму

ПРГ2 реалізований виразом MA = MB * (MC * MD) * min(Z). Для розробки паралельно математичного алгоритму сам вираз розкладено по діях:

1. xi = min(Z_H) i:= 1..P
2. x = min(x, xi) CP: x
3. $MA_H = MB*(MC*MD_H)*x$

В побудованому паралельному алгоритмі у другому пункті з’являється задача по захисту спільного ресурсу. Для вирішення даної задачі обрано засіб захисту від одночасного доступу до спільного ресурсу – атомарна зміна A1.

3.2.2 Розробка алгоритмів потоків

На основі паралельно математичних алгоритмах до виразів побудовано алгоритми взаємодії потоків і представлено у виді таблиць. Введення даних

виконується у першому потоці, тому побудовано по два алгоритму для кожного виразу, а саме алгоритм для T1 та Ti.

Алгоритми взаємодії потоків ПРГ2:

Таблиця 3.3 – Алгоритм потоку T1 ПРГ2

№	Задача	Точки синхронізації, КД
1	Введення MB, MC, MD, Z	
2	**Сигнал** про введення MB, MC, MD, Z потокам TP	Si.1
3	Обчислення № 1 xi = min(Z_H)	
4	Обчислення № 2 x = min(x, xi)	КД1
5	**Сигнал** задачам Ti про завершення обчислення № 2	Si.2
6	**Чекати** сигнал задачі Ti про завершення обчислення № 2	Wi.1
7	Обчислення № 3 MA_H = MB*(MC*MD_H)*x	
8	**Чекати** сигнал задачі Ti про завершення обчислення № 3	Wi.2
9	Виведення результату MA	

Таблиця 3.4 – Алгоритм потоку Ti ПРГ2

№	Задача	Точки синхронізації
1	**Чекати** сигнал про введення MB, MC, MD, Z потоком T1	W1.1
2	Обчислення № 1 xi = min(Z_H)	
3	Обчислення № 2 x = min(x, xi)	КД1
4	**Сигнал** задачі Ti про завершення обчислення № 2	S(i:= 1..i-1).1 S(i:= i+1..P).1

Кінець таблиці 3.4

№	Задача	Точки синхронізації
5	**Чекати** сигналу задачі Ti про завершення обчислення № 2	W1.2 W(i:= 2..i-1).1 W(i:= i+1..P).1
6	Обчислення № 3 $MA_H = MB*(MC*MD_H)*x$	
7	**Сигнал** задачі T1 про завершення обчислення № 3	S1.2

3.2.3 Розробка схеми взаємодії потоків

Аналогічно до ПРГ1 ПРГ2 обрано модель із спільною пам’яттю, введення та виведення даних у першому потоці. Для синхронізації потоків використовуються бар’єри. На відміну від ПРГ1 в даному випадку використовується три бар’єри. Бар’єри B1 та B3 аналогічні до ПРГ1, вони необхідні для синхронізації потоків при введені даних та при синхронізації потоків при виведенні результатів. Третій бар’єр B2 необхідний для синхронізації потоків після пошуку мінімального значення у векторі Z. Після того, як кожен вектор визначить свій локальний мінімум потокам треба порівняти своє значення із спільним. Перед переходом до наступних обчислень треба бути впевненим, що кожен потік порівняв значення і був визначений загальний мінімум, тому виникає необхідність у синхронізації. По завершенню обчислення №2 кожен потік надсилає сигнал всім іншим потокам про завершення обчислення і очікує на сигнали від інших потоків. Як тільки всі потоки відправлять сигнали про закінчення обчислень – бар’єр розблокує потоки і виконання програми продовжиться. Схема взаємодії потоків наведена на схемі нижче (рис. 3.5):

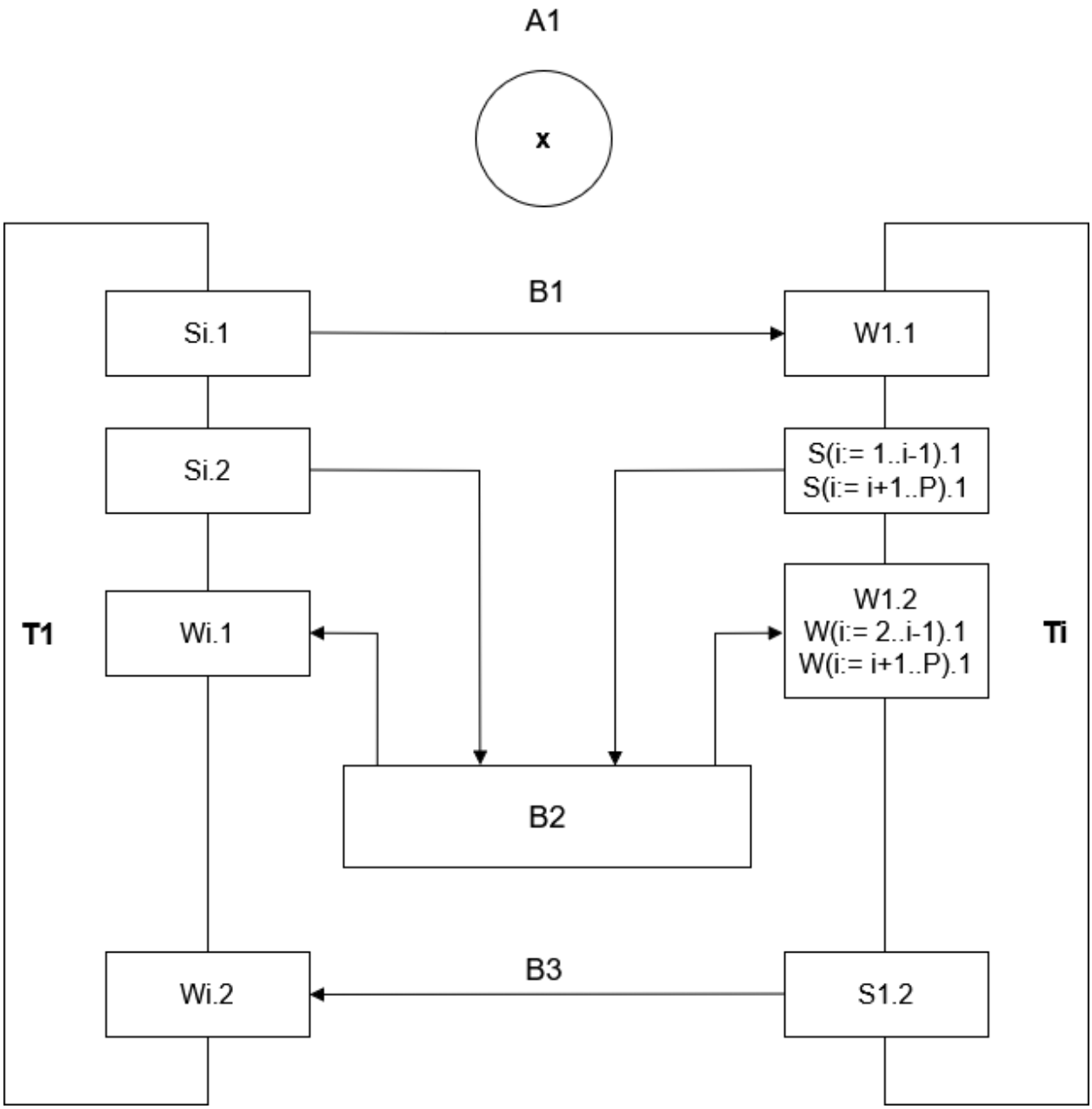


Рисунок 3.5 – Схема взаємодія потоків ПРГ2

3.2.4 Розробка програми

Для ПРГ1 та ПРГ2 створено одну універсальну програму, тому для уникнення дублювання інформації в даному розділі коротко описані головні модулі та відмінність ПРГ1 та ПРГ2. Детальний опис програми наведено у пункті 3.1.4.

Програма реалізована на мові програмування Java з використанням фреймворку ForkJoinPool для реалізації паралельних обчислень.

Аналогічно до ПРГ1 загальна структура проєкту реалізована відповідно до шаблону MVC. Модель представлена обчислювальними модулями, у яких реалізовано математичну логіку:

- подання складається з графічного інтерфейсу, розробленого у форматі FXML із відповідним оформленням;
- контролер забезпечує обробку дій користувача та передачу даних між поданням і моделлю.

Основні модулі:

Main.java – точка входу у застосунок. Відповідає за ініціалізацію графічного середовища. Після запуску застосунок передає керування класу-контролеру.

MainController.java – містить ключову логіку взаємодії з інтерфейсом. У цьому класі реалізовано обробку подій від елементів інтерфейсу, зокрема зчитування даних з полів введення, виклик математичних операцій та виведення результатів. Контролер є посередником між візуальним рівнем програми та обчислювальними модулями, такими як MatrixUtils і VectorUtils, PRG1 та PRG2.

MatrixUtils.java містить математичні операції над матрицями. До функціоналу цього модуля належать функції: паралельне множення матриць multiply(), паралельне множення матриці на скаляр multiply_z() та форматування в строку логування matrixToString(). Матричні функції можуть використовуватися як самостійно, так і в складі складніших матричних операцій.

VectorUtils.java включає в себе операції з векторами. Модуль містить функції для паралельного визначення мінімального значення вектора min_vector() та паралельне множення матриці на вектор multiply_matrix_vector(). Векторні функції можуть використовуватися як самостійно, так і в складі складніших матричних операцій.

Головною функцією ПРГ2 є *prg2()*, що знаходиться у модулі PRG2.java. Дана функція реалізує головну логіку роботи програми, а саме виконує обрахунок виразу ПРГ2. Функція спочатку виконує множення матриць MC та MD, використовуючи функцію паралельного множення із модулю MatrixUtils.java *multiply()*. Наступним кроком функція визначає мінімальне значення вектора Z. Для цього функція використовує представлену у модулі VectorUtils.java функцію для паралельного знаходження мінімального значення вектора *min_vector()*. Надалі функція виконує множення отриманої матриці в результаті множення MC та MD на скаляр x, тобто мінімальне значення вектору Z за допомогою . Для даного обчислення використовується функція *multiply_z()* із модулю MatrixUtils.java. По закінченню обчислень Функція виводить проміжні кроки обчислення виразу та кінцеві результати у графічний інтерфейс.

Аналогічна до ПРГ1, ПРГ2 має вікно із налаштуванням персоналізації, таких як: увімкнення та вимкнення відображення кроків розрахунку виразу у вікні «Результати», зміна теми інтерфейсу, вибрір розміру шрифту, увімкнення та вимкнення автоматичного зберігання результатів обчислень, увімкнення та вимкнення підтвердження перед виходом із програми, увімкнення та вимкнення звукового сповіщння про завершення обрахунків, вибору мови інтерфейсу та вибору кількіості потоків на якій буде працювати програма.

Функція збереження результатів у файл також присутня і є аналогічною до ПРГ1. Користувач має змогу після проведеггя розрахунків зберегти результати виконання виразу і кроки у текстовий документ. У збереженому документі зберігається вся інформація: задані елементи матриць та векторів, усі перетворення, проміжні значення матриць та кінцеві значення невідомих.

3.3 Тестування програмної складової

Варто зауважити, що через неможливість побудувати описаний у роботі програмно-апаратний комплекс з багатьох об’єктивних причин, в першу чергу фінансових, тестування ПРГ1 та ПРГ2 проводиться на персональній робочій станції автора, що є значно слабшою за розроблену апаратну частину для ПАК. Дане тестування, звичайно, не зможе продемонструвати всі переваги розробленого впродовж виконання роботи ПАК, але є достатньою для перевірки програмної складової.

Опис апаратної частини робочої станції наведено в таблиці 3.5.

Таблиця 3.5 – Характеристики робочої станції для проведення тестування

Компонент	**Характеристики**
Процесор (CPU)	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz (4 ядра, 4 потоки)
Материнська плата	HP 8219 83.72
Оперативна пам’ять (RAM)	16 ГБ DDR4 (2x8 ГБ, 2400 МГц)
Накопичувач SSD	Samsung 980 1TB M.2 PCIe 3.0 x4
Дисплей	17.3", IPS, FullHD (1920x1080)
ОС	Windows 10, 22H2
Версія JDK	24.0.1
IDE	IntelliJ IDEA 2023.2.5

В ході тестування буде перевірятися правильність обрахунків, визначатися час витрачений на розрахунки, коефіцієнт прискорення та коефіцієнт ефективності. Алгоритм тестування наступний:

1. Перевірка правильності обрахунків. Проводиться розрахунок виразу при малих розмірностях вручну і порівнюється з результатами, що виводить програма.

2. Визначення часу виконання програми при різній кількості створених потоків. Відповідно до можливостей процесора на наявній робочій станції, тестування буде виконуватись при P=1, P=2, P=4, P=16 та P=32.
3. На основі отриманого часу виконання при різній кількості потоків розраховується коефіцієнт прискорення ($К_П$). Даний коефіцієнт визначає у скільки разів вдалось збільшити швидкість виконання розрахунків відносно збільшення кількості потоків. Для розрахунку коефіцієнту використовується наступна формула:

$$K_{\text{ПН}} = \frac{T1}{TP}, \qquad (3.1)$$

де T – час виконання, P – кількість потоків.

4. На основі отриманих у попередніх пунктах даних розраховується коефіцієнт ефективності ($К_Е$). Даний коефіцієнт визначає на скільки були завантаженні ядра під час обрахунків, тобто на скільки ефективно використовувались ядра. Для визначення даного показника використовується наступна формула:

$$K_E = \frac{K_{\text{П}}}{P} \cdot 100\%, \qquad (3.2)$$

де $К_П$ – коефіцієнт прискорення, P – кількість ядер.

3.3.1 Тестування програми ПРГ1

Відповідно до алгоритму описаного вище, перший етап тестування – це перевірка правильності вирішення програмою виразу. Як вже було вказано раніше, для перевірки необхідно для початку вручну вирішити вираз A = B * (MB * MC). Для зручності обрахунків для вектору B обрано довжину 3 , де кожен його елемент 1. Матриці MB та MC за таким самим принципом зроблено квадратними із розмірностями 3 на 3, де кожен елемент – 1. За таких умов вираз має наступний вигляд:

$$A = B \cdot (MB \cdot MC) = [1 \quad 1 \quad 1] \cdot \left(\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \right) \rightarrow$$

$$\mathrm{A} = [1 \quad 1 \quad 1] \cdot \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix} = [9 \quad 9 \quad 9]$$

Отже, вирішенням даного виразу при заданих раніше значень є вектор із довжиною 3, де кожен елемент є 9.

Для тестування правильності обрахунків в програмі задано аналогічні до ручного розрахунку вхідні дані (рис. 3.6).

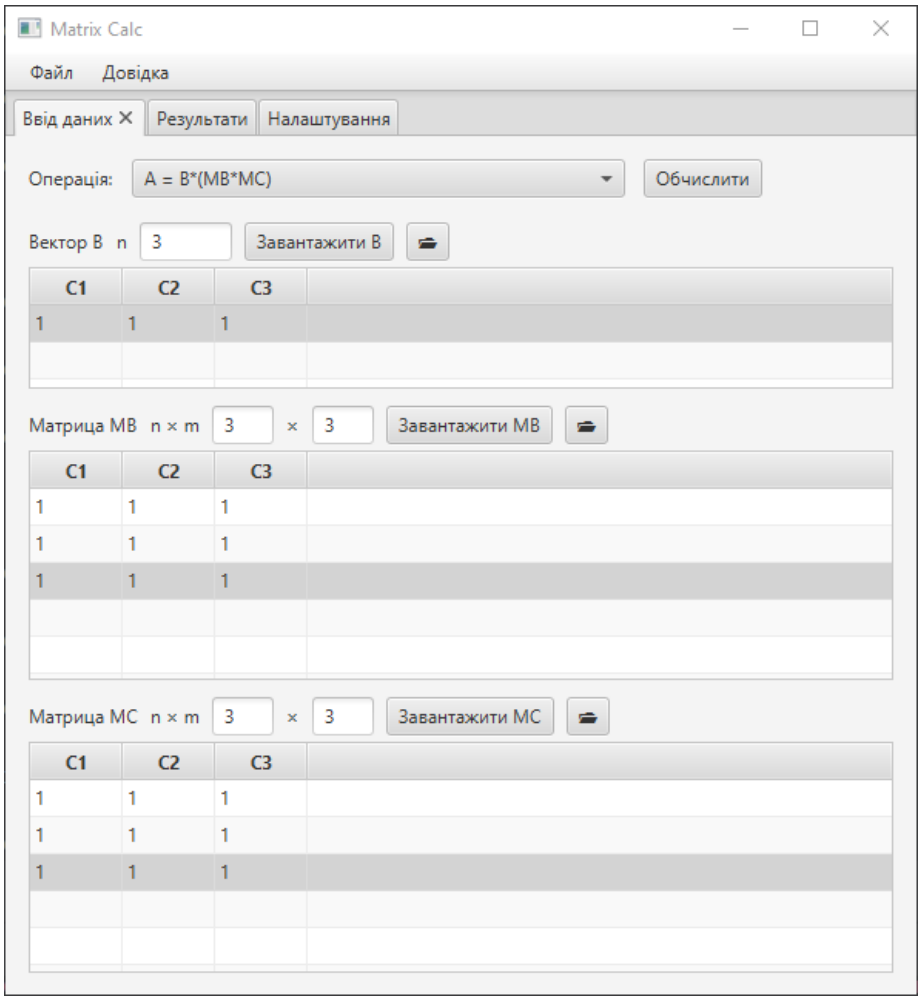


Рисунок 3.6 – Ввід даних для вирішення виразу ПРГ1

Після виконання обчислень програма вивела результати у вікні «Результати». Результати виконання обчислень наведені нижче (рис. 3.7):

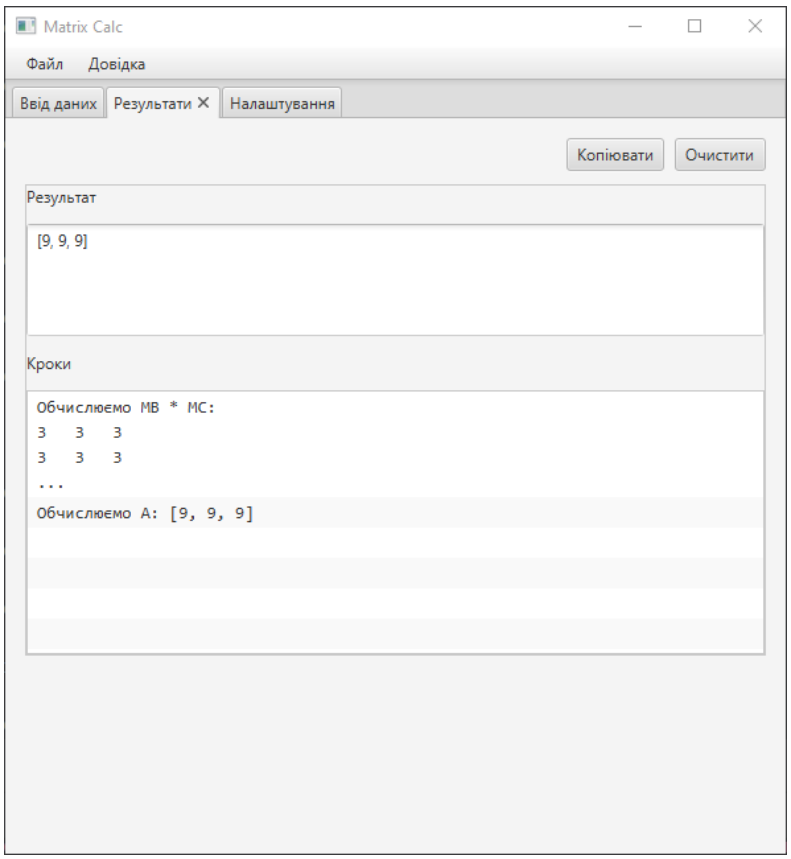


Рисунок 3.7 – Вивід результатів обчислень ПРГ1

Результати ручного обрахунку та обрахунку з використанням програми при роботі на різній кількості ядер співпадають, що підтверджує правильність розрахунків і дає змогу перейти до наступних етапів тестування

Наступним етапом тестування є визначення не менш важливого показника для програмної складової програмно-апаратного комплексу, а саме виміри часу, витрачені програмою для здійснення обрахунків при різній кількості ядер. Даний етап тестування є важливим, адже надає необхідну інформацію для подальшого аналізу результатів, що дозволить оцінити ефективність і правильність прийнятих рішень впродовж розробки апаратної та програмної складових комплексу.

Виміри часу будуть проводитися в три серії. Кожна серія тестування включає в себе три тестування при різній кількості ядер. Розподіл на серії обумовлений різним підходами до підготовки даних до тестування, а точніше різного об’єму даних Таким чином для першої серії розмірність матриць буде становити 512 на 512, довжина вектора, відповідно – 512. В другій серії

розмірності будуть наступними: для матриць – 1024 на 1024, довжина вектору – 1024. Третя і найбільш об’ємна серія: розмірність матриць становить 2048 на 2048, довжина вектору – 2048. Дані для обробки зчитуються з диску перед кожною серією тестувань. Результати тестувань на вимір часу виконання при різній кількості потоків та різних розмірностей записані у таблиці 3.6.

Таблиця 3.6 – Час виконання ПРГ1

N	Час виконання (мс) в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	240	125	66	61	61
1024	2322	1199	579	543	544
2048	104371	53187	31772	27410	27645

На даному етапі тестування важко оцінити чи провести аналіз ефективності розробленої програми. Більше інформації можна отримати шляхом аналізу наступних метрик: коефіцієнт прискорення так коефіцієнт ефективності.

Використовуючи вказану раніше формулу (3.1) для визначення коефіцієнта прискорення та результати із минулого етапу тестування було побудовано таблицю коефіцієнта прискорення в залежності від кількості ядер та об’єму інформації.

Таблиця 3.7 – Коефіцієнт прискорення ПРГ1

N	Коефіцієнт прискорення (рази) в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	1	1,92	3,64	3,93	3,93
1024	1	1,94	4	4,28	4,27
2048	1	1,96	3,28	3,81	3,78

Також отримані результати представлені у вигляді графіку(рис.3.8):

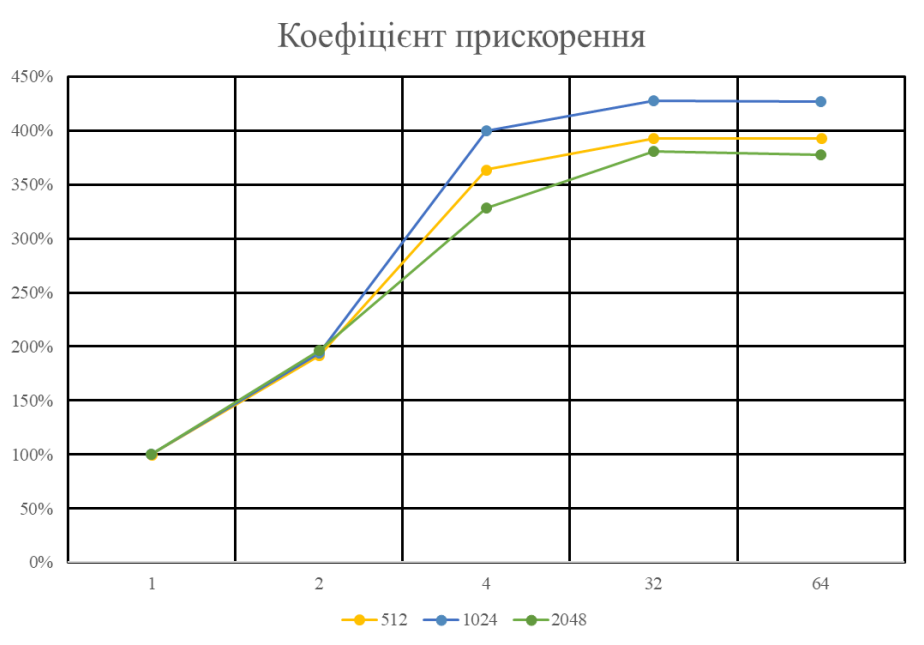


Рисунок 3.8 – Графік коефіцієнту прискорення ПРГ1

Для аналізу отриманих даних необхідно порівняти їх із теоретичними. Теоретичне прискорення обчислень прямо пропорційне збільшенню кількості паралельних потоків виконання. Іншими словами, якщо кількість потоків зростає у певну кількість разів, то й обчислювальна швидкість має зрости у стільки ж разів, за ідеальних умов.

Відмінності теоретичних та практичних даних (див. таблицю 3.7) присутні майже у кожному пункті і на це є пояснення. Теоретичні обчислення виконуються в ідеальній математичній моделі, де не враховується час на створення і синхронізацію потоків, не враховується втрата потужності через нагрівання процесора і багато інших чинників. Враховуючи зазначені зауваження про відмінність теоретичних та практичних даних було проведено аналіз отриманих.

Результати отримані при тестування при різних розмірностях мають схожий характер, тому доцільно аналізувати одразу всі результати акцентуючи уваги на кількості потоків. Таким чином при збільшенні кількості потоків з одного до двох було отримано майже теоретичні значення, а саме: прискорення у 1,92 рази для N=512, прискорення у 1,94 рази для N=1024 та

прискорення у 1,96 рази для N=2048. Ідеального значення у 2 рази не було досягнути через час витрачений на створення та синхронізацію потоків. З іншою сторони на результати тестування позитивно впливало, що програма використовувала половину ядер процесору (рис. 3.9).

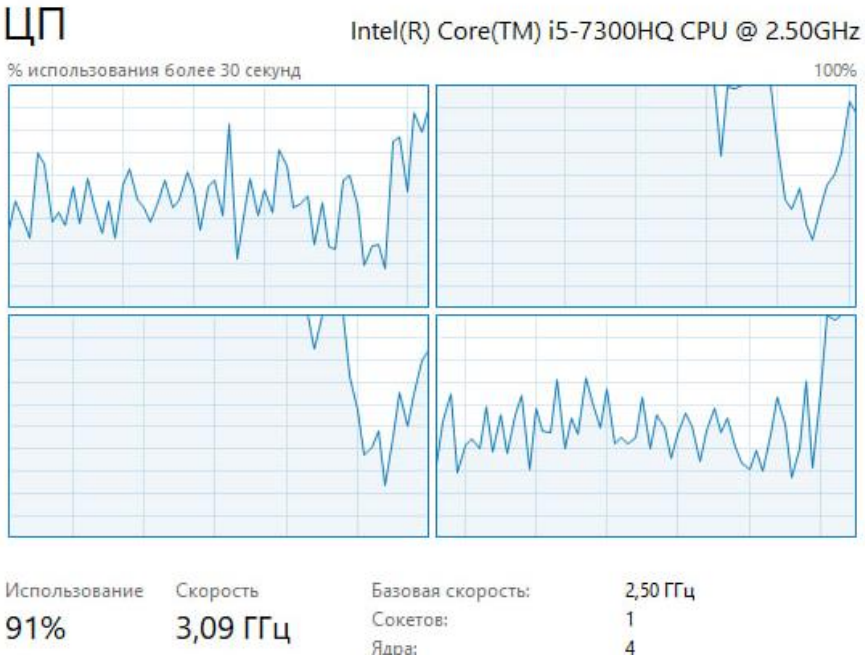


Рисунок 3.9 – Навантаження процесора при виконанні ПРГ1 Т=2

При такому розподілені навантаження використовується приблизно на 50 відсотків потужності процесора. В даному режимі роботи система не зменшує частоту роботи ядер для збереження загальної температури процесора у межах норми. Також при наявності вільних ядер інші процеси операційної системи не заважали програмі виконуватись, адже їх навантаження було перерозподілено по іншим ядрам.

Тестування програми на 4 потоках показало високі показники хоча не такі ідеальні як при тестуванні на двох потоках. Були отримані наступні показники: прискорення у 3,64 рази для N=512, прискорення у 4 рази для N=1024 та прискорення у 3,28 рази для N=2048. Отримання теоретичного показника при N=1024 обумовлене вдалим вибором розмірностей елементів виразу, що створило ідеальне відношення часу витраченого на створення та синхронізацію потоків до часу їх виконання. При розмірностях N=512 та

N=2048 відмінність від теоретичного значення пояснюються апаратними обмеженням робочої станції, на якій проводиться тестування. Негативно на результат вплинуло, те що процесор працював на 100% потужності (рис. 3.10), тому система знизила частоту роботи ядер із 3,3 ГГц до 3 ГГц для уникнення перегріву. Також негативно впливали фонові процеси системи, що під час обрахунку конкурували за ресурс процесора із процесами програми.

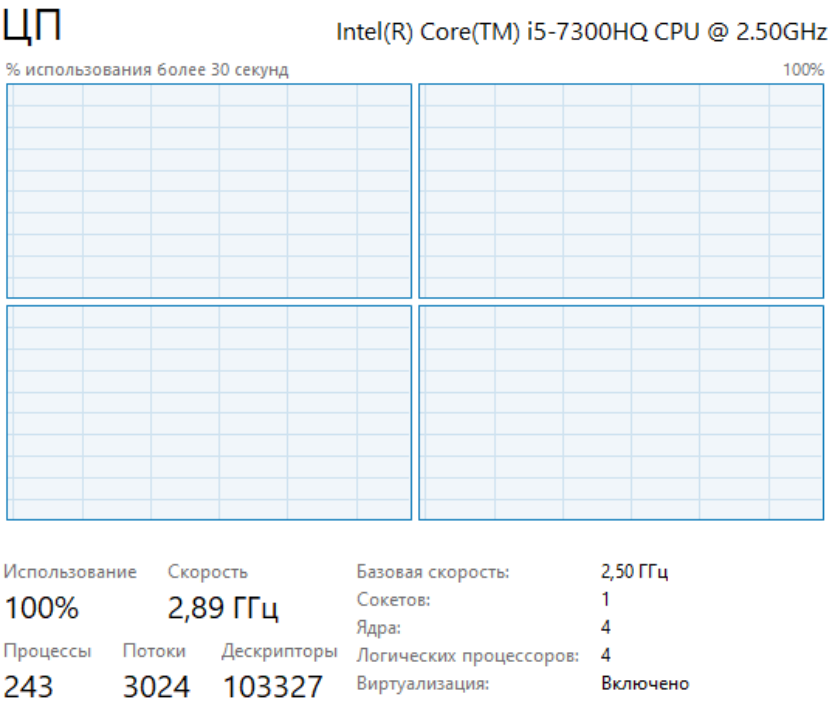


Рисунок 3.10 – Навантаження процесора при виконанні ПРГ1 T=4

Тестування при T=32 та T=64 дали показники далекі від теоретичних значень, а саме: при роботі програми із 32 потоками було отримано прискорення у 3,93 рази для N=512, у 4,28 рази при N=1024 та у 3,81 рази при N=2048; при роботі програми із 64 потоками було отримано прискорення у 3,93 рази для N=512, у 4,27 рази при N=1024 та у 3,78 рази при N=2048. Дана розбіжність у теоретичних та практичних показниках пояснюється апаратним обмеженням робочої станції, на якій виконувалось дослідження. Не дивлячись на те, що програма створювали 32 та 64 потоки, відповідно, кількість ядер в процесорі залишалась рівна чотирьом. Таким чином було витрачено багато часу на створення потоків і розбиття задачі на більшу кількість підзадач.

Однак, отримані результати кращі ніж при тестуванні на 4 потоках і це пояснюється більш дрібним розбиттям задачі. Таким чином потоки швидше виконувались, оскільки мали менше даних для обробки, що позитивно вплинуло на загальний час виконання розрахунків.

Наступним етапом аналізу є визначення коефіцієнту ефективності. Аналогічно до коефіцієнту прискорення побудована таблиця 3.8 із результатами обчислення коефіцієнту за раніше згаданою формулою (3.2):

Таблиця 3.8 – Коефіцієнт ефективності ПРГ1

N	Коефіцієнт ефективності в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	100%	96%	91%	12,28%	6,14%
1024	100%	97%	100%	13,38%	6,67%
2048	100%	98%	82%	11,9%	5,91%

Також отримані результати представлені у вигляді графіку(рис.3.11):

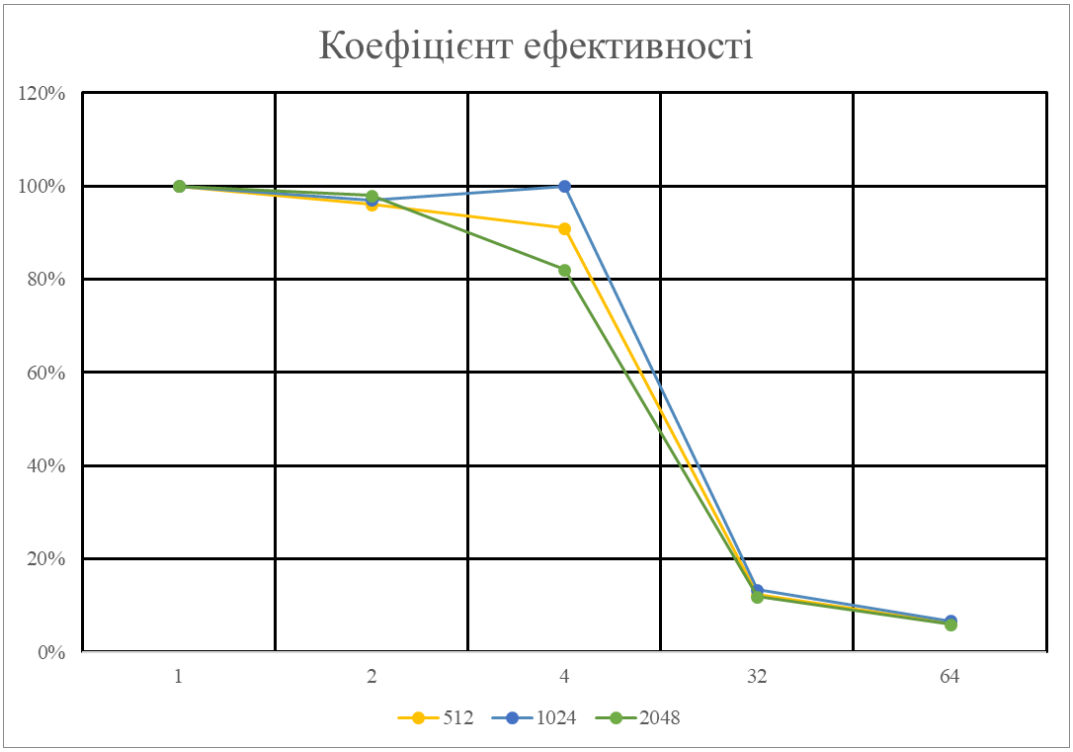


Рисунок 3.11 – Графік коефіцієнту ефективності ПРГ1

Отримані показники варто розподілити на дві групи: група тестувань, де кількість потоків рівна кількості ядер у процесорі та група тестувань, де кількість потоків значно більша за кількість ядер у процесорі.

До першої умовної групи належать тестування при T=1, T=2 та T=4. При зазначених кількостях потоках показники для різних розмірностей ж майже однаковими і при цьому високими. Невелике відхилення показників від 100% пояснюється витраченим часом, в очікуванні синхронізації із іншими потоками. Особливо це гарно демонструє показник при N=2048. В даному випадку через великий об'єм інформації потоки завершили виконання із вже відчутною різницею у часі, і демонструє коефіцієнт ефективності у 82%.

Тестуваннями другої групи є тестування при T=32 та T=64. Дані випадки показали низький коефіцієнт ефективності у 11 – 13 відсотків для 32 потоків, та у 5 – 6 відсотки для 64 потоків. Причиною таких показників є апаратне обмеження у 4 ядра при тестуванні на робочій станції автора. Через те, що всі створені задачі виконували 4 ядра, то потоки витратили багато часу в очікуванні синхронізації.

3.3.2 Тестування програми ПРГ2

Алгоритм тестування залишається незмінним і є аналогічним до тестування ПРГ1. Першочергово було перевірено правильність проведення обрахунків програмою при різній кількості ядер. Для розв'язання заданого виразу MA = MB * (MC * MD) * min(Z) вручну було обрано зручні значення. Для вектору Z обрано довжину 3 з елементами 3, 2 та 5. Для матриць MB, MC та MD було обрано розмірність 3 на 3, де кожен елемент – 1. За таких умов вираз має наступний вигляд:

$$A = MB \cdot (MC \cdot MD) \cdot \min(Z) =$$

$$= \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \left(\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \right) \cdot \min([3 \quad 2 \quad 5])$$

Для зручності виконаємо розрахунки у декілька простих дій:

1. Визначимо мінімальний елемент вектору Z:

   $\min([3 \quad 2 \quad 5]) = 2$

2. Виконаємо множення матриць MC та MD:

$$MC \cdot MD = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix}$$

3. Виконаємо множення матриці MB на добуток матриць MC та MD і на скаляр отриманій після операції знаходження мінімум вектора Z:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix} \cdot 2 = \begin{pmatrix} 18 & 18 & 18 \\ 18 & 18 & 18 \\ 18 & 18 & 18 \end{pmatrix}$$

Отже, вирішенням даного виразу при заданих раніше значень є матриця розмірністю 3 на 3, де кожен елемент є 18.

Для тестування правильності обрахунків в програмі задано аналогічні до ручного розрахунку вхідні дані (рис. 3.12).

Рисунок 3.12 – Ввід даних для вирішення виразу ПРГ2

Після введення даних був проведений обрахунок. Тестування проводилось при T=1, T=2, T=4, T=32 та T=64 та вивело однаковий результат. Виконання обчислень програма вивела результати у вікні «Результати». Результати виконання обчислень наведені нижче (рис. 3.13):

Рисунок 3.13 – Вивід результатів обчислень ПРГ2

Результати співпадають із порахованими вручну значеннями, тобто програма проводить розрахунки правильно.

Наступним етапом тестування є визначення часу, витраченого для здійснення обрахунків при різній кількості потоків та різних розмірностей матриць та вектора. Порівняно із тестуванням ПРГ1, ПРГ2 має складніший вираз тому час розрахунків очікується довшим.

Аналогічно до тестування ПРГ1, тестування буде проводитися, так само, у різних розмірностей матриць та векторів, а саме N=512, N=1024, N=2048Ю та різній кількості потоків у T=1, T=2, T=4, T=32 та T-64.

Відповідно до зазначених вище умов було проведено тестування ПРГ2. Результати занесено в таблицю.

Таблиця 3.9 – Час виконання ПРГ2

N	Час виконання (мс) в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	483	249	140	127	132
1024	4571	2414	1320	1160	1175
2048	211828	106984	61700	54590	53190

Використовуючи формулу для розрахунку коефіцієнта прискорення (3.1) та результати отримані під час замірів часу було побудовано таблицю коефіцієнта прискорення (таблиця 3.10). В таблиці представлені значення, визначені відповідно до об’єму даних (N) та кількості потоків (T).

Таблиця 3.10 – Коефіцієнт прискорення ПРГ2

N	Коефіцієнт прискорення (рази) в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	1	1,94	3,45	3,80	3,66
1024	1	1,89	3,46	3,94	3,89
2048	1	1,98	3,43	3,88	3,98

В ході проведення аналізу отримані результати були порівняні із теоретичними значеннями.

Із наведених результатів у таблиці видно різке прискорення при виконанні програми порівняно із виконанням на одному потоці (рис. 3.14).

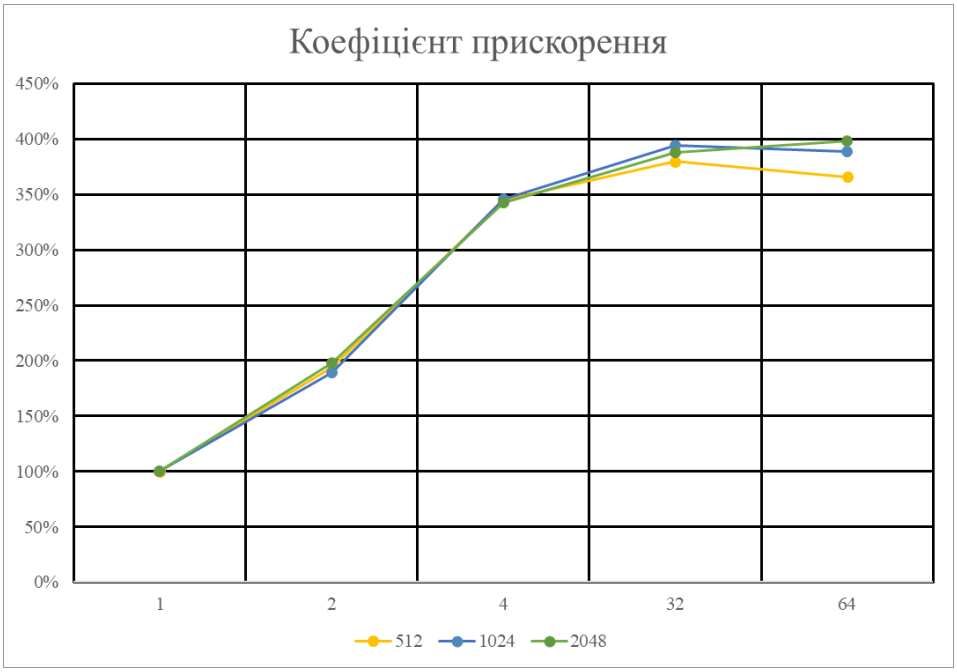


Рисунок 3.14 – Коефіцієнт прискорення ПРГ2

При збільшенні кількості потоків з одного до двох було отримано майже теоретичні значення, а саме: прискорення у 1,94 рази для N=512, прискорення

у 1,89 рази для N=1024 та прискорення у 1,98 рази для N=2048. Ідеального значення у 2 рази не було досягнуто через час витрачений на створення та синхронізацію потоків. З іншої сторони на результати тестування позитивно впливало, що програма використовувала половину ядер процесору. В даному режимі роботи система не зменшує частоту роботи ядер для збереження загальної температури процесора у межах норми. Також при наявності вільних ядер інші процеси операційної системи не заважали програмі виконуватись, адже їх навантаження було перерозподілено по іншим ядрам.

Тестування програми на 4 потоках дало високі показники. Були отримані наступні значення: прискорення у 3,45 рази для N=512, прискорення у 3,46 рази для N=1024 та прискорення у 3,43 рази для N=2048. При вказаних розмірностях відмінність від теоретичного значення пояснюються апаратними обмеженням робочої станції, на якій проводиться тестування. Негативно на результат вплинуло, те що процесор працював на 100% потужності, в наслідок чого були помічено зниження частоти роботи ядер із 3,3 ГГц до 3 ГГц для уникнення перегріву. Також негативно впливали фонові процеси системи, що під час обрахунку конкурували за ресурс процесора із процесами програми.

При тестуванні програми на T=32 та T=64 було отримано прискорення у 3 – 4 рази. Показники менші за теоретично очікуванні через малу кількість ядер у процесора робочій станції, на якій проводиться тестування.

Наступним етапом аналізу є визначення коефіцієнту ефективності за формулою (3.2). Дані наведені у таблиці 3.11:

Таблиця 3.11 – Коефіцієнт ефективності ПРГ2

N	Коефіцієнт ефективності в залежності від кількості потоків				
	T1	T2	T4	T32	T64
512	1	97%	86,25%	11,88%	5,72%
1024	1	94,5%	86,5	12,31%	6,08%
2048	1	99%	85,75%	12,13%	6,22%

Для спрощення проведення аналізу отримані результати поділено на дві групи: група тестувань, де кількість потоків рівна кількості ядер у процесорі та група тестувань, де кількість потоків значно більша за кількість ядер у процесорі.

До першої умовної групи належать тестування при T=1, T=2 та T=4. При зазначених кількостях потоках показники ефективності є високими. Невелике відхилення показників від 100% пояснюється витраченим часом, в очікуванні синхронізації із іншими потоками.

Тестуваннями другої групи – тестування при T=32 та T=64. Дані випадки показали низький коефіцієнт ефективності у 11 – 12 відсотків для 32 потоків, та у 5 – 6 відсотки для 64 потоків. Причиною таких показників є апаратне обмеження у 4 ядра на робочій станції, я якої здійснюється тестування.

Отримані результати наведені на рисунку 3.14:

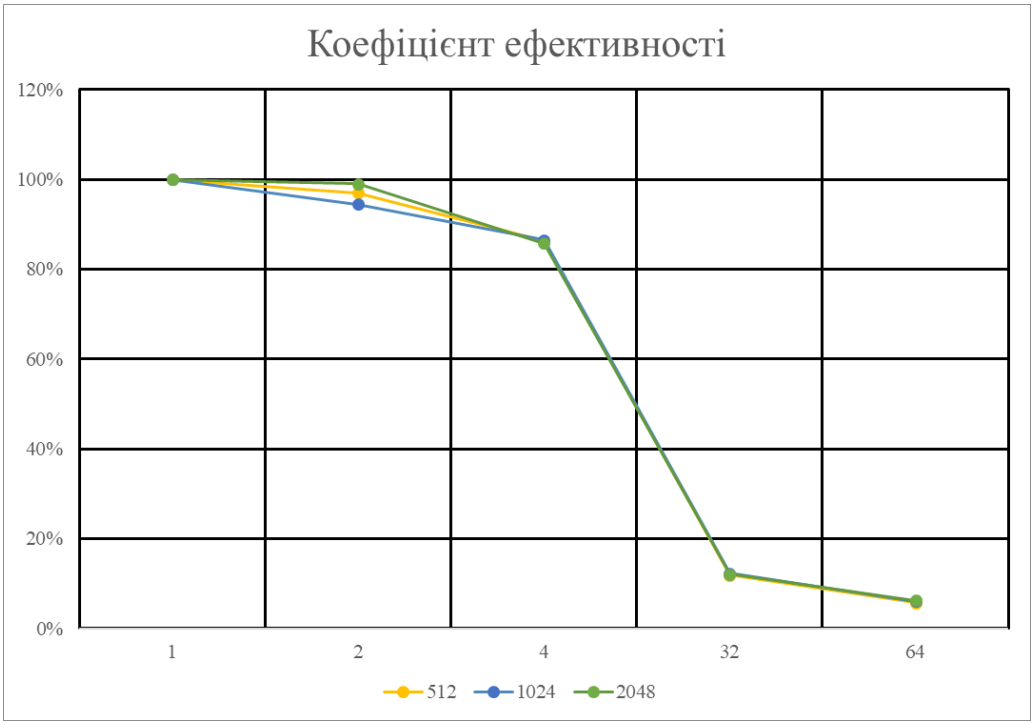


Рисунок 3.14 – Коефіцієнт ефективності ПРГ2

ВИСНОВКИ ДО РОЗДІЛУ 3

В рамках третього розділу було проведено розробку програмної складової програмно-апаратного комплексу. Для вирішення поставленої задачі, а саме прискорення обчислень двох векторно-матричних виразів було розроблено дві підпрограми ПРГ1 та ПРГ2, що об'єднані в одну універсальну програму із графічним інтерфейсом.

Задля вирішення задачі прискорення розрахунків заданих виразів розробка програмного забезпечення базувалась на принципах паралельності і паралельного програмування. З метою реалізації підпрограм на вказаних принципах першочергово було розроблено паралельні алгоритми для кожного виразу. На основі паралельних алгоритмів були побудовані алгоритми роботи для потоків T1 та Ti. Необхідність розробки двох алгоритмів потоків для кожної програми полягає у введені і виведені даних у першому потоці на відміну від всіх інших При розробці алгоритмів потоків в якості засобу для синхронізації були обрані бар'єри. В ПРГ1 бар'єри використовуються для синхронізації потоків після введення початкових даних та перед виведенням кінцевих результатів. У підпрограмі ПРГ2 з'являється додатковий, третій, бар'єр для синхронізації потоків при визначені мінімального значення вектора. Для наочного пояснення взаємодії потоків були розроблені відповідні схеми (див. рис. 3.1 та рис 3.5). На основі розроблених паралельних алгоритмів, алгоритмів роботи потоків та схем взаємодії потоків було написано програму на мові програмування Java із використанням фреймворку ForkJoinPool для реалізації паралельного обчислення виразів. Програма була розроблена на принципах ООП, тому розподілена на окремі модулі для роботи з графічним інтерфейсом, виконанням операцій з матрицями, виконанням операцій із векторами.

Важливим етапом розробки програмного забезпечення було його тестування. Через фізичну відсутність розробленого у розділі 2 програмно-

апаратного комплексу, тестування проводилось на робочій станції автора, що значно вплинуло на тестування програм при кількості потоків більшої за 4. Підпрограми ПРГ1 та ПРГ2 тестувались окремо, але за одним алгоритмом, а саме: перевірка правильності проведення розрахунків, тестування із виміром часу, визначення коефіцієнта прискорення та визначення коефіцієнта ефективності. Під час тестування ПРГ1 при збільшені потоків до T=2 вдалося досягти близьких до теоретичних значень із невеликими відхиленнями. При збільшені потоків до T=4 коефіцієнт прискорення становив в середньому 3,5 рази. Відхилення від теоретичного значення пояснюється недосконалістю апаратної частини робочої станції, на якій проводилось тестування та витрати часу на створення й синхронізацію потоків. При збільшені потоків до 32 або 64 – кількість разів прискорення майже не змінилась, а низький коефіцієнт ефективності у 11 – 12% та 5 – 6% вказав на недостатню кількості ядер процесора на робочій станції для проведення повноцінного тестування при великій кількості потоків.

У результаті роботи, описаної в третьому розділі, було розроблено програмне забезпечення, яке відповідає поставленій меті – зменешення часу виконання обрахунків векторно-матричних задач. Завдяки обраним принципам побудови системи, програма здійснює прогнозоване прискорення близьке до теоретичного значення, характеризується високою гнучкістю, що забезпечує її подальшу оптимізацію для розв’язання інших обчислювальних задач. Передбачено можливість як програмного розширення функціональності так і покращення апаратної частини.

ВИСНОВКИ

У рамках дипломного проєкту було розроблено програмно-апаратний комплекс на базі багатоядерного процесору з метою збільшення швидкості векторно-матричних обчислень.

Проведено огляд існуючих рішень побудови програмно-апаратного комплексу. В результаті розглянутих рішень було розроблено апаратну складову на базі 32-ядерного процесора AMD Threadripper 7970X, встановлено 128 Гб оперативної пам’яті DDR5 від компанії Kingstone, оснащено швидким SSD Samsung 990 EVO PLUS на 2 ТБ, підключеного до пристрою введення-виведення та повітряним охолодженням Arctic Freezer 4U-M. Для покращення продуктивності в майбутньому закладена можливість оснащення комплексу більш потужним GPU, можливість збільшення об'єму оперативної пам'яті до 1 ТБ та збільшенням кількості пристрої вводу-виведення для зменшення часу читання та запису інформації.

Для розробки програмної складової було обрано мову програмування Java із використанням фреймворку ForkJoinPool для реалізації паралельного виконання векторно-матричних обчислень. За допомогою платформи JavaFX розроблено графічний інтерфейс, що дозволяє користувачу керувати виконанням програми: переключатися між ПРГ1 та ПРГ2, вводити дані вручну, завантажувати або зберігати їх у файл.

Для розробки ПРГ1 та ПРГ2 розроблено паралельно-математичні алгоритми виразів, розроблено схеми взаємодії потоків. Для вирішення задачі синхронізації потоків обрано засіб синхронізації бар’єр. В свою чергу для захисту спільного ресурсу використовуються атомарні змінні. Програму реалізовано на засадах об’єктно орієнтованого програмування, що надає можливість легкого і безпечного розширення функціоналу у майбутньому.

На етапі тестування ПРГ1 та ПРГ2 було перевірено правильність виконання розрахунків кожної із програм при різній кількості потоків: 1, 2, 4,

32, 64. Було визначено час виконання обрахунків програмами при різній кількості потоків та різному об’ємі дані. На основі визначеного часу було розраховано коефіцієнти прискорення та ефективності. Тестування показало високі результати прискорення і ефективності у програм, а саме: для ПРГ1 середнє прискорення для T=2 склало 1,94 рази та 3,64 для T=4 при ефективності 97% та 91% відповідно. Для ПРГ2 при T=2 та середній ефективності 96% середнє прискорення склало 1,93 рази; при T=4, ефективності у 86% середнє прискорення склало 3,45 рази. Наведені результати у тестуванні близькі до теоретичних значень, що вказує на ефективність розробленої програми і прогнозованість її роботи при масштабуванні.

Розроблений програмно-апаратний комплекс легко піддається модифікація та масштабуванню. ПАК можна легко адаптувати для вирішення інших складних обчислювальних задач, має значний потенціал для вдосконалення, оптимізації та збільшення функціональності у майбутньому.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Classification of Computers. Geeks for Geeks. Classification of Computers. 04.04.2025. URL: https://www.geeksforgeeks.org/classification-of-computers/ (дата звернення: 02.06.2025).
2. AMD. AMD Ryzen Threadripper PRO 3995WX Drivers & Support. URL: https://www.amd.com/en/support/downloads/drivers.html/processors/ryzen-threadripper-pro/ryzen-threadripper-pro-3000wx-series/amd-ryzen-threadripper-pro-3995wx.html (дата звернення: 02.06.2025).
3. Apache. Apache Hadoop. URL: https://hadoop.apache.org/ (дата звернення: 02.06.2025).
4. Intel. Intel Core i9-13900K Processor. URL: https://www.intel.com/content/www/us/en/products/sku/230496/intel-core-i913900k-processor-36m-cache-up-to-5-80-ghz/specifications.html (дата звернення: 02.06.2025).
5. AMD. AMD Ryzen Threadripper Processor. URL: https://www.amd.com/en/products/processors/workstations/ryzen-threadripper.html (дата звернення: 02.06.2025).
6. Gigabyte. TRX50 AERO D (rev 1.1). URL: https://www.gigabyte.com/ua/Motherboard/TRX50-AERO-D-rev-11 (дата звернення: 02.06.2025).
7. Kingston. DDR5 4800MT/s ECC Registered DIMM. URL: https://www.kingston.com/unitedkingdom/ua/memory/server-premier/ddr5-4800mts-ecc-registered-dimm (дата звернення: 02.06.2025).
8. Sapphire. PULSE AMD Radeon RX 6500 XT 8GB. URL: https://www.sapphiretech.com/en/consumer/pulse-radeon-rx-6500-xt-8g-gddr6 (дата звернення: 02.06.2025).
9. Arctic. Freezer 4U-M. URL: https://www.arctic.de/en/Freezer-4U-M/ (дата звернення: 02.06.2025).

10. Chieftec. POLARIS 3.0 SERIES. URL: https://www.chieftec.eu/products-detail/544/POLARIS_3.0_SERIES_/ (дата звернення: 02.06.2025).
11. Gigabyte. GIGABYTE C301 GLASS V2. URL: https://www.gigabyte.com/ua/PC-Case/GB-C301G-TYPE-C-V2 (дата звернення: 02.06.2025).
12. Samsung. 990 EVO Plus. URL: https://semiconductor.samsung.com/consumer-storage/internal-ssd/990-evo-plus/ (дата звернення: 02.06.2025).
13. Alcorn P. AMD Threadripper Pro 3995WX Review: Ripping With 8 Memory Channels. Tom'sHARDWARE. Reviews. 07.09.2023. URL: https://www.tomshardware.com/reviews/amd-threadripper-pro-3995wx-review (дата звернення: 02.06.2025).
14. Lee J. AMD Ryzen Threadripper PRO 3995WX Review A Bold WEPYC. ServeTheHome. 10.01.2021. URL: https://www.servethehome.com/amd-ryzen-threadripper-pro-3995wx-review-a-bold-wepyc/2/ (дата звернення: 02.06.2025).

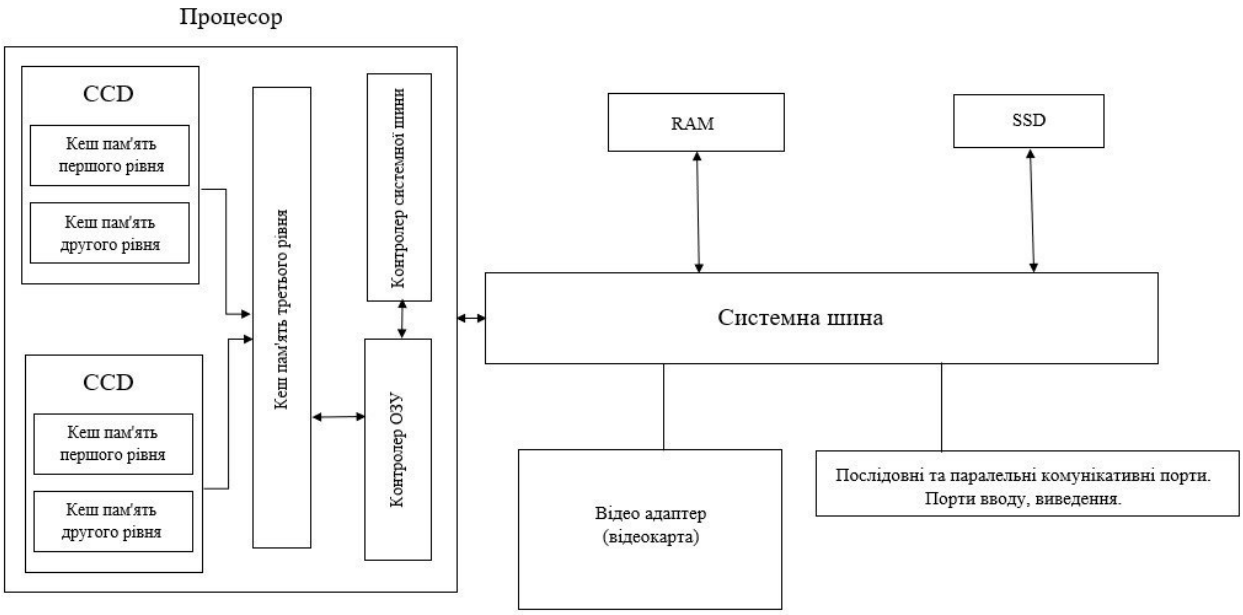
ДОДАТОК А

Програмно-апаратний комплекс для векторно-матричних обчислень

Структурна схема програмно-апаратного комплексу
ІАЛЦ.466500.004 Д1

Аркушів 1

Київ 2025 р.



					ІАЛЦ.466500.004 Д1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Маруженко І.С.			**Програмно-апаратний комплекс для векторно-матричних обчислень** Структурна схема програмно-апаратного комплексу	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Корочкін О.В.					
Реценз.						**НТУУ «КПІ», ФІОТ, ІО-13**	
Н. Контр.		Гончарено О.О.					
Затвердив		Новотарський М.А					

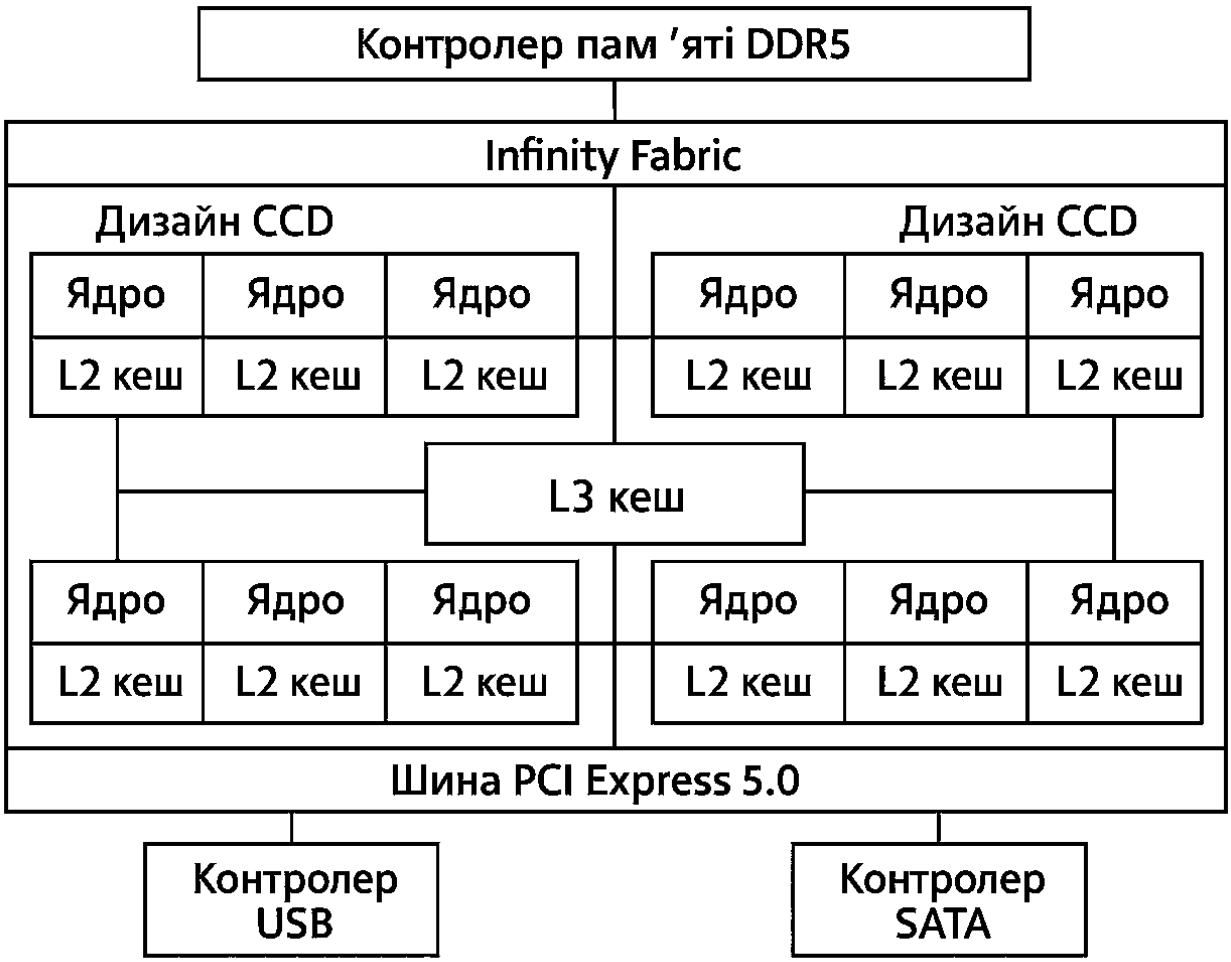
ДОДАТОК Б

Програмно-апаратний комплекс для векторно-матричних обчислень

Структурна схема процесору програмно-апаратного комплексу ІАЛЦ.466500.005 Д2

Аркушів 1

Київ 2025 р.



					ІАЛЦ.466500.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Маруженко І.С.			**Програмно-апаратний комплекс для векторно-матричних обчислень** Структурна схема процесору програмно-апаратного комплексу	Літ. / Аркуш: 1 / Аркушів: 1
Перевірив		Корочкін О.В.				
Реценз.						**НТУУ «КПІ», ФІОТ, ІО-13**
Н. Контр.		Гончарено О.О.				
Затвердив		Новотарський М.А				

ДОДАТОК В

Програмно-апаратний комплекс для векторно-матричних обчислень

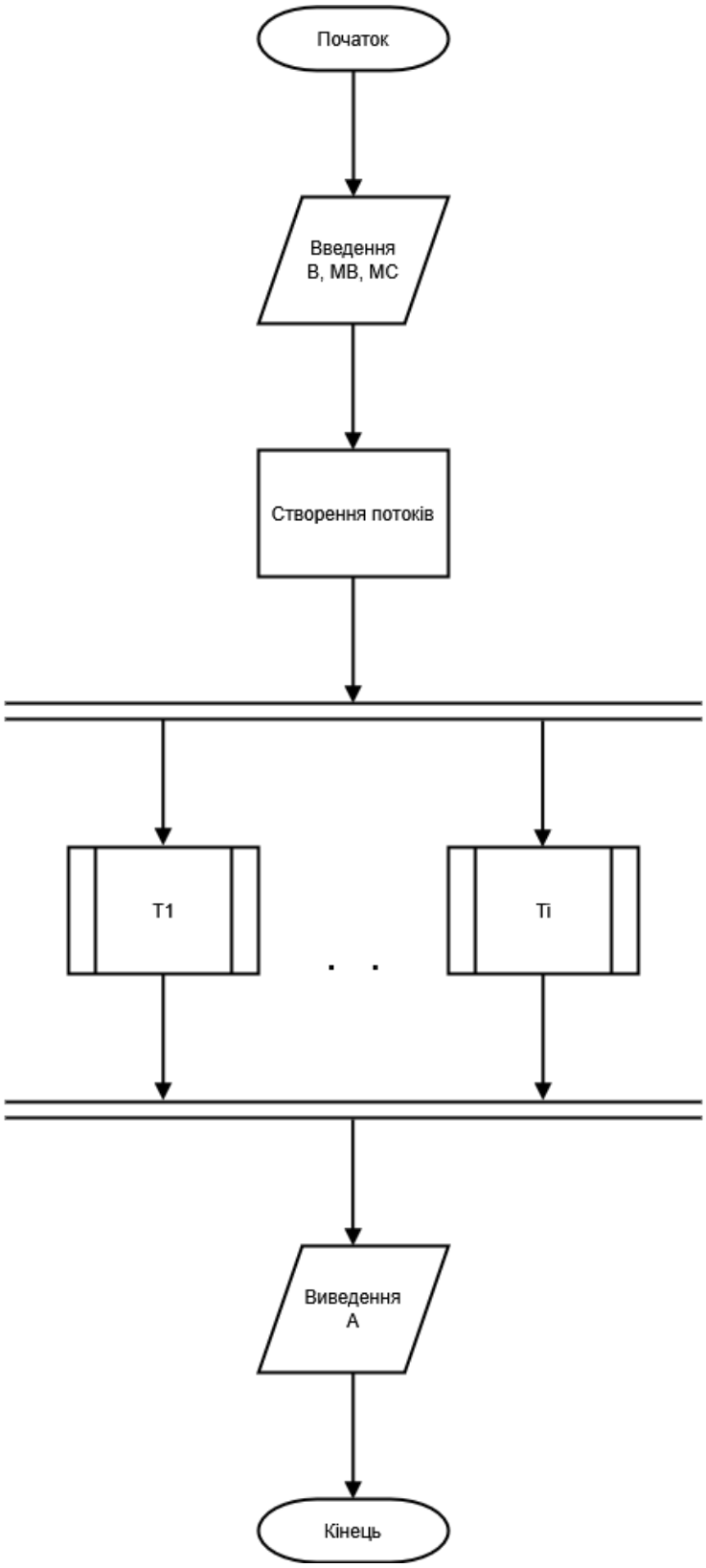
Блок-схеми алгоритмів

ІАЛЦ.466500.006 Д3

Аркушів 2

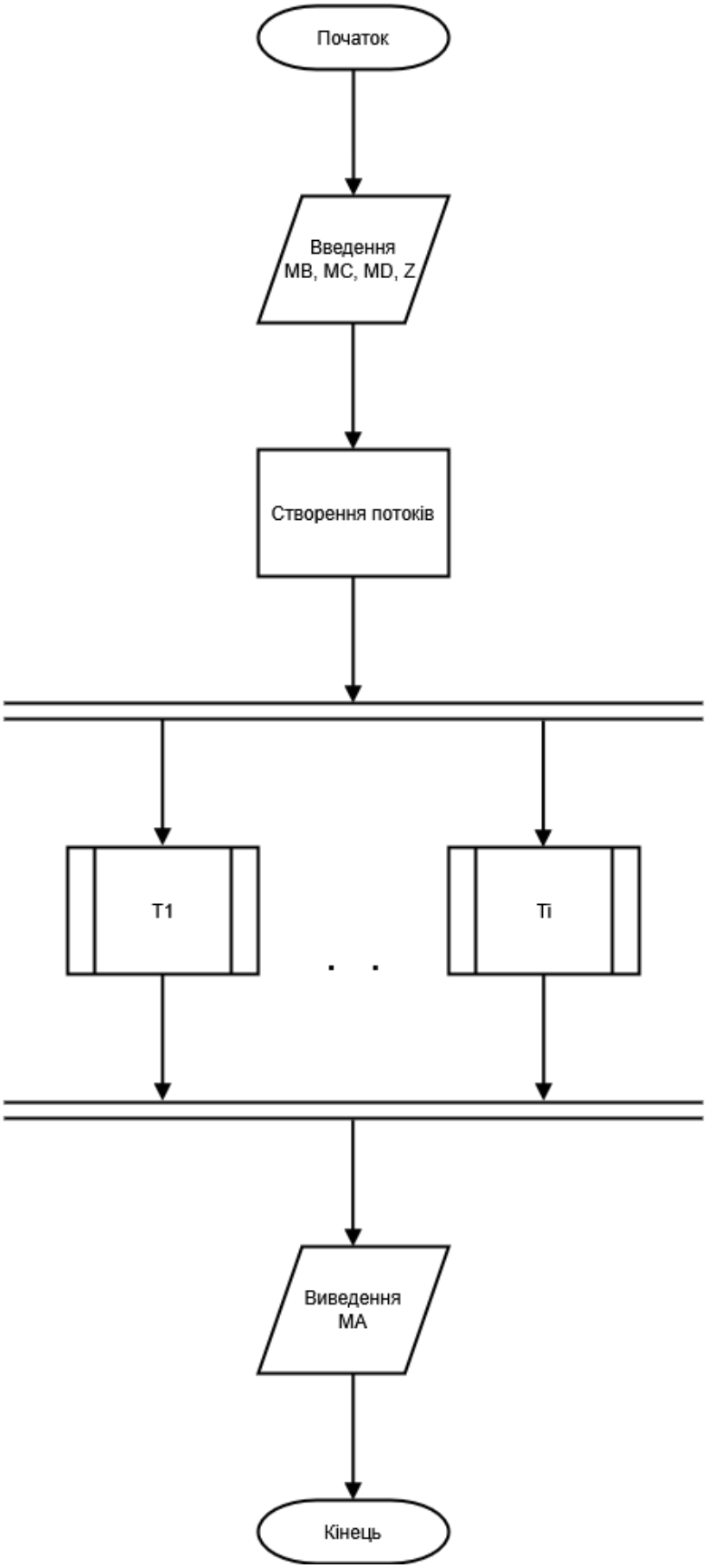
Київ 2025 р.

Блок-схема ПРГ1



					ІАЛЦ.466500.006 Д3	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Маруженко І.С.			**Програмно-апаратний комплекс для векторно-матричних обчислень** Блок-схема алгоритмів	Літ. / Аркуш 1 / Аркушів 2
Перевірив		Корочкін О.В.				
Реценз.						**НТУУ «КПІ», ФІОТ, ІО-13**
Н. Контр.		Гончарено О.О.				
Затвердив		Новотарський М.А				

Блок-схема ПРГ2



ДОДАТОК Г

Програмно-апаратний комплекс для векторно-матричних обчислень

Лістинг програми

ІАЛЦ.466500.007 Д4

Аркушів 21

Київ 2025 р.

Main.java

```
/**
 * Програмно-апаратний комплекс для векторно-матричних обчислень
 *
 * ПРГ1 та ПРГ2
 *
 * Розробив студент 4 курсу
 * групи ІО-13
 * Маруженко Іван Сергійович
 *
 * Release: 18.05.2025
 *
 * Version: 1.7
 */

package com.example.matrixcalc;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.util.Locale;
import java.util.ResourceBundle;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {

        ResourceBundle bundle = ResourceBundle.getBundle("lang", Locale.forLanguageTag("uk"));
        Locale.setDefault(new Locale("uk"));

        FXMLLoader loader = new FXMLLoader(getClass().getResource("/com/example/matrixcalc/Main.fxml"), bundle);

        Parent root = loader.load();

        Scene scene = new Scene(root, 1200, 800);

        primaryStage.setScene(scene);
        primaryStage.setTitle("Matrix Calc");
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

PRG1.java

```
package com.example.matrixcalc.util;

import java.util.Arrays;
import java.util.List;
import java.util.ResourceBundle;

public class PRG1 {
    // A = B * (MB * MC)
    public static int[] prg1(int[] B, int[][] MB, int[][] MC, int P, List<String> steps, ResourceBundle bundle) {
        int N = B.length;
        int H = (int) Math.ceil((double) N / P);

        // Крок 1. Множення MB та MC
        int[][] mb_mc = MatrixUtils.multiply(MB, MC, P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.mul_mb_mc") + "\n" + MatrixUtils.matrixToString(mb_mc));
        }

        // Крок 2. Множення B на результат попереднього кроку
        int[] A = VectorUtils.multiply_matrix_vector(mb_mc, B, P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.mul_b_mb_mc") + " " + Arrays.toString(A));
        }

        return A;
    }
}
```

PRG2.java

```
package com.example.matrixcalc.util;

import java.util.List;
import java.util.ResourceBundle;
import java.util.concurrent.atomic.AtomicInteger;

public class PRG2 {
    // MA = MB * (MC * MD) * min(Z)
    public static int[][] prg2(int[][] MB, int[][] MC, int[][] MD, int[] Z, int P,
                               List<String> steps, ResourceBundle bundle) {
        int N = Z.length;
        int H = (int) Math.ceil((double) N / P);

        // Крок 1. Множення MC та MD
        int[][] mc_md = MatrixUtils.multiply(MC, MD, P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.mul_mc_md") + "\n" + MatrixUtils.matrixToString(mc_md));
        }

        // Крок 2. Множення MB на результат попереднього кроку
        int[][] mb_mc_md = MatrixUtils.multiply(MB, mc_md, P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.mul_mb_mc_md") + "\n" + MatrixUtils.matrixToString(mb_mc_md));
        }

        // Крок 3. Знаходимо мінімальний елемент масиву Z
        AtomicInteger x = new AtomicInteger(Integer.MAX_VALUE);
        VectorUtils.min_vector(Z, x, P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.min_z") + x);
```

```
        }

        // Крок 4. Домножуємо матрицю mb_mc_md на мінімальний Z
        int[][] MA = MatrixUtils.multiply_z(mb_mc_md, x.get(), P, H);
        if (steps != null) {
            steps.add(bundle.getString("step.mul_min_z") + "\n" + MatrixUtils.matrixToString(MA));
        }

        return MA;
    }
}
```

MatrixUtils.java

```
package com.example.matrixcalc.util;

import java.util.concurrent.ForkJoinPool;

public class MatrixUtils {
    /**
     * Паралельне множення матриць
     */
    public static int[][] multiply(int[][] a, int[][] b, int pCount, int H) {
        if (b.length != a[0].length) {
            throw new IllegalArgumentException("Incompatible matrices");
        }

        int[][] result = new int[a.length][b[0].length];
        ForkJoinPool.commonPool().setParallelism(pCount);
        ForkJoinPool.commonPool().submit(() ->
                java.util.stream.IntStream.range(0, pCount).parallel().forEach(threadID -> {
                    int sum;
                    for (int i = 0; i < a.length; i++)
                    {
                        for (int j = threadID * H; j < threadID * H + H; j++) {
                            if (j < b[0].length) {
                                sum = 0;
                                for (int k = 0; k < b.length; k++) {
                                    sum += a[i][k] * b[k][j];
                                }
                                result[i][j] = sum;
                            } else {
                                break;
                            }
                        }
                    }
                })
        ).join();

        return result;
    }

    /**
     * Паралельне множення матриці на скаляр
     */
    public static int[][] multiply_z(int[][] a, int x, int pCount, int H) {
        int[][] result = new int[a.length][a[0].length];
        ForkJoinPool.commonPool().submit(() ->
                java.util.stream.IntStream.range(0, pCount).parallel().forEach(threadID -> {
                    for (int i = threadID * H; i < threadID * H + H; i++) {
                        if (i < a.length) {
```

```
                    for (int j = 0; j < a[0].length; j++) {
                        result[i][j] = a[i][j] * x;
                    }
                } else {
                    break;
                }
            }
        })
    ).join();
    return result;
  }

  /**
   * Форматування матриць в строку логування
   */
  public static String matrixToString(int[][] m) {
    StringBuilder sb = new StringBuilder();
    for (int[] row : m) {
      for (int x : row) sb.append(x).append("   ");
      sb.append("\n");
    }
    return sb.toString();
  }
}
```

VectorUtils.java

```
package com.example.matrixcalc.util;

import java.util.concurrent.ForkJoinPool;

public class MatrixUtils {
  /**
   * Паралельне множення матриць
   */
  public static int[][] multiply(int[][] a, int[][] b, int pCount, int H) {
    if (b.length != a[0].length) {
      throw new IllegalArgumentException("Incompatible matrices");
    }

    int[][] result = new int[a.length][b[0].length];
    ForkJoinPool.commonPool().setParallelism(pCount);
    ForkJoinPool.commonPool().submit(() ->
        java.util.stream.IntStream.range(0, pCount).parallel().forEach(threadID -> {
          int sum;
          for (int i = 0; i < a.length; i++)
          {
            for (int j = threadID * H; j < threadID * H + H; j++) {
              if (j < b[0].length) {
                sum = 0;
                for (int k = 0; k < b.length; k++) {
                  sum += a[i][k] * b[k][j];
                }
                result[i][j] = sum;
              } else {
                break;
              }
            }
          }
        })
    ).join();
```

```
        return result;
    }

    /**
     * Паралельне множення матриці на скаляр
     */
    public static int[][] multiply_z(int[][] a, int x, int pCount, int H) {
        int[][] result = new int[a.length][a[0].length];
        ForkJoinPool.commonPool().submit(() ->
                java.util.stream.IntStream.range(0, pCount).parallel().forEach(threadID -> {
                    for (int i = threadID * H; i < threadID * H + H; i++) {
                        if (i < a.length) {
                            for (int j = 0; j < a[0].length; j++) {
                                result[i][j] = a[i][j] * x;
                            }
                        } else {
                            break;
                        }
                    }
                })
        ).join();
        return result;
    }

    /**
     * Форматування матриць в строку логування
     */
    public static String matrixToString(int[][] m) {
        StringBuilder sb = new StringBuilder();
        for (int[] row : m) {
            for (int x : row) sb.append(x).append("   ");
            sb.append("\n");
        }
        return sb.toString();
    }
}
```

MainController.java

```
package com.example.matrixcalc.controller;

import com.example.matrixcalc.util.PRG1;
import com.example.matrixcalc.util.PRG2;
import com.example.matrixcalc.util.MatrixUtils;
import javafx.application.Platform;
import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.*;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.scene.input.ClipboardContent;
```

```
import javafx.scene.layout.VBox;
import javafx.scene.text.Font;
import javafx.stage.FileChooser;
import javafx.stage.Window;
import javafx.util.converter.IntegerStringConverter;

import java.awt.*;
import java.io.*;
import java.util.List;
import java.util.*;

public class MainController {

    // MB
    @FXML
    private TextField rowsMB, colsMB;
    @FXML
    private TableView<ObservableList<IntegerProperty>> tableMB;
    @FXML
    private Button createMB, loadMBButton;

    // MC
    @FXML
    private TextField rowsMC, colsMC;
    @FXML
    private TableView<ObservableList<IntegerProperty>> tableMC;
    @FXML
    private Button createMC, loadMCButton;

    // MD
    @FXML
    private TextField rowsMD, colsMD;
    @FXML
    private TableView<ObservableList<IntegerProperty>> tableMD;
    @FXML
    private Button createMD, loadMDButton;

    // B
    @FXML
    private TextField lengthB;
    @FXML
    private TableView<ObservableList<IntegerProperty>> tableB;
    @FXML
    private Button createB, loadBButton;

    // Z
    @FXML
    private TextField lengthZ;
    @FXML
    private TableView<ObservableList<IntegerProperty>> tableZ;
    @FXML
    private Button createZ, loadZButton;

    // Інші елементи
    @FXML
    private Button calculateButton, copyButton, clearButton;
    @FXML
    private CheckBox showStepsCheck, darkThemeCheck, autoSaveCheck, confirmExitCheck, soundNotifyCheck;
    @FXML
    private ComboBox<String> languageCombo, operationCombo;
    @FXML
    private Label fontSizeLabel, fontSizeTitle;
```

```
@FXML
private Label threadCountLabel, threadCountTitle;
@FXML
private Label bLabel, mbLabel, mcLabel, mdLabel, zLabel;
@FXML
private Label resultLabel, stepsLabel;
@FXML
private Label languageLabel, operationLabel;
@FXML
private Label lengthLabelB;
@FXML
private Label lengthLabelZ;
@FXML
private ListView<String> stepsList;
@FXML
private Menu fileMenu;
@FXML
private Menu helpMenu;
@FXML
private MenuItem saveTxtMenuItem, exitMenuItem, aboutMenuItem;
@FXML
private Slider fontSizeSlider, threadCountSlider;
@FXML
private Tab inputTab, resultTab, settingsTab;
@FXML
private TextArea resultArea;
@FXML
private VBox multiMatrixPane;
@FXML
private VBox blockB, blockMB, blockMC, blockMD, blockZ;

// Приватні змінні
private final List<String> langCodes = List.of("uk", "en");
private static final String DARK_THEME_CSS = "/styles/dark-theme.css";
private static final String SETTINGS_FILE = "settings.properties";
private ResourceBundle bundle;

private final List<String> opKeys = List.of(
        "operation.prg1",
        "operation.prg2"
);

private int P;

@FXML
public void initialize() {
    // 1. Список мов
    List<String> langCodes = List.of("uk", "en");
    List<String> langKeys = List.of("language.ukrainian", "language.english");

    // 2. Завантажити обрану мову або за замовчуванням "uk"
    String savedLang = loadLanguage();
    String langCode = (savedLang != null && langCodes.contains(savedLang)) ? savedLang : "uk";
    setLanguage(langCode);

    // 3. Отримати список мов
    ObservableList<String> languages = FXCollections.observableArrayList();
    for (String key : langKeys) {
        languages.add(bundle.getString(key));
    }
    languageCombo.setItems(languages);
```

```
// 4. Встановити обрану мову
int langIndex = langCodes.indexOf(langCode);
if (langIndex < 0) langIndex = 0;
languageCombo.getSelectionModel().select(langIndex);

// 5. Listener зміна мови
languageCombo.valueProperty().addListener((obs, oldVal, newVal) -> {
   int idx = languages.indexOf(newVal);
   String selectedCode = (idx >= 0 && idx < langCodes.size()) ? langCodes.get(idx) : "uk";
   saveLanguage(selectedCode);
   Alert alert = new Alert(
         Alert.AlertType.INFORMATION,
         bundle.getString("alert.restart_to_change_language"),
         ButtonType.OK
   );
   alert.showAndWait();
});

updateOperationsCombo();
operationCombo.getSelectionModel().selectFirst();
operationCombo.getSelectionModel().selectedItemProperty().addListener((obs, oldOp, newOp) ->
      updateInputPanels(newOp));

// --- Події для MB ---
createMB.setOnAction(e -> {
   createMatrix(tableMB, rowsMB, colsMB, "MB");
   fillTableWithRandom(tableMB);
});
loadMBButton.setOnAction(e -> loadTableFromFile(tableMB, true));

// --- Події для MC ---
createMC.setOnAction(e -> {
   createMatrix(tableMC, rowsMC, colsMC, "MC");
   fillTableWithRandom(tableMC);
});
loadMCButton.setOnAction(e -> loadTableFromFile(tableMC, true));

// --- Події для MD ---
createMD.setOnAction(e -> {
   createMatrix(tableMD, rowsMD, colsMD, "MD");
   fillTableWithRandom(tableMD);
});
loadMDButton.setOnAction(e -> loadTableFromFile(tableMD, true));

// --- Події для B ---
createB.setOnAction(e -> {
   createVector(tableB, lengthB, "B");
   fillTableWithRandom(tableB);
});
loadBButton.setOnAction(e -> loadTableFromFile(tableB, false));

// --- Події для Z ---
createZ.setOnAction(e -> {
   createVector(tableZ, lengthZ, "Z");
   fillTableWithRandom(tableZ);
});
loadZButton.setOnAction(e -> loadTableFromFile(tableZ, false));

// --- Задати початкове значення потоків ---
setP(4);

calculateButton.setOnAction(e -> onCalculate());
```

```
saveTxtMenuItem.setOnAction(e -> onSaveTxt());
exitMenuItem.setOnAction(e -> handleExit());
aboutMenuItem.setOnAction(e -> showAboutDialog());

copyButton.setOnAction(e -> {
   StringBuilder sb = new StringBuilder(bundle.getString("label.result") + ":\n")
         .append(resultArea.getText())
         .append("\n\n")
         .append(bundle.getString("label.steps"))
         .append(":\n");
   stepsList.getItems().forEach(line -> sb.append(line).append("\n"));
   ClipboardContent content = new ClipboardContent();
   content.putString(sb.toString());
   javafx.scene.input.Clipboard.getSystemClipboard().setContent(content);
});

clearButton.setOnAction(e -> {
   resultArea.clear();
   stepsList.getItems().clear();
});

showStepsCheck.setSelected(true);
stepsList.setVisible(true);
stepsList.setManaged(true);
showStepsCheck.selectedProperty().addListener((obs, oldVal, newVal) -> {
   stepsList.setVisible(newVal);
   stepsList.setManaged(newVal);
});

darkThemeCheck.selectedProperty().addListener((obs, oldVal, isOn) -> {
   Scene scene = showStepsCheck.getScene();
   if (scene == null) return;
   String css = getClass().getResource(DARK_THEME_CSS).toExternalForm();
   if (isOn) {
      if (!scene.getStylesheets().contains(css)) {
         scene.getStylesheets().add(css);
      }
   } else {
      scene.getStylesheets().remove(css);
   }
});

fontSizeSlider.valueProperty().addListener((obs, oldVal, newVal) -> {
   int size = newVal.intValue();
   fontSizeLabel.setText(Integer.toString(size));
   resultArea.setStyle("-fx-font-size: " + size + "px;");
   stepsList.setStyle("-fx-font-size: " + size + "px;");
});

threadCountSlider.valueProperty().addListener((obs, oldVal, newVal) -> {
   setP(newVal.intValue());
   threadCountLabel.setText(Integer.toString(getP()));
});

stepsList.setCellFactory(lv -> {
   ListCell<String> cell = new ListCell<>();
   Label lbl = new Label();
   lbl.setWrapText(true);
   fontSizeSlider.valueProperty().addListener((o, a, b) ->
         lbl.setFont(Font.font("Consolas", b.doubleValue()))
   );
   lbl.setFont(Font.font("Consolas", fontSizeSlider.getValue()));
```

```
            cell.setGraphic(lbl);
            cell.setPrefWidth(0);
            cell.setWrapText(true);
            cell.itemProperty().addListener((o, oldText, newText) ->
                lbl.setText(newText == null ? "" : newText)
            );
            return cell;
        });

        operationCombo.getSelectionModel().selectedItemProperty().addListener((obs, oldOp, newOp) ->
            updateInputPanels(newOp));
        updateInputPanels(operationCombo.getSelectionModel().getSelectedItem());
    }

    private void createMatrix(TableView<ObservableList<IntegerProperty>> table, TextField rowsField,
                              TextField colsField, String label) {
        int rows = parseSafe(rowsField.getText());
        int cols = parseSafe(colsField.getText());
        if (rows <= 0 || cols <= 0) {
            showError("Вкажіть коректні розміри матриці " + label + "!");
            return;
        }
        setupTable(table, rows, cols);
    }

    private void createVector(TableView<ObservableList<IntegerProperty>> table, TextField lenField, String label) {
        int len = parseSafe(lenField.getText());
        if (len <= 0) {
            showError("Вкажіть коректну довжину вектора " + label + "!");
            return;
        }
        setupTable(table, 1, len);
    }

    private void fillTableWithRandom(TableView<ObservableList<IntegerProperty>> table) {
        Random rnd = new Random();
        for (ObservableList<IntegerProperty> row : table.getItems()) {
            for (IntegerProperty value : row) {
                value.set(1 + rnd.nextInt(20));
            }
        }
    }

    private void setupTable(TableView<ObservableList<IntegerProperty>> table, int rows, int cols) {
        if (rows <= 0 || cols <= 0) return;
        table.getItems().clear();
        table.getColumns().clear();
        for (int j = 0; j < cols; j++) {
            final int idx = j;
            TableColumn<ObservableList<IntegerProperty>, Integer> col = new TableColumn<>("C" + (j + 1));
            col.setCellValueFactory(p -> p.getValue().get(idx).asObject());
            col.setCellFactory(TextFieldTableCell.forTableColumn(new IntegerStringConverter()));
            col.setPrefWidth(60);
            table.getColumns().add(col);
        }
        for (int i = 0; i < rows; i++) {
            ObservableList<IntegerProperty> row = FXCollections.observableArrayList();
            for (int j = 0; j < cols; j++) {
                row.add(new SimpleIntegerProperty(0));
            }
            table.getItems().add(row);
```

```
    }
    table.setEditable(true);
}

private void loadTableFromFile(TableView<ObservableList<IntegerProperty>> table, boolean isMatrix) {
    FileChooser fc = new FileChooser();
    fc.setTitle("Завантажити " + (isMatrix ? "матрицю" : "вектор"));
    fc.getExtensionFilters().add(new FileChooser.ExtensionFilter("Текстові файли", "*.txt"));
    Window win = table.getScene().getWindow();
    File file = fc.showOpenDialog(win);
    if (file == null) return;

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        List<int[]> rows = new ArrayList<>();
        String line;
        while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty()) continue;
            String[] parts = line.trim().split("\\s+|,");
            int[] row = Arrays.stream(parts)
                    .filter(p -> !p.isEmpty())
                    .mapToInt(Integer::parseInt)
                    .toArray();
            rows.add(row);
        }

        if (isMatrix) {
            int n = rows.size();
            int m = n > 0 ? rows.get(0).length : 0;
            setupTable(table, n, m);
            for (int i = 0; i < n; i++) {
                ObservableList<IntegerProperty> rowProps = table.getItems().get(i);
                for (int j = 0; j < m; j++) {
                    rowProps.get(j).set(rows.get(i)[j]);
                }
            }
            if (table == tableMB) {
                rowsMB.setText(String.valueOf(n));
                colsMB.setText(String.valueOf(m));
            } else if (table == tableMC) {
                rowsMC.setText(String.valueOf(n));
                colsMC.setText(String.valueOf(m));
            } else if (table == tableMD) {
                rowsMD.setText(String.valueOf(n));
                colsMD.setText(String.valueOf(m));
            }
        } else {
            int[] arr = !rows.isEmpty() ? rows.get(0) : new int[0];
            setupTable(table, 1, arr.length);
            ObservableList<IntegerProperty> rowProps = table.getItems().get(0);
            for (int j = 0; j < arr.length; j++) {
                rowProps.get(j).set(arr[j]);
            }
            if (table == tableB) {
                lengthB.setText(String.valueOf(arr.length));
            } else if (table == tableZ) {
                lengthZ.setText(String.valueOf(arr.length));
            }
        }
    } catch (Exception ex) {
        showError("Помилка завантаження: " + ex.getMessage());
    }
}
```

```
private void setLanguage(String langTag) {
    Locale locale = Locale.forLanguageTag(langTag);
    bundle = ResourceBundle.getBundle("lang", locale);
    updateLabels();
}

private void updateLabels() {
    lengthLabelB.setText(bundle.getString("prompt.n"));
    lengthLabelZ.setText(bundle.getString("prompt.n"));

    fileMenu.setText(bundle.getString("menu.file"));
    helpMenu.setText(bundle.getString("menu.help"));

    inputTab.setText(bundle.getString("tab.input"));
    resultTab.setText(bundle.getString("tab.result"));
    settingsTab.setText(bundle.getString("tab.settings"));

    saveTxtMenuItem.setText(bundle.getString("menuitem.save_txt"));
    exitMenuItem.setText(bundle.getString("menuitem.exit"));

    aboutMenuItem.setText(bundle.getString("menuitem.about"));

    operationLabel.setText(bundle.getString("label.operation"));
    calculateButton.setText(bundle.getString("button.calculate"));

    bLabel.setText(bundle.getString("label.vector_B"));
    lengthB.setPromptText(bundle.getString("prompt.n"));
    createB.setText(bundle.getString("button.create_B"));
    loadBButton.getTooltip().setText(bundle.getString("tooltip.load_B"));

    mbLabel.setText(bundle.getString("label.matrix_MB"));
    rowsMB.setPromptText(bundle.getString("prompt.n"));
    colsMB.setPromptText(bundle.getString("prompt.m"));
    createMB.setText(bundle.getString("button.create_MB"));
    loadMBButton.getTooltip().setText(bundle.getString("tooltip.load_MB"));

    mcLabel.setText(bundle.getString("label.matrix_MC"));
    rowsMC.setPromptText(bundle.getString("prompt.n"));
    colsMC.setPromptText(bundle.getString("prompt.m"));
    createMC.setText(bundle.getString("button.create_MC"));
    loadMCButton.getTooltip().setText(bundle.getString("tooltip.load_MC"));

    mdLabel.setText(bundle.getString("label.matrix_MD"));
    rowsMD.setPromptText(bundle.getString("prompt.n"));
    colsMD.setPromptText(bundle.getString("prompt.m"));
    createMD.setText(bundle.getString("button.create_MD"));
    loadMDButton.getTooltip().setText(bundle.getString("tooltip.load_MD"));

    zLabel.setText(bundle.getString("label.vector_Z"));
    lengthZ.setPromptText(bundle.getString("prompt.n"));
    createZ.setText(bundle.getString("button.create_Z"));
    loadZButton.getTooltip().setText(bundle.getString("tooltip.load_Z"));

    resultLabel.setText(bundle.getString("label.result"));
    copyButton.setText(bundle.getString("button.copy"));
    clearButton.setText(bundle.getString("button.clear"));
    stepsLabel.setText(bundle.getString("label.steps"));

    showStepsCheck.setText(bundle.getString("checkbox.show_steps"));
    darkThemeCheck.setText(bundle.getString("checkbox.dark_theme"));
    languageLabel.setText(bundle.getString("label.interface_language"));
```

```
    fontSizeTitle.setText(bundle.getString("label.font_size"));
    threadCountTitle.setText(bundle.getString("label.thread_size"));
    autoSaveCheck.setText(bundle.getString("checkbox.auto_save"));
    confirmExitCheck.setText(bundle.getString("checkbox.confirm_exit"));
    soundNotifyCheck.setText(bundle.getString("checkbox.sound_notify"));

    fontSizeLabel.setText(String.valueOf((int) fontSizeSlider.getValue()));
    threadCountLabel.setText(String.valueOf((int) threadCountSlider.getValue()));

    List<String> langKeys = List.of("language.ukrainian", "language.english");
    ObservableList<String> languages = FXCollections.observableArrayList();
    for (String key : langKeys) languages.add(bundle.getString(key));
    languageCombo.setItems(languages);

    String langCode = loadLanguage();
    int idx = (langCode != null && this.langCodes.contains(langCode)) ? this.langCodes.indexOf(langCode) : 0;
    languageCombo.getSelectionModel().select(idx);

    updateOperationsCombo();
  }

  private void updateOperationsCombo() {
    List<String> localized = opKeys.stream()
        .map(bundle::getString)
        .toList();
    operationCombo.setItems(FXCollections.observableArrayList(localized));
    operationCombo.getSelectionModel().selectFirst();
  }

  private void handleExit() {
    if (confirmExitCheck.isSelected()) {
      Alert alert = new Alert(
          Alert.AlertType.CONFIRMATION,
          bundle.getString("alert.confirm_exit"),
          ButtonType.YES, ButtonType.NO
      );
      if (alert.showAndWait().orElse(ButtonType.NO) == ButtonType.NO) {
        return;
      }
    }
    Platform.exit();
  }

  private void onSaveTxt() {
    Window win = tableMB.getScene().getWindow();
    FileChooser fc = new FileChooser();
    fc.setTitle(bundle.getString("dialog.save_txt"));
    fc.getExtensionFilters().add(
        new FileChooser.ExtensionFilter("Text Files", "*.txt")
    );
    File file = fc.showSaveDialog(win);
    if (file == null) return;
    try (BufferedWriter w = new BufferedWriter(new FileWriter(file))) {
      w.write(bundle.getString("label.result") + ":\n");
      w.write(resultArea.getText() + "\n\n");
      w.write(bundle.getString("label.steps") + ":\n");
      for (String step : stepsList.getItems()) {
        w.write(step + "\n");
      }
    } catch (Exception ex) {
      showError(bundle.getString("error.save_txt") + ": " + ex.getMessage());
    }
```

```
}

private void showAboutDialog() {
   String title = bundle.getString("about.title");
   String header = bundle.getString("about.header");
   String content = bundle.getString("about.content");
   Alert alert = new Alert(Alert.AlertType.INFORMATION);
   alert.setTitle(title);
   alert.setHeaderText(header);
   alert.setContentText(content);
   alert.showAndWait();
}

private void saveLanguage(String lang) {
   try {
      Properties props = new Properties();
      props.setProperty("language", lang);
      try (FileWriter writer = new FileWriter(SETTINGS_FILE)) {
         props.store(writer, "User settings");
      }
   } catch (Exception ignored) {
   }
}

private String loadLanguage() {
   try {
      Properties props = new Properties();
      File file = new File(SETTINGS_FILE);
      if (!file.exists()) return null;
      try (FileReader reader = new FileReader(file)) {
         props.load(reader);
         return props.getProperty("language");
      }
   } catch (Exception ignored) {
   }
   return null;
}

@FXML
private void onCalculate() {
   try {
      int opIndex = operationCombo.getSelectionModel().getSelectedIndex();
      String opKey = opKeys.get(opIndex);

      List<String> steps = new ArrayList<>();
      String resultText = null;

      long t0 = 0;
      if (opKey.equals("operation.prg1")) {
         // Зчитуємо довжину
         int N = parseSafe(lengthB.getText());
         int nMB = parseSafe(rowsMB.getText()), mMB = parseSafe(colsMB.getText());
         int nMC = parseSafe(rowsMC.getText()), mMC = parseSafe(colsMC.getText());

         // Зчитуємо дані
         int[] B = Arrays.stream(readMatrix(tableB, 1, N)[0]).toArray();
         int[][] MB = readMatrix(tableMB, nMB, mMB);
         int[][] MC = readMatrix(tableMC, nMC, mMC);

         // Запускаємо таймер
         t0 = System.nanoTime();
```

```
            // Запускаємо PRG1
            if (showStepsCheck.isSelected()) {
                resultText = Arrays.toString(PRG1.prg1(B, MB, MC, getP(), steps, bundle));
            } else {
                resultText = Arrays.toString(PRG1.prg1(B, MB, MC, getP(), null, bundle));
            }
        }
        if (opKey.equals("operation.prg2")) {
            // Зчитуємо довжину
            int N = parseSafe(lengthZ.getText());
            int nMB = parseSafe(rowsMB.getText()), mMB = parseSafe(colsMB.getText());
            int nMC = parseSafe(rowsMC.getText()), mMC = parseSafe(colsMC.getText());
            int nMD = parseSafe(rowsMD.getText()), mMD = parseSafe(colsMD.getText());

            // Зчитуємо дані
            int[] Z = Arrays.stream(readMatrix(tableZ, 1, N)[0]).toArray();
            int[][] MB = readMatrix(tableMB, nMB, mMB);
            int[][] MC = readMatrix(tableMC, nMC, mMC);
            int[][] MD = readMatrix(tableMD, nMD, mMD);

            // Запускаємо таймер
            t0 = System.nanoTime();

            // Запускаємо PRG2
            if (showStepsCheck.isSelected()) {
                resultText = MatrixUtils.matrixToString(PRG2.prg2(MB, MC, MD, Z, getP(), steps, bundle));
            } else {
                resultText = MatrixUtils.matrixToString(PRG2.prg2(MB, MC, MD, Z, getP(), null, bundle));
            }
        }
        if (resultText == null) {
            throw new IllegalStateException("Непідтримувана операція: " + opKey);
        }
        long elapsed = (System.nanoTime() - t0) / 1_000_000; // в мс

        resultArea.setText(resultText);
        if (showStepsCheck.isSelected()) {
            stepsList.setItems(FXCollections.observableArrayList(steps));
        } else {
            stepsList.getItems().clear();
        }

        Alert info = new Alert(Alert.AlertType.INFORMATION);
        info.setTitle(bundle.getString("alert.calc_done"));

        String opLabel = bundle.getString("alert.operation")
                .replace("{0}", String.valueOf(operationCombo.getSelectionModel().getSelectedItem()));
        String timeLabel = bundle.getString("alert.time")
                .replace("{0}", String.valueOf(elapsed));

        info.setHeaderText(opLabel);
        info.setContentText(timeLabel);
        info.showAndWait();

        if (autoSaveCheck.isSelected()) {
            onSaveTxt();
        }
        if (soundNotifyCheck.isSelected()) {
            Toolkit.getDefaultToolkit().beep();
        }

    } catch (NumberFormatException ex) {
```

```
            showError(bundle.getString("error.invalid_input"));
        } catch (Exception ex) {
            String msg = bundle.containsKey("error.general")
                    ? bundle.getString("error.general")
                    : "Error";
            showError(msg + ": " + ex.getMessage());
        }
    }

    private int[][] readMatrix(TableView<ObservableList<IntegerProperty>> table, int n, int m) {
        int[][] a = new int[n][m];
        ObservableList<?> rows = table.getItems();
        for (int i = 0; i < n; i++) {
            ObservableList<?> row = (ObservableList<?>) rows.get(i);
            for (int j = 0; j < m; j++) {
                Object val = row.get(j);
                a[i][j] = (val instanceof IntegerProperty dp) ? dp.get() : Integer.parseInt(val.toString());
            }
        }
        return a;
    }

    private void updateInputPanels(String op) {
        boolean isA = bundle.getString("operation.prg1").equals(op);
        boolean isMA = bundle.getString("operation.prg2").equals(op);
        boolean multi = isA || isMA;

        multiMatrixPane.setVisible(multi);
        multiMatrixPane.setManaged(multi);

        blockB.setVisible(isA);
        blockB.setManaged(isA);
        blockMB.setVisible(isA || isMA);
        blockMB.setManaged(isA || isMA);
        blockMC.setVisible(isA || isMA);
        blockMC.setManaged(isA || isMA);
        blockMD.setVisible(isMA);
        blockMD.setManaged(isMA);
        blockZ.setVisible(isMA);
        blockZ.setManaged(isMA);
    }

    private int parseSafe(String s) {
        try {
            return Integer.parseInt(s.trim());
        } catch (Exception e) {
            return 0;
        }
    }

    private void showError(String msg) {
        Alert alert = new Alert(Alert.AlertType.ERROR, msg, ButtonType.OK);
        alert.showAndWait();
    }

    private int getP() {
        return P;
    }

    private void setP(int newP) {
        P = newP;
    }
```

```
}
```

module-info.java

```
module com.example.matrixcalc {
    requires javafx.controls;
    requires javafx.fxml;
    requires org.apache.pdfbox;
    requires java.desktop;


    opens com.example.matrixcalc to javafx.fxml;
    exports com.example.matrixcalc;
    opens com.example.matrixcalc.controller to javafx.fxml;
}
```

Main.fxml

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.collections.FXCollections?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import java.lang.String?>

<BorderPane xmlns:fx="http://javafx.com/fxml"
        fx:controller="com.example.matrixcalc.controller.MainController">

  <!-- Меню -->
  <top>
    <MenuBar>
      <Menu fx:id="fileMenu" text="%menu.file">
        <MenuItem fx:id="saveTxtMenuItem" text="%menuitem.save_txt"/>
        <MenuItem fx:id="exitMenuItem" text="%menuitem.exit"/>
      </Menu>
      <Menu fx:id="helpMenu" text="%menu.help">
        <MenuItem fx:id="aboutMenuItem" text="%menuitem.about"/>
      </Menu>
    </MenuBar>
  </top>

  <!-- Основний контент -->
  <center>
    <TabPane fx:id="mainTabPane">

      <!-- Вкладка вводу -->
      <Tab fx:id="inputTab" text="%tab.input">
        <VBox spacing="16">
          <padding>
            <Insets top="15" right="15" bottom="15" left="15"/>
          </padding>

          <!-- Окрема матриця -->
          <HBox spacing="12" alignment="CENTER_LEFT">
            <Label fx:id="operationLabel" text="%label.operation"/>
            <ComboBox fx:id="operationCombo" prefWidth="320"/>
            <Button fx:id="calculateButton" text="%button.calculate"/>
          </HBox>
          <VBox fx:id="singleMatrixPane" spacing="8">
            <HBox spacing="8" alignment="CENTER_LEFT">
              <Label fx:id="rowsLabel" text="%label.rows"/>
              <TextField fx:id="rowsField" prefWidth="50" promptText="%prompt.n"/>
              <Label fx:id="colsLabel" text="%label.cols"/>
              <TextField fx:id="colsField" prefWidth="50" promptText="%prompt.m"/>
              <Button fx:id="createTableButton" text="%button.create"/>
              <Button fx:id="loadMatrixButton" text="📂">
                <tooltip>
                  <Tooltip text="%tooltip.load_matrix"/>
                </tooltip>
              </Button>
            </HBox>
            <TableView fx:id="matrixTable" prefHeight="200"/>
          </VBox>

          <!-- Мультиматриці / вектори -->
          <VBox fx:id="multiMatrixPane" spacing="12" visible="false" managed="false">
            <!-- Вектор B -->
            <VBox fx:id="blockB" spacing="4">
```

```
    <HBox spacing="8" alignment="CENTER_LEFT">
      <Label fx:id="bLabel" text="%label.vector_B"/>
      <Label fx:id="lengthLabelB" text="%label.length"/>
      <TextField fx:id="lengthB" prefWidth="60" promptText="%prompt.length"/>
      <Button fx:id="createB" text="%button.create_B"/>
      <Button fx:id="loadBButton" text="📂">
        <tooltip>
          <Tooltip text="%tooltip.load_B"/>
        </tooltip>
      </Button>
    </HBox>
    <TableView fx:id="tableB" prefHeight="80"/>
  </VBox>

  <!-- Матриця MB -->
  <VBox fx:id="blockMB" spacing="4">
    <HBox spacing="8" alignment="CENTER_LEFT">
      <Label fx:id="mbLabel" text="%label.matrix_MB"/>
      <Label text="n × m"/>
      <TextField fx:id="rowsMB" prefWidth="40" promptText="n"/>
      <Label text="×"/>
      <TextField fx:id="colsMB" prefWidth="40" promptText="m"/>
      <Button fx:id="createMB" text="%button.create_MB"/>
      <Button fx:id="loadMBButton" text="📂">
        <tooltip>
          <Tooltip text="%tooltip.load_MB"/>
        </tooltip>
      </Button>
    </HBox>
    <TableView fx:id="tableMB" prefHeight="80"/>
  </VBox>

  <!-- Матриця MC -->
  <VBox fx:id="blockMC" spacing="4">
    <HBox spacing="8" alignment="CENTER_LEFT">
      <Label fx:id="mcLabel" text="%label.matrix_MC"/>
      <Label text="n × m"/>
      <TextField fx:id="rowsMC" prefWidth="40" promptText="n"/>
      <Label text="×"/>
      <TextField fx:id="colsMC" prefWidth="40" promptText="m"/>
      <Button fx:id="createMC" text="%button.create_MC"/>
      <Button fx:id="loadMCButton" text="📂">
        <tooltip>
          <Tooltip text="%tooltip.load_MC"/>
        </tooltip>
      </Button>
    </HBox>
    <TableView fx:id="tableMC" prefHeight="80"/>
  </VBox>

  <!-- Матриця MD -->
  <VBox fx:id="blockMD" spacing="4">
    <HBox spacing="8" alignment="CENTER_LEFT">
      <Label fx:id="mdLabel" text="%label.matrix_MD"/>
      <Label text="n × m"/>
      <TextField fx:id="rowsMD" prefWidth="40" promptText="n"/>
      <Label text="×"/>
      <TextField fx:id="colsMD" prefWidth="40" promptText="m"/>
      <Button fx:id="createMD" text="%button.create_MD"/>
      <Button fx:id="loadMDButton" text="📂">
        <tooltip>
          <Tooltip text="%tooltip.load_MD"/>
```

```
                </tooltip>
              </Button>
            </HBox>
            <TableView fx:id="tableMD" prefHeight="80"/>
          </VBox>

          <!-- Масив Z (вектор) -->
          <VBox fx:id="blockZ" spacing="4">
            <HBox spacing="8" alignment="CENTER_LEFT">
              <Label fx:id="zLabel" text="%label.vector_Z"/>
              <Label fx:id="lengthLabelZ" text="%label.length"/>
              <TextField fx:id="lengthZ" prefWidth="60" promptText="%prompt.length"/>
              <Button fx:id="createZ" text="%button.create_Z"/>
              <Button fx:id="loadZButton" text="📂">
                <tooltip>
                  <Tooltip text="%tooltip.load_Z"/>
                </tooltip>
              </Button>
            </HBox>
            <TableView fx:id="tableZ" prefHeight="80"/>
          </VBox>
        </VBox>
      </VBox>
    </Tab>

    <Tab fx:id="resultTab" text="%tab.result">
      <VBox spacing="10">
        <padding>
          <Insets top="15" right="15" bottom="15" left="15"/>
        </padding>
        <HBox spacing="8" alignment="CENTER_RIGHT" styleClass="result-toolbar">
          <Button fx:id="copyButton" text="%button.copy" styleClass="action-button"/>
          <Button fx:id="clearButton" text="%button.clear" styleClass="action-button"/>
        </HBox>
        <ScrollPane fitToWidth="true" fitToHeight="true">
          <VBox spacing="12">
            <Label fx:id="resultLabel" text="%label.result" styleClass="section-label"/>
            <TextArea fx:id="resultArea" editable="false" prefRowCount="4" styleClass="result-area"/>
            <Label fx:id="stepsLabel" text="%label.steps" styleClass="section-label"/>
            <ListView fx:id="stepsList" styleClass="steps-container" prefHeight="200"/>
          </VBox>
        </ScrollPane>
      </VBox>
    </Tab>

    <!-- Вкладка налаштувань -->
    <Tab fx:id="settingsTab" text="%tab.settings">
      <VBox spacing="12">
        <padding>
          <Insets top="15" right="15" bottom="15" left="15"/>
        </padding>
        <CheckBox fx:id="showStepsCheck" text="%checkbox.show_steps"/>
        <CheckBox fx:id="darkThemeCheck" text="%checkbox.dark_theme"/>

        <HBox spacing="10" alignment="CENTER_LEFT">
          <Label fx:id="languageLabel" text="%label.interface_language"/>
          <ComboBox fx:id="languageCombo" prefWidth="120">
            <items>
              <FXCollections fx:factory="observableArrayList">
                <String fx:value="Українська"/>
                <String fx:value="English"/>
              </FXCollections>
```

```
                </items>
              </ComboBox>
            </HBox>

            <HBox spacing="10" alignment="CENTER_LEFT">
              <Label fx:id="fontSizeTitle" text="%label.font_size"/>
              <Slider fx:id="fontSizeSlider" min="10" max="24" value="13" blockIncrement="1"/>
              <Label fx:id="fontSizeLabel" text="13"/>
            </HBox>

            <CheckBox fx:id="autoSaveCheck" text="%checkbox.auto_save"/>
            <CheckBox fx:id="confirmExitCheck" text="%checkbox.confirm_exit"/>
            <CheckBox fx:id="soundNotifyCheck" text="%checkbox.sound_notify"/>

          </VBox>
        </Tab>

      </TabPane>
    </center>
  </BorderPane>
```