НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК 004.2

До захисту допущено:
В.О. Завідувача кафедри
_____ Михайло Новотарський
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Комп'ютерні системи та мережі» зі спеціальності 123 «Комп'ютерна інженерія»

на тему «Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри»

Виконав:
студент ІІ курсу, групи ІО-32мп
Андрейцов Ярослав Євгенович _____

Керівник:
доцент к.т.н.
Корочкін Олександр Володимирович
_____

Консультант з нормоконтролю:
асистент каф. ОТ, к.т.н.
Кулаков Олексій Юрійович _____

Рецензент:
доцент, к.т.н., доцент,
Тарасенко-Клятченко Оксана Володимірівна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет/ННІ Факультет інформатики та обчислювальної техніки
(повна назва)

Кафедра Кафедра обчислювальної техніки
(повна назва)

Рівень вищої – освіти – другий (магістерський)

Спеціальність, спеціалізація 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

В. О. Завідувача кафедри
____ Михайло Новотарський

«__»_____ 2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Андрейцовому Ярославові Євгеновичу
(прізвище ім'я по-батькові)

1. Тема проекту " Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри "
науковий керівник дисертації Корочкін Олександр Володимирович доцент к.т.н. ,
затверджені наказом по університету від « 08 » листопада 2024р. № 5016с
2. Термін подання студентом дисертації: _____
3. Об'єкт дослідження: Високопродуктивний програмно-апаратний комплекс
4. Вихідні дані: _____
5. Перелік завдань, які потрібно розробити: огляд існуючих рішень для високопродуктивних програмно-апаратних комплексів, розробка апаратного забезпечення, розробка програмного забезпечення
6. Орієнтовний перелік графічного (ілюстративного) матеріалу: структурна схема роботи програми, структурна схема потоків, ілюстрації по темі роботи
7. Орієнтовний перелік публікацій: _____

8. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
нормоконтроль	асист. Кулаков О. Ю.		

9. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Огляд існуючих рішень	05.09.2024 - 12.09.2024	
2	Планування розробки	13.09.2024 - 19.09.2024	
3	Вибір технологій для розробки	20.09.2024 - 22.09.2024	
4	Реалізація системи	23.09.2024 - 31.10.2024	
5	Тестування системи	1.11.2024 - 06.11.2024	
6	Розрахунки стартапу	07.11.2024 - 14.11.2024	
7	Оформлення документації	15.11.2024 - 25.11.2024	

Студент _____ Ярослав Андрейцов
(підпис)

Науковий керівник _____Олександр Корочкін
(підпис)

РЕФЕРАТ

Магістерська дисертація присвячена дослідженню та розробці високопродуктивного програмно-апаратного комплексу (ПАК) з використанням сучасних технологій паралельного програмування. Основною метою роботи є створення ефективного рішення для виконання складних обчислювальних завдань із забезпеченням оптимального використання апаратних ресурсів.

У роботі проведено огляд сучасних підходів до розробки багатопотокових програм, аналіз програмних інструментів та мов програмування, що підтримують паралельне виконання. Розглянуто особливості синхронізації потоків із застосуванням різних механізмів, таких як семафори, м’ютекси, монітори та бар’єри.

Особливу увагу приділено аналізу апаратних компонентів, зокрема сучасних процесорів від компаній Intel та AMD. Вивчено їх архітектурні особливості, порівняно продуктивність та визначено оптимальні рішення для застосування у ПАК..

У практичній частині проведено архітектурне проєктування ПАК, здійснено вибір компонентів та їх інтеграцію. Розроблено програмне забезпечення з використанням сучасних інструментів для багатопотокового програмування, а саме OpenMP.

Результати тестування показали, що розроблений ПАК забезпечує високу продуктивність та стабільність у виконанні складних обчислювальних завдань. Дана робота демонструє ефективність інтегрованих рішень і закладає основу для подальшого розвитку високопродуктивних систем.

ABSTRACT

The master's thesis is devoted to the research anddevelopment of a high-performance hardware and software system (HPS) with using modern parallel programming technologies. The main purpose of the work of the work is to create an effective solution for performing complex computational tasks with optimal utilization of hardware resources.

The paper reviews modern approaches to the development of multithreaded programs, analysis of software tools and programming languages, that support parallel execution. The features of synchronization of threads using various mechanisms, such as semaphores, mutexes, monitors, and barriers.

Particular attention is paid to the analysis of hardware components, in particular, modern processors from Intel and AMD. We have studied their architectural features, performance was compared, and the optimal solutions for use in PACs are determined.

In the practical part, the architectural architectural design of the SAR, selection of components and their integration. We have developed software using modern tools for for multithreaded programming, namely OpenMP.

The test results showed that the developed PAC provides high performance and stability in performing complex computational tasks. This work demonstrates the effectiveness of integrated solutions and lays the foundation for further development of high-performance system

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ПАК – програмно апаратний комплекс

GPU – graphics processing unit

ARM – Advanced RISC Machine

AI – artificial intelligence

AMD – Advanced Micro Devices

XFR – Extended Frequency Range

SMT – Simultaneous Multithreading

AWS – Amazon Web Services

TPL – Task Parallel Library

JVM – Java Virtual Machine

API – application programming interface

TDP – thermal design power

ATX – Advanced Technology Extended

ПЗ – Програмне забезпечення

РОЗДІЛ 1
АНАЛІЗ АПАРАТНИХ РІШЕНЬ ДЛЯ ПОБУДОВИ ВИСОКОПРОДУКТИВНИХ ПАК

1.1. Тенденції розвитку сучасних процесорів

Процесори є одним із ключових елементів комп'ютерних систем, які визначають їх продуктивність і здатність виконувати складні обчислювальні завдання. Через постійний розвиток технологій і зростаючі вимоги до продуктивності, тенденції розвитку сучасних процесорів стають особливо важливими для розуміння майбутніх напрямків у розробці обчислювальних систем. У цьому підрозділі ми розглянемо основні тенденції, які впливають на проектування та виробництво процесорів, а також їхні впливи на продуктивність і енергоефективність.

Мультиядерні архітектури

Однією з основних тенденцій розвитку сучасних процесорів є перехід до мультиядерних архітектур. На відміну від одноядерних процесорів, які можуть виконувати лише один потік інструкцій за раз, мультиядерні процесори мають кілька ядер. Це дозволяє їм одночасно обробляти декілька потоків, що значно підвищує загальну продуктивність системи, особливо для багатопотокових додатків.

Переваги мультиядерності:

- **Паралельна обробка**: Користувачі можуть запускати кілька програм одночасно без зниження продуктивності.
- **Оптимізація під задачі**: Сучасні додатки, такі як програми для обробки відео, моделювання або машинне навчання, можуть

ефективно використовувати всі ядра, що дозволяє скоротити час обробки [1].

Виклики:

- **Ускладненість програмування**: Розробка програмного забезпечення, яке може ефективно використовувати всі доступні ядра, потребує спеціальних знань і навичок.
- **Проблеми з синхронізацією**: Зі збільшенням кількості потоків зростають і вимоги до механізмів синхронізації, що може призвести до зниження продуктивності [1].

Зменшення розміру транзисторів

Технології виготовлення процесорів постійно вдосконалюються, що дозволяє зменшувати розмір транзисторів. Зменшення розміру транзисторів не лише підвищує щільність інтеграції, але й знижує енергоспоживання, оскільки менші транзистори споживають менше енергії.

Тенденції у виробництві:

- **Технологічні процеси**: Від 14 нм до 7 нм та менше, нові технологічні процеси дозволяють виробляти мікросхеми з меншими втратами енергії і вищою продуктивністю.
- **Збільшення кількості транзисторів**: Висока щільність транзисторів у сучасних процесорах дозволяє інтегрувати більше функцій на одному кристалі, що сприяє підвищенню загальної продуктивності [2].

Вплив на енергоспоживання:

- Зменшення розміру транзисторів сприяє підвищенню енергоефективності, що є критично важливим у сучасних обчислювальних системах, де енергетичні витрати стають важливим чинником [2].

Інтеграція графічних процесорів (GPU)

Сучасні процесори все частіше містять інтегровані графічні процесори, що дозволяє покращити графічну продуктивність без необхідності в окремій графічній карті. Це є особливо актуальним для ноутбуків і компактних систем, де обмежений простір і енергоспоживання.

Переваги інтеграції GPU:

- **Зменшення витрат на енергію**: Інтеграція GPU в процесор зменшує енергоспоживання, оскільки дозволяє зменшити кількість компонентів, які потрібно живити.
- **Поліпшення графічної продуктивності**: Сучасні інтегровані графічні рішення здатні виконувати більш складні обчислення, що робить їх придатними для ігор та графічних додатків.

Виклики:

- **Обмежені можливості**: Інтегровані графічні процесори зазвичай мають нижчу продуктивність у порівнянні з дискретними графічними картами, що обмежує їх використання у вимогливих графічних додатках [1,2].

Енергоефективність і терморегулювання

Зі зростанням обчислювальних потужностей підвищуються вимоги до енергоефективності. Розробники процесорів постійно шукають способи зменшення споживання енергії та поліпшення теплових характеристик.

Основні підходи:

- **Динамічне регулювання частоти**: Сучасні процесори здатні адаптувати свою тактову частоту в залежності від навантаження, що дозволяє зменшити споживання енергії під час простою.
- **Управління живленням**: Інтегровані технології для управління живленням допомагають оптимізувати енергоспоживання, що є особливо важливим для мобільних пристроїв.

Теплове управління:

- **Системи охолодження**: Збільшення потужностей процесорів вимагає нових рішень у системах охолодження, таких як рідинне охолодження або новітні технології терморегуляції [2].

Архітектурні інновації

Нові архітектури процесорів, такі як ARM, RISC-V та інші, впливають на ринок, пропонуючи альтернативи традиційним x86 архітектурам. Ці архітектури спроектовані для певних типів завдань і можуть запропонувати високу продуктивність при низькому споживанні енергії.

Перспективи нових архітектур:

- **Спеціалізовані обчислення**: Архітектури, що оптимізовані для конкретних завдань (наприклад, AI або обробка зображень), можуть забезпечити вищу продуктивність у своїй категорії.
- **Відкриті архітектури**: Відкриті архітектури, такі як RISC-V, пропонують можливість модифікації та вдосконалення, що дозволяє різним компаніям адаптувати процесори під свої специфічні вимоги [3].

1.2. Огляд процесорів від Intel

Компанія Intel є одним із найбільших і найвідоміших виробників мікропроцесорів у світі. Її продукти використовуються в різноманітних обчислювальних системах — від персональних комп’ютерів до потужних серверів. В даному підрозділі ми розглянемо основні серії процесорів Intel, зокрема Intel Xeon, Intel Core і Intel Core Extreme/X-series, акцентуючи увагу на їхніх архітектурних особливостях, призначенні та продуктивності.

Серія Intel Xeon: архітектурні особливості та призначення

Процесори серії Intel Xeon призначені для використання в серверах, робочих станціях і системах з високими вимогами до продуктивності. Вони відрізняються від звичайних споживчих процесорів наявністю розширених можливостей для обробки даних, більшою кількістю ядер, кеш-пам’яті та підтримкою технологій, що забезпечують високу надійність і масштабованість [4].

Основні архітектурні особливості:

1. **Багатоядерність**: Процесори Xeon зазвичай мають від 4 до 56 ядер, що дозволяє одночасно обробляти велику кількість потоків, що є критично важливим для сервісів, які потребують високої продуктивності.
2. **Підтримка великого обсягу пам’яті**: Xeon підтримує до 6 ТБ оперативної пам’яті в залежності від архітектури, що робить їх ідеальними для завдань, що вимагають обробки великих обсягів даних, таких як бази даних або аналітика.
3. **Паралелізм і масштабованість**: Технології, такі як Intel Turbo Boost та Hyper-Threading, забезпечують оптимізацію використання ядер, що підвищує ефективність паралельних обчислень [4].

Призначення:

Процесори серії Xeon часто використовуються в:

- **Серверах**: для обробки запитів від користувачів і виконання серверних додатків.
- **Наукових обчисленнях**: для розрахунків в наукових дослідженнях та моделях.
- **Віртуалізації**: завдяки своїй масштабованості та можливості обробки кількох віртуальних машин одночасно [4].

Серія Intel Core та її роль у ПАК

Серія Intel Core є однією з найбільш популярних лінійок процесорів для персональних комп'ютерів, ноутбуків і робочих станцій. Вона включає кілька підсерій, таких як Core i3, i5, i7 та i9, які відрізняються за потужністю та функціональними можливостями.

Основні архітектурні особливості:

1. **Гнучкість у використанні**: Intel Core забезпечує баланс між потужністю та енергоспоживанням, що робить їх універсальними для різних застосувань — від офісної роботи до геймінгу.
2. **Технології Turbo Boost і Hyper-Threading**: Ці технології дозволяють процесорам Core автоматично підвищувати частоту в умовах високого навантаження, забезпечуючи підвищену продуктивність без значного збільшення енергоспоживання.
3. **Вбудовані графічні процесори**: Сучасні Intel Core мають інтегровані графічні рішення, що дозволяє користувачам виконувати базові графічні завдання без потреби у дискретній графічній карті.

Роль у ПАК:

У контексті високопродуктивних програмно-апаратних комплексів (ПАК), серія Intel Core виконує такі функції:

- **Обробка даних**: Використовується для обробки і виконання обчислень, що є критично важливими для реалізації специфічних алгоритмів.
- **Графічні обчислення**: Здатність інтегрованих графічних процесорів забезпечує необхідну продуктивність для графічних інтерфейсів та візуалізації даних.
- **Гнучкість у розробці**: Завдяки широкій доступності та підтримці різних технологій, процесори Intel Core стають вибором для розробників програмного забезпечення, які прагнуть до оптимізації своїх рішень [5].

Серія Intel Core Extreme/X-series: продуктивність і перспективи

Серія Intel Core Extreme (також відома як X-series) була розроблена для ентузіастів і професіоналів, які потребують максимальної продуктивності. Ці процесори пропонують високу кількість ядер та потоків, що робить їх ідеальними для ресурсомістких завдань, таких як рендеринг, відеомонтаж і 3D-моделювання [4].

Основні архітектурні особливості:

1. **Висока продуктивність**: Процесори X-series зазвичай мають від 8 до 18 ядер, що дозволяє досягати вражаючих результатів у багатопотокових застосуваннях.
2. **Можливості розгону**: Багато моделей серії Extreme підтримують можливість розгону, що дозволяє користувачам збільшувати тактову частоту для досягнення максимальної продуктивності.
3. **Розширені кеш-пам’яті**: Процесори X-series оснащені великою кількістю кеш-пам’яті, що забезпечує швидкий доступ до часто використовуваних даних і підвищує загальну швидкість виконання програм.

Перспективи:

- **Тенденції розвитку**: Серія X-series продовжує розвиватися, впроваджуючи нові технології, такі як підтримка новітніх стандартів пам’яті (DDR4, DDR5) та комунікаційних технологій (PCIe 4.0 і 5.0), що дозволяє зберігати актуальність у швидко змінюваному ринку.
- **Зростаючий попит**: Зростання популярності високоякісного контенту, геймінгу та професійного програмного забезпечення стимулює попит на такі потужні рішення, як X-series, що підвищує їхню значимість у розробці високопродуктивних комплексів [4].

1.3. Огляд процесорів від AMD

Компанія AMD (Advanced Micro Devices) є одним із основних конкурентів Intel на ринку мікропроцесорів. З моменту свого заснування в 1969 році, AMD постійно вдосконалює свої технології, пропонуючи інноваційні рішення для різноманітних обчислювальних потреб. У цьому підрозділі ми розглянемо основні серії процесорів AMD: EPYC, Ryzen та Ryzen Threadripper, акцентуючи увагу на їхніх архітектурних особливостях і призначенні.

Серія EPYC: рішення для серверних систем

Процесори EPYC від AMD були спеціально розроблені для використання в серверних системах та центрах обробки даних. Завдяки своїй архітектурі Zen та інноваційним технологіям, EPYC забезпечує високу продуктивність, енергоефективність і надійність.

Основні архітектурні особливості:

1. **Багатоядерність**: Процесори EPYC зазвичай мають від 8 до 64 ядер, що дозволяє одночасно обробляти велику кількість запитів. Це особливо важливо для віртуалізації та обробки даних у реальному часі.
2. **Підтримка великої обсягу пам’яті**: Серія EPYC може підтримувати до 2 ТБ оперативної пам’яті, що робить їх ідеальними для великих серверних систем та завдань, що потребують значних обсягів пам’яті.
3. **Технологія Infinity Fabric**: Дана технологія забезпечує швидке з’єднання між ядрами, кеш-пам’яттю та іншими компонентами, що підвищує загальну продуктивність системи.

Призначення:

Процесори EPYC активно використовуються в:

- **Серверах**: для обробки великих обсягів даних, таких як бази даних та аналітика.

- **Центрах обробки даних**: завдяки своїй масштабованості та продуктивності, EPYC стають вибором для хмарних обчислень і віртуалізації.
- **Наукових обчисленнях**: їх потужність дозволяє виконувати складні обчислення, такі як моделювання та симуляції [6].

Серія Ryzen: багатозадачність для загальних потреб

Процесори серії Ryzen призначені для широкого спектра застосувань — від домашніх комп'ютерів до професійних робочих станцій. Завдяки своїй архітектурі Zen, Ryzen забезпечують високу продуктивність і енергоефективність.

Основні архітектурні особливості:

1. **Модульна архітектура**: Ryzen використовує модульну архітектуру, що дозволяє розміщувати кілька ядер у кожному чипі, що забезпечує високу продуктивність у багатопотокових задачах.
2. **Технології Precision Boost і Extended Frequency Range (XFR)**: Ці технології дозволяють автоматично підвищувати тактову частоту в залежності від навантаження, забезпечуючи кращу продуктивність у режимах з високим навантаженням.
3. **Вбудовані графічні рішення**: Багато моделей Ryzen мають інтегровану графіку, що дозволяє виконувати базові графічні завдання без дискретної графічної карти.

Призначення:

Серія Ryzen є оптимальним вибором для:

- **Геймерів**: Завдяки своїй потужності та можливостям обробки графіки.
- **Професійних користувачів**: Для роботи з програмним забезпеченням, яке вимагає високих обчислювальних потужностей, наприклад, при відеомонтажі чи 3D-моделюванні.

- **Загального використання**: Ryzen підходять для офісної роботи, мультимедійних завдань і розваг [6,7].

Серія Ryzen Threadripper: екстремальна продуктивність

Серія Ryzen Threadripper є флагманською лінійкою процесорів AMD, розробленою для професіоналів, які потребують надзвичайної продуктивності. Threadripper забезпечує екстремальні можливості обробки, ідеально підходячи для ресурсоємних завдань.

Основні архітектурні особливості:

1. **Висока кількість ядер**: Threadripper може мати до 64 ядер і 128 потоків, що робить їх ідеальними для багатозадачності та паралельних обчислень.
2. **Широка шина даних**: Процесори Threadripper підтримують величезні обсяги оперативної пам’яті та велику пропускну спроможність, що критично важливо для обробки великих наборів даних.
3. **Технологія TR4**: Даний сокет забезпечує відмінну підтримку та інтеграцію з материнськими платами, що дозволяє користувачам досягати максимальної продуктивності своїх систем [8].

Перспективи:

- **Попит на екстремальну продуктивність**: Зростаючий інтерес до контенту 4K, віртуальної реальності та складних наукових обчислень підвищує попит на такі потужні рішення.
- **Розвиток технологій**: AMD постійно вдосконалює свої технології, інтегруючи нові функції, такі як підтримка PCIe 4.0, що підвищує швидкість обробки даних і пропускну спроможність [8].

1.4. Порівняльний аналіз продуктивності процесорів Intel і AMD

У сучасному світі, де обчислювальні потужності відіграють ключову роль в успішності технологічних рішень, порівняльний аналіз продуктивності процесорів Intel і AMD є надзвичайно актуальним. Обидві компанії пропонують широкий спектр процесорів для різних застосувань, від простих домашніх комп'ютерів до потужних серверних рішень. У цьому підрозділі ми розглянемо різні аспекти продуктивності процесорів, зокрема їхню архітектуру, продуктивність у синтетичних тестах, реальних задачах та енергетичну ефективність.

Архітектурні відмінності

Архітектурні особливості процесорів Intel і AMD впливають на їхню продуктивність у різних сценаріях використання.

1. **Архітектура Intel**:
   - **Сокети**: Intel використовує різні сокети для своїх процесорів (наприклад, LGA 1151, LGA 2066), що визначає їхню сумісність з материнськими платами.
   - **Технологія Hyper-Threading**: Дозволяє кожному ядру виконувати два потоки одночасно, що підвищує продуктивність у багатопотокових завданнях.
   - **Продуктивність на ядро**: Intel традиційно має вищу продуктивність на ядро, що робить їхні процесори ідеальними для задач, які не використовують багато потоків [9].
2. **Архітектура AMD**:
   - **Сокети**: AMD використовує сокети AM4 і TR4, що дозволяє значну гнучкість у виборі материнських плат.
   - **Технологія Simultaneous Multithreading (SMT)**: Аналогічна Intel Hyper-Threading, ця технологія дозволяє поліпшити продуктивність у багатопотокових завданнях.

- **Модульна архітектура**: Ryzen має модульну архітектуру, що дозволяє розміщувати декілька ядер на одному чипі, що покращує ефективність у багатопотокових середовищах [9].

Результати синтетичних тестів

Синтетичні тести дозволяють отримати об’єктивну інформацію про продуктивність процесорів у різних сценаріях.

1. **Процесори Intel**:
   - У тестах, таких як Cinebench R20 і Geekbench, процесори Intel зазвичай демонструють високі результати завдяки своїй архітектурі та високій частоті такту.
   - Моделі з серії Core i9 показують вражаючі результати у одноядерних завданнях, що робить їх кращим вибором для ігор та інших однопоточних програм.
2. **Процесори AMD**:
   - В останніх поколіннях Ryzen, зокрема Ryzen 5000 серії, AMD змогла наздогнати Intel за продуктивністю в синтетичних тестах.
   - У багатопотокових тестах, таких як Blender та Handbrake, процесори AMD часто перевершують Intel завдяки великій кількості ядер і потоків [10].

Продуктивність у реальних задачах

Хоча синтетичні тести дають змогу оцінити теоретичну продуктивність, реальні завдання є важливим критерієм для оцінки ефективності процесорів.

1. **Геймери**:
   - У іграх, де важлива продуктивність на ядро, процесори Intel, зокрема серії i7 і i9, часто забезпечують вищу частоту кадрів завдяки їхній архітектурі.

- Проте Ryzen 5 і 7 також демонструють відмінну продуктивність у нових іграх, де оптимізація під багатопоточність стає все більш важливою.

2. **Професійні користувачі**:
   - Для професіоналів, які працюють у таких сферах, як відеомонтаж чи 3D-моделювання, процесори AMD, з великою кількістю ядер, можуть запропонувати кращу продуктивність завдяки паралельній обробці.
   - Intel, у свою чергу, може залишатися кращим вибором для спеціалізованих завдань, які вимагають високої продуктивності на ядро [10,11].

Ефективність використання електроенергії

Енергетична ефективність процесорів стає все більш важливою у світі, де зростають вимоги до продуктивності при зменшенні споживання енергії.

1. **Intel**:
   - Процесори Intel зазвичай показують хороші результати в енергетичній ефективності, особливо у нових поколіннях, завдяки технології 10 нм.
   - Водночас, серії Core i9 мають досить високе споживання енергії, що може вплинути на загальну ефективність системи.
2. **AMD**:
   - Процесори AMD, особливо серії Ryzen, показують вражаючу продуктивність при низькому споживанні енергії завдяки 7 нм технології виготовлення.
   - Це робить AMD більш привабливими для систем, де важлива енергетична ефективність, зокрема для ноутбуків та серверів [10,11].

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі магістерської роботи було здійснено детальний огляд процесорів від Intel і AMD, а також порівняно різні характеристики, такі як: продуктивність, енергетична ефективність а також відмінності у будові. Також розглянули характеристики окремих серій процесорів, і їх реальних показників продуктивності в різних сценаріях синтетичних тестів. Це дало змогу з'ясувати переваги та недоліки кожного з виробників.

По-перше, серія **Intel**, зокрема процесори Core та Xeon, продемонструвала високу продуктивність на ядро, що робить їх ідеальними для завдань, які не використовують багато потоків. Технології, такі як Hyper-Threading, дозволяють інтенсифікувати роботу в багатопотокових середовищах. Особливо варто відзначити успіх Intel у виробництві процесорів для ігор, де висока частота кадрів є критично важливою.

По-друге, **AMD** в останніх поколіннях своїх процесорів, зокрема Ryzen і EPYC, продемонструвала значний прогрес у зростанні продуктивності, завдяки великій кількості ядер та підтримці нових технологій, таких як SMT. Це дозволяє AMD бути конкурентоспроможною в сегменті серверних рішень та серед професіоналів, які працюють із ресурсоємними програмами. Рішення AMD часто виграють у багатопотокових завданнях, завдяки модульній архітектурі та оптимізації для високої пропускної здатності.

Крім того, у порівняльному аналізі було відзначено, що обидві компанії продовжують вдосконалювати свої технології виготовлення. Процесори AMD, виготовлені за 7 нм технологією, демонструють високу енергетичну ефективність, що стає важливим чинником у сучасних умовах, коли питання екологічності та енергозбереження стають актуальними. Intel, у свою чергу, також працює над зниженням енергоспоживання своїх нових моделей, що свідчить про зростаючу важливість цього аспекту в розробці процесорів.

У підсумку, вибір між процесорами Intel та AMD залежить від конкретних потреб користувачів. Intel може бути кращим вибором для тих, хто

потребує максимальної продуктивності на ядро, тоді як AMD є відмінним варіантом для користувачів, які працюють у середовищах, що вимагають високої паралельності та багатозадачності. З огляду на постійні інновації обох компаній, ринок процесорів залишається динамічним і цікавим, що дає можливість користувачам обирати оптимальні рішення відповідно до своїх потреб.

Цей розділ демонструє, як розвиток технологій впливає на продуктивність обчислювальних систем та їх застосування у різних сферах, що є важливим для розуміння актуальних тенденцій у розробці високопродуктивних програмно-апаратних комплексів, які будуть розглянуті в третьому розділі роботи.

РОЗДІЛ 2
АНАЛІЗ ПРОГРАМНИХ РІШЕНЬ ДЛЯ ПОБУДОВИ ВИСОКОПРОДУКТИВНИХ ПАК

2.1. Сучасні тенденції розробки багатопотокових додатків

Сучасний світ програмування активно рухається у напрямку багатопотокових та паралельних рішень. Зростаюча популярність багатоядерних процесорів, розвиток хмарних обчислень і необхідність обробки великих обсягів даних створюють нові виклики та можливості для розробників. Цей підрозділ аналізує сучасні тенденції в області розробки багатопотокових додатків, розглядаючи вплив апаратного прогресу, появу нових програмних парадигм та проблеми, з якими стикаються розробники.

Прогрес апаратного забезпечення як драйвер багатопотоковості

Однією з ключових тенденцій у розробці багатопотокових додатків є оптимізація програм під багатоядерні та багатопроцесорні системи. Виробники процесорів (Intel, AMD) припинили гонитву за підвищенням тактових частот і натомість зосередилися на збільшенні кількості ядер. У сучасних процесорах можна знайти від кількох до кількох десятків ядер, кожне з яких здатне виконувати паралельні завдання [12].

Розподіл навантаження між потоками та ядрами дозволяє уникнути простоїв і забезпечує максимально ефективне використання обчислювальних ресурсів. Крім того, розвиток графічних процесорів (GPU) також стимулює використання паралельних обчислень, оскільки вони здатні виконувати сотні й тисячі потоків одночасно, що особливо корисно для машинного навчання та обробки великих масивів даних [12].

Популяризація хмарних і розподілених обчислень

Ще однією важливою тенденцією є зростання популярності хмарних обчислень, що надають розробникам можливість масштабувати додатки та обробляти дані в паралельному режимі на десятках або навіть сотнях серверів одночасно. Такі платформи, як Amazon Web Services (AWS), Microsoft Azure та Google Cloud, надають інструменти для побудови розподілених систем, де обчислення виконуються паралельно.

Парадигма **"serverless computing"** дозволяє запускати функції та мікросервіси незалежно від конкретної інфраструктури, спрощуючи розробку паралельних рішень. Це звільняє розробників від необхідності вручну керувати потоками та синхронізацією, зосередивши увагу на логіці додатка [13].

Перехід до асинхронного програмування

Асинхронне програмування стало новим стандартом у розробці додатків, де необхідно забезпечити швидке реагування на події та зменшити час очікування. Традиційний підхід, у якому програма чекає завершення операції вводу-виводу (наприклад, запиту до бази даних), поступається місцем асинхронним моделям, де виконання завдання не блокує основний потік [12].

Асинхронне програмування стало особливо популярним у веб-додатках та мобільних системах, де важливо забезпечити плавний користувацький досвід. Наприклад, у JavaScript використання промісів (Promises) і ключових слів async/await дозволяє ефективно керувати асинхронними викликами. Аналогічний підхід використовується в мовах Python та C#, що робить їх популярними для серверних додатків.

Проблеми та виклики багатопотокового програмування

Хоча багатопотокові рішення відкривають нові можливості для підвищення продуктивності, розробники стикаються з низкою проблем [12,13]:

1. **Умови гонки (race condition)**: виникають, коли кілька потоків одночасно звертаються до спільного ресурсу без належної синхронізації.
2. **Блокування (deadlock)**: ситуація, коли два або більше потоки блокують один одного, очікуючи на звільнення ресурсів.
3. **Збільшення складності коду**: додавання паралельних потоків ускладнює програму, що підвищує ризик появи помилок і ускладнює процес тестування.
4. **Накладні витрати**: створення і перемикання потоків потребує ресурсів, що може знижувати продуктивність на деяких системах.

Інструменти для багатопотоковості

Сучасні розробники мають у своєму розпорядженні широкий вибір інструментів для створення багатопотокових рішень. Це включає як високорівневі фреймворки, так і низькорівневі бібліотеки для точного контролю за роботою потоків [13].

1. **OpenMP**: фреймворк для паралельного програмування на C/C++, що дозволяє розподіляти обчислення між потоками за допомогою директив компілятора.
2. **PThreads**: низькорівнева бібліотека для роботи з потоками в POSIX-сумісних системах.
3. **Task Parallel Library (TPL)** у C# і .NET: бібліотека для спрощення роботи з паралельними завданнями.
4. **ExecutorService** у Java: інструмент для управління пулами потоків і асинхронними завданнями.

2.2. Огляд мов програмування для роботи з потоками

У цьому підрозділі детально розглянемо мови та інструменти програмування, які підтримують багатопотокове програмування та асинхронні обчислення. Аналіз зосереджується на таких мовах і бібліотеках, як C#, Java, PThreads, та фреймворку OpenMP. Кожна з них має свої переваги, недоліки та специфіку використання. Їхнє розуміння є важливим для вибору відповідних інструментів для розробки високопродуктивних програмно-апаратних комплексів.

Мова програмування C#

C# є однією з найбільш популярних мов програмування для розробки багатопотокових і асинхронних додатків, особливо у межах платформи .NET. Вона надає розробникам багатий набір інструментів для роботи з потоками, завдяки чому забезпечується ефективне використання ресурсів комп'ютера [14].

Інструменти багатопотоковості в C#:

- **Thread** – базовий клас, що дозволяє створювати й керувати потоками. Він дає змогу виконувати окремі завдання паралельно з основним потоком. Однак пряме використання потоків ускладнює управління ресурсами.
- **Task** – більш високорівнева абстракція над потоками. Об’єкти `Task` дозволяють виконувати завдання асинхронно, забезпечуючи простіший механізм управління.
- **Async/Await** – конструкції для асинхронного виконання операцій. Вони знижують складність коду, роблячи його зрозумілішим та менш схильним до помилок.
- **TPL (Task Parallel Library)** – бібліотека, яка автоматизує розподіл завдань між потоками та ядрами процесора. Вона дозволяє

виконувати завдання паралельно з мінімальними зусиллями з боку розробника.

Приклад використання TPL:

```
Task.Run(() =>
{
    // Виконання довготривалого завдання
    Console.WriteLine("Завдання виконано.");
});
```

Рис. 2.1. Приклад використання TPL

C# є чудовим вибором для створення корпоративних додатків, веб-сервісів та ігрових програм. Його екосистема підтримує ефективне управління потоками, що особливо важливо для розробки високопродуктивних рішень [14].

Мова програмування Java

Java – ще одна популярна мова програмування, що забезпечує підтримку багатопотоковості на рівні віртуальної машини Java (JVM). Вона використовується для розробки великих серверних систем, де продуктивність і стабільність мають вирішальне значення [15].

Основні інструменти для роботи з потоками в Java:

- **Thread** – клас для створення потоків, що дає змогу виконувати завдання паралельно. Однак управління потоками може бути складним при збільшенні кількості завдань.
- **ExecutorService** – API для управління пулами потоків. Воно дозволяє розподіляти завдання між кількома потоками, що спрощує масштабування додатка.

- **Future та CompletableFuture** – інструменти для виконання асинхронних завдань. Вони дозволяють відстежувати стан виконання завдань і обробляти результати після завершення.
- **Synchronized-блоки та Locks** – механізми для управління доступом до спільних ресурсів, які запобігають виникненню умов гонки.

Приклад створення потоку в Java:

```
Thread thread = new Thread(() -> {
    System.out.println("Потік виконано.");
});
thread.start();
```

Рис. 2.2. Приклад створення потоку в Java

Java активно використовується у фінансовій сфері, веб-розробці та для створення масштабованих серверних додатків. Її підтримка багатопотоковості та стабільна робота з великими обсягами даних роблять її важливим інструментом для високопродуктивних рішень.[15]

PThreads

PThreads (POSIX Threads) – це низькорівнева бібліотека для роботи з потоками в UNIX-подібних операційних системах, таких як Linux. Вона надає програмістам високий рівень контролю за роботою потоків і забезпечує максимально ефективне використання ресурсів [16].

Особливості використання PThreads:

- **Точний контроль над потоками**: програмісти можуть керувати створенням, зупинкою та завершенням роботи потоків на низькому рівні.
- **Синхронізація через м'ютекси та умови (condition variables)**: дозволяє уникнути конфліктів між потоками при доступі до спільних ресурсів.

- **Висока продуктивність**: завдяки мінімальним накладним витратам на управління потоками.

Приклад створення потоку за допомогою PThreads:

```c
#include <pthread.h>
#include <stdio.h>

void* threadFunction(void* arg) {
    printf("Потік виконано.\n");
    return NULL;
}

int main() {
    pthread_t thread;
    pthread_create(&thread, NULL, threadFunction, NULL);
    pthread_join(thread, NULL);
    return 0;
}
```

Рис 2.3. Приклад створення потоку за допомогою PThreads

PThreads забезпечує максимальну продуктивність, але його використання пов'язане зі значною складністю, оскільки програмісти мають керувати всіма аспектами роботи потоків самостійно. Цей підхід підходить для системного програмування, де важлива точність і надійність [16].

OpenMP

OpenMP – це стандарт для паралельного програмування на C, C++ та Fortran, який використовується для розподілу обчислень між кількома потоками. Він базується на директивах компілятора, що робить його простим у використанні для наукових та інженерних обчислень [15,16].

Особливості OpenMP:

- **Простота використання**: паралельні секції коду визначаються директивами #pragma, що не потребує значних змін у коді.

- **Автоматичний розподіл завдань**: OpenMP автоматично розподіляє обчислення між потоками.
- **Гнучкість**: розробники можуть точно контролювати кількість потоків і розподіл навантаження.

Приклад використання OpenMP:

```
#include <omp.h>
#include <stdio.h>

int main() {
    #pragma omp parallel
    {
        printf("Потік: %d\n", omp_get_thread_num());
    }
    return 0;
}
```

Рис. 2.4. Приклад використання OpenMP

OpenMP широко використовується для задач, пов'язаних із моделюванням і обробкою великих обсягів даних. Однак його застосування обмежується лише такими мовами, як C/C++ та Fortran, що звужує коло його користувачів [15,16].

Паралельне виконання циклів

У OpenMP є можливість паралелізувати цикли через директиву **for**, яка дозволяє розподіляти ітерації циклу між різними потоками. Вона має наступний синтаксис:

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    // код циклу
}
```

Рис. 2.5. Синтаксис директиви for

Директиву **#pragma omp parallel for** використовують перед циклом for, для того щоб вказати компілятору, що ітерації циклу повинні виконуватися паралельно. Розподілення ітерацій між потоками відбувається автоматично [17].

За допомогою директиви **num_threads** можна визначити конкретну кількість потоків, що будуть використовуватися для виконання циклу:

```
#pragma omp parallel for num_threads(4)
for (int i = 0; i < n; i++) {
    // код циклу
}
```

Рис. 2.6 Використання директиви num_threads

OpenMP надає кілька варіантів розподілу ітерацій:

- Статичний розподіл: кожен потік отримує однакову кількість ітерацій. Використовується за замовчуванням.
- Динамічний розподіл ітерацій: ітерації діляться на блоки, і потоки отримують нові блоки по мірі завершення своїх поточних завдань.
- Розподіл із непостійними розмірами блоків: потоки спочатку отримують більші блоки, а потім вже менші.

У разі необхідності синхронізаціх можна використати наступні директиви OpenMP [17]:

1. **#pragma omp barrier:** синхронізує потоки, примушуючи їх чекати один одного в точці синхронізації.
2. **#pragma omp critical:** гарантує, що тільки один потік одночасно виконуватиме блок коду.

2.3. Порівняння підходів до організації взаємодії потоків

Багатопотокове програмування передбачає взаємодію кількох потоків, які можуть одночасно виконувати завдання. При цьому важливим є правильне керування доступом до спільних ресурсів, щоб уникнути помилок, таких як **умови гонки (race condition)** або **блокування (deadlock)**. Для цього використовуються різні інструменти та механізми синхронізації, які забезпечують узгоджену роботу потоків. У цьому підрозділі розглянемо найпоширеніші підходи: **семафори**, **м'ютекси та події**, **монітори**, **бар'єри** та **критичні секції**. Кожен із цих механізмів має свої особливості, переваги й недоліки [16].

Семафори

Семафор – це примітив синхронізації, що використовується для обмеження доступу до певного ресурсу. Він працює як лічильник, який контролює кількість одночасних доступів до ресурсу. Семафор можна уявити як чергу, яка дозволяє певній кількості потоків одночасно користуватися ресурсом [16].

Види семафорів:

1. **Бінарний семафор**: функціонує як м'ютекс, допускаючи доступ тільки одному потоку за раз.
2. **Лічильниковий семафор**: встановлює обмеження на кількість потоків, що можуть одночасно використовувати ресурс (наприклад, 3 потоки).

Приклад використання семафора в C#:

```
Semaphore semaphore = new Semaphore(3, 3);  // Дозволяє доступ 3 потокам

void UseResource() {
    semaphore.WaitOne();  // Чекає, якщо всі місця зайняті
    Console.WriteLine("Ресурс використовується.");
    semaphore.Release();  // Звільняє місце
}
```

Рис. 2.7. Приклад використання семафора в C#

Переваги та недоліки:

Переваги: дозволяє обмежувати кількість одночасних доступів до ресурсу.

Недоліки: може ускладнювати код і призводити до **взаємного блокування (deadlock)**, якщо не правильно керувати його використанням [18].

М'ютекси та події

М'ютекс (mutual exclusion) – це об’єкт синхронізації, що дозволяє уникнути одночасного доступу кількох потоків до одного ресурсу. На відміну від семафорів, м'ютекси використовуються для захисту критичних секцій коду. Він надає доступ лише одному потоку та блокує інші до моменту звільнення ресурсу [18].

Приклад використання м'ютекса в C#:

```
Mutex mutex = new Mutex();

void CriticalSection() {
    mutex.WaitOne();  // Блокує ресурс для інших потоків
    Console.WriteLine("Критична секція.");
    mutex.ReleaseMutex();  // Звільняє ресурс
}
```

Рис. 2.8. Приклад використання м'ютекса в C#

Події (events) використовуються для сигналізації між потоками, що певна умова виконана, й інші потоки можуть продовжити роботу. Це зручно для координації потоків, які виконують завдання у певній послідовності.

Приклад події в C#:

```
AutoResetEvent autoEvent = new AutoResetEvent(false);

void FirstTask() {
    Console.WriteLine("Перше завдання завершено.");
    autoEvent.Set();  // Сигналізує іншому потоку
}

void SecondTask() {
    autoEvent.WaitOne();  // Чекає сигналу
    Console.WriteLine("Друге завдання починається.");
}
```

Рис. 2.9. Приклад події в C#

Переваги та недоліки:

Переваги: м'ютекси забезпечують надійну синхронізацію критичних секцій. Події дозволяють організувати координацію між потоками.

Недоліки: м'ютекси можуть призвести до взаємного блокування, а робота з подіями ускладнює код [18].

Монітори

Монітор – це високорівневий механізм синхронізації, який автоматично забезпечує доступ до критичних секцій та управляє блокуванням і звільненням ресурсів. Він поєднує функціональність м'ютексів та умовних змінних, дозволяючи зручно управляти доступом до об'єктів у багатопотоковому середовищі.

У C# монітори реалізовані через ключове слово **lock**, яке забезпечує безпечний доступ до об’єкта.

Приклад використання монітора в C#:

```
object lockObject = new object();

void UseResource() {
    lock (lockObject) {
        Console.WriteLine("Монітор заблокував ресурс.");
    }
}
```

Рис. 2.10. Приклад використання монітора в C#

Переваги та недоліки:

Переваги: монітори зручні для використання завдяки простому синтаксису.

Недоліки: монітор може спричиняти **блокування** потоків, якщо ресурс не звільняється вчасно [19].

Бар’єри

Бар'єри – це механізм синхронізації, який використовується для координації роботи кількох потоків. Він зупиняє потоки на певній точці виконання доти, поки всі потоки не досягнуть цієї точки. Після цього всі потоки продовжують виконання одночасно [19].

Приклад використання бар’єру в C#:

```
Barrier barrier = new Barrier(3);

void TaskWithBarrier() {
    Console.WriteLine("Завдання до бар'єру.");
    barrier.SignalAndWait();  // Чекає інших потоків
    Console.WriteLine("Завдання після бар'єру.");
}
```

Рис. 2.11. Приклад використання бар’єру в C#

Переваги та недоліки:

Переваги: забезпечує синхронне виконання кількох потоків на певному етапі.

Недоліки: може спричиняти **простої** при асиметричному навантаженні потоків.

Критичні секції

Критична секція – це частина коду, яка може виконуватися лише одним потоком у будь-який момент часу. Використання критичних секцій дозволяє уникнути конфліктів при доступі до спільних ресурсів.

У C# для позначення критичних секцій зазвичай використовують `lock`, як у прикладі з моніторами. У інших мовах доступні аналогічні механізми, наприклад, у C/C++ використовується EnterCriticalSection [19,20].

Приклад:

```
object criticalLock = new object();

void CriticalSection() {
    lock (criticalLock) {
        Console.WriteLine("Виконання критичної секції.");
    }
}
```

Рис. 2.12 Приклад використання критичних секцій в C#

Переваги та недоліки:

Переваги: забезпечує безпеку доступу до ресурсів.

Недоліки: може знижувати продуктивність, якщо використовується занадто часто [20].

2.4. Аналіз продуктивності програмних засобів у багатопотоковому середовищі

Багатопотокове програмування є потужним інструментом для підвищення продуктивності програм, проте для досягнення бажаного результату необхідно проводити детальний аналіз продуктивності застосовуваних програмних засобів у багатопоточному середовищі. У цьому підрозділі розглянемо ключові фактори, які впливають на продуктивність, методи вимірювання та порівняння продуктивності, а також практичні приклади оптимізації [18,19].

Ключові фактори продуктивності

Визначення продуктивності багатопотокових додатків залежить від ряду факторів, які можуть значно вплинути на їх ефективність [19]:

1. **Кількість потоків**: Занадто велика кількість потоків може призвести до перевантаження системи та зниження продуктивності через контекстні перемикання. Оптимальна кількість потоків залежить від кількості доступних процесорів і специфіки задачі.
2. **Синхронізація**: Часті блокування і розблокування потоків, а також використання механізмів синхронізації можуть викликати затримки та знижувати загальну продуктивність. Мінімізація критичних секцій і правильний вибір механізмів синхронізації є важливими аспектами.
3. **Конфлікти доступу до ресурсів**: Коли кілька потоків намагаються одночасно отримати доступ до одного й того ж ресурсу, може виникнути конфлікт доступу до ресурсів, що суттєво знижує продуктивність програми.
4. **Системні ресурси**: Обсяг оперативної пам’яті, швидкість процесора, і конфігурація апаратного забезпечення відіграють важливу роль у визначенні продуктивності багатопотокових програм. Наприклад,

наявність багато-ядерних процесорів дозволяє більш ефективно виконувати паралельні обчислення.

Методи вимірювання продуктивності

Для оцінки продуктивності багатопотокових програм використовуються різноманітні методи і підходи [19]:

1. **Час виконання**: Найпоширеніший метод, який передбачає вимірювання загального часу, необхідного для виконання програми. Цей метод дозволяє оцінити швидкість виконання, але може не відображати повну картину через варіативність умов виконання.
2. **Використання процесора**: Вимірювання відсотка використання процесора під час виконання програми дозволяє оцінити, наскільки ефективно використовуються ресурси. Високе використання процесора може вказувати на ефективне паралельне виконання.
3. **Продуктивність**: Вимірювання продуктивності, що включає обробку певної кількості задач за одиницю часу. Це важливо для визначення, наскільки програма справляється з навантаженням.
4. **Профілювання**: Використання інструментів профілювання дозволяє детально вивчити роботу програми, виявити «вузькі місця» і оптимізувати код. Такі інструменти, як Visual Studio Profiler або Intel VTune, надають цінну інформацію про використання ресурсів.

Порівняння продуктивності

Для ефективного аналізу продуктивності різних програмних рішень в багатопоточному середовищі важливо проводити порівняння. Основні етапи включають [18,19]:

1. **Вибір тестових сценаріїв**: Необхідно визначити конкретні сценарії, які будуть використовуватись для тестування. Це можуть бути обчислювальні задачі, що потребують різних типів ресурсів.

2. **Запуск тестів**: Виконання тестів для кожного програмного рішення в однакових умовах, що дозволяє отримати коректні дані для порівняння.
3. **Аналіз результатів**: Порівняння часу виконання, використання ресурсів та інших показників для кожного з тестованих рішень. Важливо також враховувати, як різні методи синхронізації впливають на продуктивність.

Практичні приклади оптимізації

Оптимізація багатопотокових програм може включати різні стратегії, спрямовані на підвищення продуктивності [18]:

1. **Зменшення частоти блокувань**: Застосування оптимізованих алгоритмів, що мінімізують потребу у синхронізації. Наприклад, використання lock-free структур даних може зменшити час очікування для потоків.
2. **Паралельне виконання**: Розподіл завдань між потоками таким чином, щоб зменшити «вузькі місця». Наприклад, у задачах з обробки великих обсягів даних можна використовувати паралельні алгоритми, які розділяють навантаження між потоками.
3. **Кешування**: Використання локальних кешів для зменшення конфліктів доступу до загальних ресурсів. Кешування даних, які часто використовуються, може значно знизити затримки, пов’язані з доступом до пам’яті.
4. **Вибір оптимальних структур даних**: Використання структур даних, які краще підходять для багатопотокового доступу. Наприклад, використання колекцій, які підтримують паралельні операції, таких як ConcurrentBag у C#.

2.5 Вибір мови програмування

Провівши дослідження різних мов програмування та способів у них створювати паралельні програми, було прийнято рішення для виконання основного завдання обрати мову програмування C та набір директив OpenMP.

Використання C та OpenMP у поєднанні дозволяє досить ефективно використовувати багатоядерні процесори.

Мова програмування C

Це одна з найбільш використовуваних мов програмування. Вона дозволяє програмістам писати код, який працює безпосередньо із апаратним забезпеченням. Також низький рівень абстракції дозволяє отримати високий контроль над виконанням. C та OpenMP мають підтримку на різних платформах, що робить їх використання досить універсальним.

OpenMP

Для того, щоб паралелізувати частини програми OpenMP використовує директиви процесора (для прикладу #pragma omp parallel). Це дозволяє із мінімальними змінами в коді точно визначити блоки коду, які мають виконуватися паралельно. Завдяки простій можливості паралелізувати код, це можуть зробити навіть програмісти, які не дуже добре розбираються в паралельному програмуванні.

Порівняно із іншими бібліотеками для паралельного програмування, OpenMP вимагає менше зусиль на налаштування і разом з цим має простіший синтаксис для розбиття стандартних завдань на частини, що виконуватимуться паралельно.

OpenMP дозволяє легко керувати кількістю використовуваних потоків, а також автоматично може використовувати всі ядра комп’ютера, що позитивно впливає на ефективність роботи програми. Можливість

використовувати різну кількість потоків додає більше гнучкості у налаштуванні продуктивності. Також OpenMP автоматично керує синхронізацією потоків, що зменшує кількість помилок у багатопотокових програмах.

Ще одним не менш важливим фактором є можливість легко реалізувати паралелізм на рівні циклів, паралелізм на рівні функцій, а також динамічний паралелізм, що додає ще більше універсальності для використання для різних типів задач.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі магістерської роботи було проведено детальний аналіз сучасних тенденцій, технологій і механізмів, які використовуються в багатопоточному програмуванні. Розглянуті аспекти підкреслюють важливість правильного вибору методів синхронізації та структур даних для забезпечення ефективної взаємодії між потоками.

Сучасні тенденції розробки багатопотокових додатків: Багатопоточність стає все більш актуальною у зв’язку з розвитком багато-ядерних процесорів і зростанням вимог до продуктивності програм. Використання багатопотокового підходу дозволяє суттєво скоротити час виконання обчислень та обробки даних, що є важливим аспектом для розробки високопродуктивних програм.

Огляд мов програмування для роботи з потоками: Аналіз мов програмування, таких як C, C# і Java, а також використання бібліотек PThreads і OpenMP, продемонстрував різноманітність підходів до організації потокового виконання. Кожна з мов має свої специфічні механізми для управління потоками, що дозволяє вибрати оптимальний інструмент в залежності від специфіки завдання.

Порівняння підходів до організації взаємодії потоків: Розглянуті механізми синхронізації, такі як семафори, м'ютекси, монітори, бар'єри та критичні секції, продемонстрували різні способи управління доступом до ресурсів. Вибір відповідного механізму залежить від характеру задачі, а також від вимог до продуктивності та безпеки виконання.

Аналіз продуктивності програмних засобів у багатопоточному середовищі: Розгляд ключових факторів продуктивності, методів вимірювання, а також стратегій оптимізації дозволяє визначити важливі аспекти, що впливають на ефективність багатопоточних програм. Встановлення оптимального балансу між кількістю потоків, синхронізацією та

доступом до ресурсів є критично важливим для досягнення високої продуктивності.

У результаті проведеного аналізу можна стверджувати, що багатопотокове програмування є важливим напрямком у сучасній розробці програмного забезпечення. Правильний вибір інструментів, методів синхронізації та структур даних може суттєво вплинути на продуктивність та надійність програм. Зростаючі вимоги до обчислювальних систем вимагають постійного вдосконалення навичок розробників у цій галузі, що в свою чергу стимулює подальші дослідження та розробки у сфері багатопотокового програмування.

У результаті дослідження вдалося дійти до наступного висновку: поєднання мови програмування C та OpenMP є найкращим варіантом для вирішення поставлених задач. Це поєднання дозволяє легко та ефективно використовувати багатоядерні процесори для прискорення виконання програм. OpenMP дає можливість паралелізувати обчислення без значних змін у коді, що робить процес паралельного програмування доступним навіть для тих, хто тільки починає працювати з багатопоточними програмами.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ВИСОКОПРОДУКТИВНОГО ПАК ТА АНАЛІЗ ЙОГО ЕФЕКТИВНОСТІ

3.1 Розробка апаратної частини

Архітектурне проєктування високопродуктивного програмно-апаратного комплексу (ПАК) є одним із ключових етапів розробки складних обчислювальних систем, що потребують забезпечення максимальної ефективності обробки даних та паралельного виконання завдань. На цьому етапі визначаються основні компоненти системи, їхні взаємозв’язки, способи обробки потоків даних та підходи до організації взаємодії між апаратним і програмним забезпеченням. Цей підрозділ присвячено розробці структури ПАК, вибору компонентів, а також обґрунтуванню архітектурних рішень для досягнення оптимальної продуктивності.

Так як високопродуктивний ПАК має вирішувати складні завдання над векторами та матрицями, та різні варіанти поєднання дій над ними, потрібно обрати досить потужний процесор, для забезпечення ефективної роботи. Враховуючи основні типи виразів, під які буде створюватися комплекс, було прийнято рішення обрати процесор Intel Core i5-8300H(рис. 3.1), який містить 4 ядра, 8 логічних потоків, в ньому присутній графічний модуль, але для кращої роботи буде обраний ще один дискретний. На даний момент існують і більш потужні рішення, але потужності цього процесору буде достатньо для виконання поставленої задачі, що в свою чергу допоможе зекономити кошти на інші компоненти а також на витрати, які не входять до створення високопродуктивного ПАК. Конкретні характеристики даного процесора можна глянути у додатку №1. Також окрім характеристик є конкретні дані тестів для цього процесора. За потреби їх можна знайти у додатку №2 [21].

Рис 3.1 процесор Intel Core i5-8300H[19]

Враховуючи, що процесор буде виконувати складні математичні обрахунки, під час своєї роботи він буде виділяти велику кількість тепла і для того, щоб його ефективно охолоджувати, необхідно обрати відповідний кулер, який також буде мати певний запас потужності, який може знадобитися для охолодження потужнішої системи у майбутньому. Доволі не поганим варіантом буде кулер DeepCool Gammaxx 400EX. Він був обраний для дипломного проекту бакалавра і в даному випадку цілком і повністю задовольняє необхідні вимоги. Його потужності вистачає для того, щоб впоратися із TDP у 180 Вт, це дасть змогу забезпечити дуже ефективне охолодження процесора а також підтримання прийнятної температури всередині корпусу комплексу.

Наступним етапом необхідно обрати материнську плату. В умовах завдання сказано, що дані мають братися із 4-х вінчестерів, отже наша материнська плата повинна мати 4 роз'єми під них, а також мати хороше охолодження і бути сумісною із обраним процесором. Найкращим варіантом стала MSI MPG Z790 Carbon WiFi. Досить бюджетним варіантом буде додавання 2 модулі оперативної пам'яті Kingston Beast RGB DDR5 по 8Гб кожен. Цього буде достатньо для виконання поставленого завдання.

Тепер черга дійшла до запам'ятовуючого пристрою, а точніше 4-х пристроїв. Відносно дешевим і містким варіантом буде 4 накопичувачі Seagate Barracuda 500GB 3.5.

Наступним виберемо графічний процесор. У нашій системі основне навантаження буде не на ньому, тому можна обрати більш бюджетніші варіанти із доступних, які добре працюють із паралельними обчисленнями було обрано GeForce GTX 1660 Super.

Наступним кроком оберемо блок живлення. Він повинен бути достатньо потужним, щоб підтримувати всі компоненти. Хорошим варіантом буде блок живлення Seasonic A12-500 500W. Він в кілька разів потужніший ніж потрібно і це дасть змогу модифікувати систему без подальшої необхідності замінювати блок живлення більш потужним.

Останнім елементом у комплексі є корпус, у який все буде вміщено. Враховуючи набір компонентів, і необхідні потреби, було обрано корпус Deepcool Matrexx 50**.** Він підтримує ATX-плати, має місця для 4-6 вентиляторів а також підтримує відеокарти до 370мм.

Високопродуктивний програмно-апаратний комплекс складався під задачі лінійної алгебри, тому компоненти обиралися із підтриманням балансу ціни та необхідної потужності. Разом з цим, система є доволі гнучкою для модифікацій у майбутньому і деякі компоненти обиралися з огляду на майбутні апгрейди. Також її не дуже складно буде перебудувати, якщо зміниться основне завдання, для якого вона створювалася.

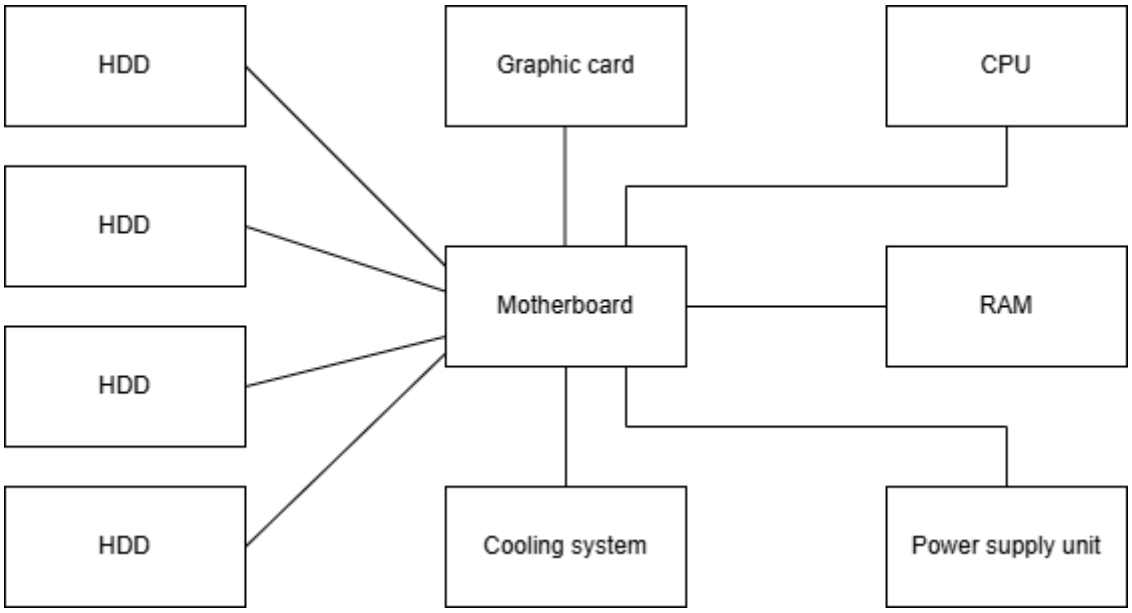


Рис. 3.2. Струкурна схема

Підібравши всі компоненти, можна порахувати вартість всієї системи. Конкретні дані занесені до таблиці у додатку №3

3.2 Розробка ПЗ високопродуктивного ПАК

На основі досліджень, які були проведені у розділі 2 дисертації, прийнято рішення обрати мову програмування C та OpenMP. Причиною цьому є можливість OpenMP автоматично розпаралелювати виконання обчислень на різній кількості потоків. Це у свою чергу дозволяє гнучко налаштувати необхідну кількість потоків під конкретні завдання. Також можливість автоматичного розпаралелювання може бути ефективнішою за роботу програміста, так як зменшує імовірність виникнення помилок.

Для прикладу, було обрано наступний математичний вираз, який буде обчислювати розроблений високопродуктивний програмно-апаратний комплекс:

$$MA = MB * MC + MD * ME$$

Паралельний математичний алгоритм

Математичний алгоритм буде мати наступний вигляд:

$$MA_H = MB * MC_H + MD * ME_H$$

Використані позначення:

- $H = \frac{N}{P}$
- N – розмірність матриць
- P – кількість ядер процесора(кількість потоків)
- MC_H – H рядків матриці MC;
- ME_H – H рядків матриці ME;

Алгоритм роботи потоків

Розроблений ПАК має 4-х ядерний процесор, що дозволить працювати із 4-ма потоками. Вираз складається із 4-х елементів, всі вони є матрицями. Значення кожної матриці знаходиться на окремому вінчестері, це дасть змогу одночасно зчитувати дані для 4-х матриць.

Зобразимо алгоритм роботи потоків у вигляді таблиці 3.1

Таблиця 3.1

Алгоритм роботи потоків

Дії потоків
T1
1. Ввести MB
2. Відправити сигнал про завершення введення MB до T2, T3, T4
3. Очікувати на завершення введення MC, MD, ME від T2, T3, T4 відповідно
4. Обчислення: $MA_H = MB * MC_H + MD * ME_H$
5. Відправити сигнал про завершення обчислення MA_H до T4

Продовження таблиці 3.1

T2
1. Ввести MC
2. Відправити сигнал про завершення введення MC до T1, T3, T4
3. Очікувати на завершення введення MB, MD, ME від T1, T3, T4 відповідно
4. Обчислення: $MA_H = MB * MC_H + MD * ME_H$
5. Відправити сигнал про завершення обчислення MA_H до T4
T3
1. Ввести MD
2. Відправити сигнал про завершення введення MD до T1, T2, T4
3. Очікувати на завершення введення MB, MC, ME від T1, T2, T4 відповідно
4. Обчислення: $MA_H = MB * MC_H + MD * ME_H$
5. Відправити сигнал про завершення обчислення MA_H до T4
T4
1. Ввести ME
2. Відправити сигнал про завершення введення ME до T1, T2, T3
3. Очікувати на завершення введення MB, MC, MD від T1, T2, T3 відповідно
4. Обчислення: $MA_H = MB * MC_H + MD * ME_H$
5. Очікувати на завершення обчислення MA_H від T1, T2, T3
6. Вивести MA.

Взаємодія потоків

Після написання алгоритму роботи потоків можна створити графічне представлення взаємодії потоків(рис. 3.3). У такому форматі набагато зручніше відслідковувати точки синхронізації.

Коли точки синхронізації не пов’язані між собою, виникають тупикові ситуації. Коли очікування на подію відбувається вічно, тому що сигнал про цю подію ніколи не наступить.

Для того, аби тупикових ситуацій не виникало, були використані бар’єри. Принцип їхньої роботи полягає в тому, що допоки всі потоки не завершать виконання своїх завдань, виконання наступних завдань буде заборонене. Як тільки всі потоки завершать свою роботу, виконання програми продовжиться.

На рисунку зображено 4 потоки T1- T4. Їх завдання полягає у зчитуванні значення матриць, а також обчислення частини виразу. Окрім цього T4 відповідає за виведення результату

Також під час моделювання було використано 2 бар’єри:

- B0 – для синхронізації по введенню. Допоки повністю не відбудеться зчитування даних, ніяких обчислень виконуватися не буде.
- B1 – для синхронізації по обчисленню значення MA. Допоки всі потоки не виконають обрахунки своєї частини виразу, загальний результат виводитися(записуватися не буде).

Схематично взаємодію потоків зображено на Рис. 3.3

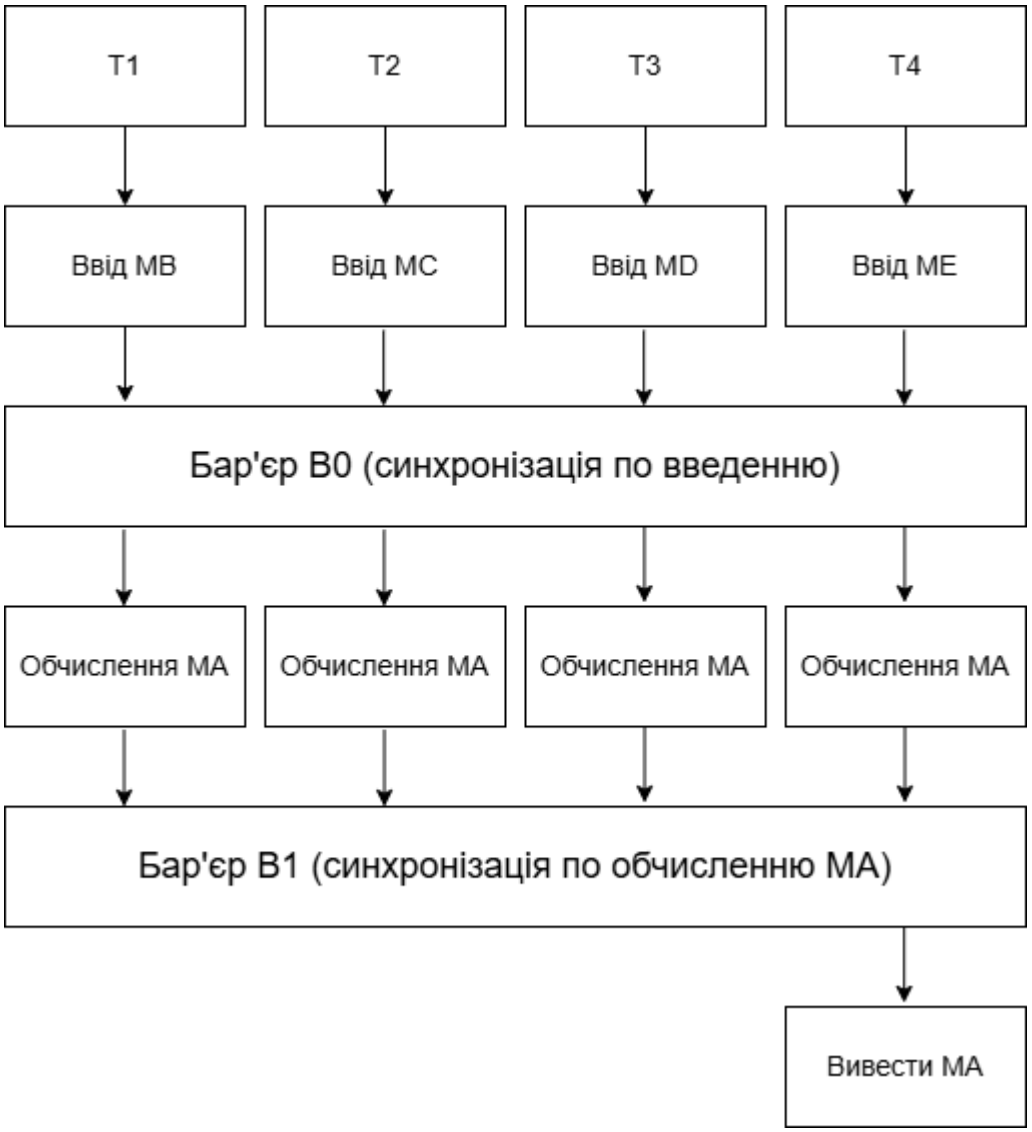


Рис 3.3 Взаємодія потоків

3.3 Розробка програм

Проаналізувавши поставлене завдання було прийнято рішення розробити програму на мові програмування C із використанням OpenMP. OpenMP дозволяє додатково використовувати дрібно-зернистий паралелізм (прагма for) та автоматично розпаралелити завдання, що у свою чергу є більш ефективним рішенням, ніж у бакалаврській роботі, так як нівелює людський фактор і зменшує ризик утворення помилок там, де алгоритм їх не допустить.

Використання pragma for в коді

Для тестування ефективності було створено 3 різні варіанти виконання завдання, а також четвертий – виконання на 1 потоці для порівняння результатів:

1. Використання 4 потоків і 1 pragma for

```
// Використання 1 pragma for
if (num_pragmas == 1) {
    #pragma omp parallel for
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}
```

Рис. 3.4. Використання 4 потоків і 1 pragma for

2. Використання 4 потоків і 2 pragma for

```
// Використання 2 pragma for
else if (num_pragmas == 2) {
    #pragma omp parallel for collapse(2)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}
```

Рис. 3.5. Використання 4 потоків і 2 pragma for

3. Використання 4 потоків і 3 pragma for:

```
// Використання 3 pragma for
else if (num_pragmas == 3) {
    #pragma omp parallel for collapse(3)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}
```

Рис.3.6. Використання 4 потоків і 3 pragma for:

4. Використання одного потоку:

```
// Звичайне однопоточне виконання
else {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}
```

Рис.3.7. Використання одного потоку

#pragma omp parallel for: дозволяє розподіляти ітерації циклу між потоками. **collapse** використовується для об'єднання кількох вкладених циклів у єдиний ітераційний простір, щоб OpenMP могла ефективно розподілити ітерації між потоками.

Огляд функцій програм:

create_matrix – функція, яка використовується для створення динамічних матриць.

freee_matrix – функція для звільнення пам’яті динамічної матриці.

fill_matrix – функція для заповнення матриці випадковими числами. Використовується для моделювання реальної ситуації. Так як на практиці дані матриці будуть завантажуватися з вінчестерів.

multiply_matrices – функція для множення матриць. Вміщує у собі всі варіанти виконання, включно і з варіантом виконання на 1 потоці.

add_matrices – функція для додавання матриць.

multiply_matrices_with_timing – функція для виконання множення матриць із підрахунком часу виконання для кожного варіанту, а також підрахонком суми, для перевірки правильності виконання всіма потоками.

add_matrices_with_timing – функція для додавання матриць із підрахунком часу виконання.

main – це вхідна точка програми. Тут відбувається ініціалізація масивів та кількості потоків, які будуть використовуватися для виконання завдання, утворення даних за допомогою функції **fill_matrix,** обрахунок різних варіантів виконання виразу за допомогою функції **multiply_matrices,** а також вивід проміжних та кінцевих результатів часу виконання програми для подальшого аналізу за допомогою функції **printf.**

3.4 Тестування високопродуктивного ПАК

Для тестування програми необхідно зафіксувати час виконання програми, щоб в подальшому можна було визначити коефіцієнти прискорення та ефективності. Так як дії будуть виконуватися над матрицями, по часу виконання різниця може бути доволі велика. Для розрахунку коефіцієнтів прискорення та ефективності використаємо наступні формули:

Для коефіцієнту прискорення:

$$Кп = T_1/T_P ;$$

Для коефіцієнту ефективності:

$$К_e = T_1/(T_P \cdot P) \cdot 100\% = Кп/P \cdot 100\%$$

Де P це кількість ядер, T_1 позначає час виконання на 1 потоці, T_P позначає час виконання на P потоках

Таблиця 3.2

Час виконання програми

	Час виконання на 4 ядрах			
N	P =1, c	P =4 використання, 1 pragma for, c	P =4 використання, 2 pragma for, c	P =4 використання, 3 pragma for, c
1000	13,534	3,954	3,863	4,008
2000	153,091	48,896	46,958	47,317
3000	597,574	184.077	181,872	187,306
4000	1 487,926	445,56	428,619	445,814

Як можна помітити, другий варіант: використання 4 потоків із паралельним виконанням циклів та використанням 2 pragma for дає найкращий час виконання. Варто зауважити, що цей варіант не у всіх випадках буде ефективнішим, інколи він не приноситиме ніякого результату, а в окремих випадках взагалі буде гіршим. Результативність даного підходу залежить від конкретно поставленої задачі і може мати різні результати в залежності від конфігурації системи і специфіки поставленого завдання

Вирахуємо коефіцієнти прискорення:

Таблиця 3.3

Коефіцієнти прискорення програми

	Кількість ядер			
N	P =1	P =4 використання, 1 pragma for, с	P =4 використання, 2 pragma for, с	P =4 використання, 3 pragma for, с
1000	1.0	3.42	3.50	3.38
2000	1.0	3.13	3.26	3.24
3000	1.0	3.25	3.29	3.19
4000	1.0	3.34	3.47	3.34

Розглянемо коефіцієнти ефективності

Таблиця 3.4

Коефіцієнти ефективності програми

	Кількість ядер			
N	P =1	P =4 використання, 1 pragma for, с	P =4 використання, 2 pragma for, с	P =4 використання, 3 pragma for, с
1000	100%	86%	88%	84%
2000	100%	78%	82%	81%
3000	100%	81%	82%	80%
4000	100%	83%	87%	83%

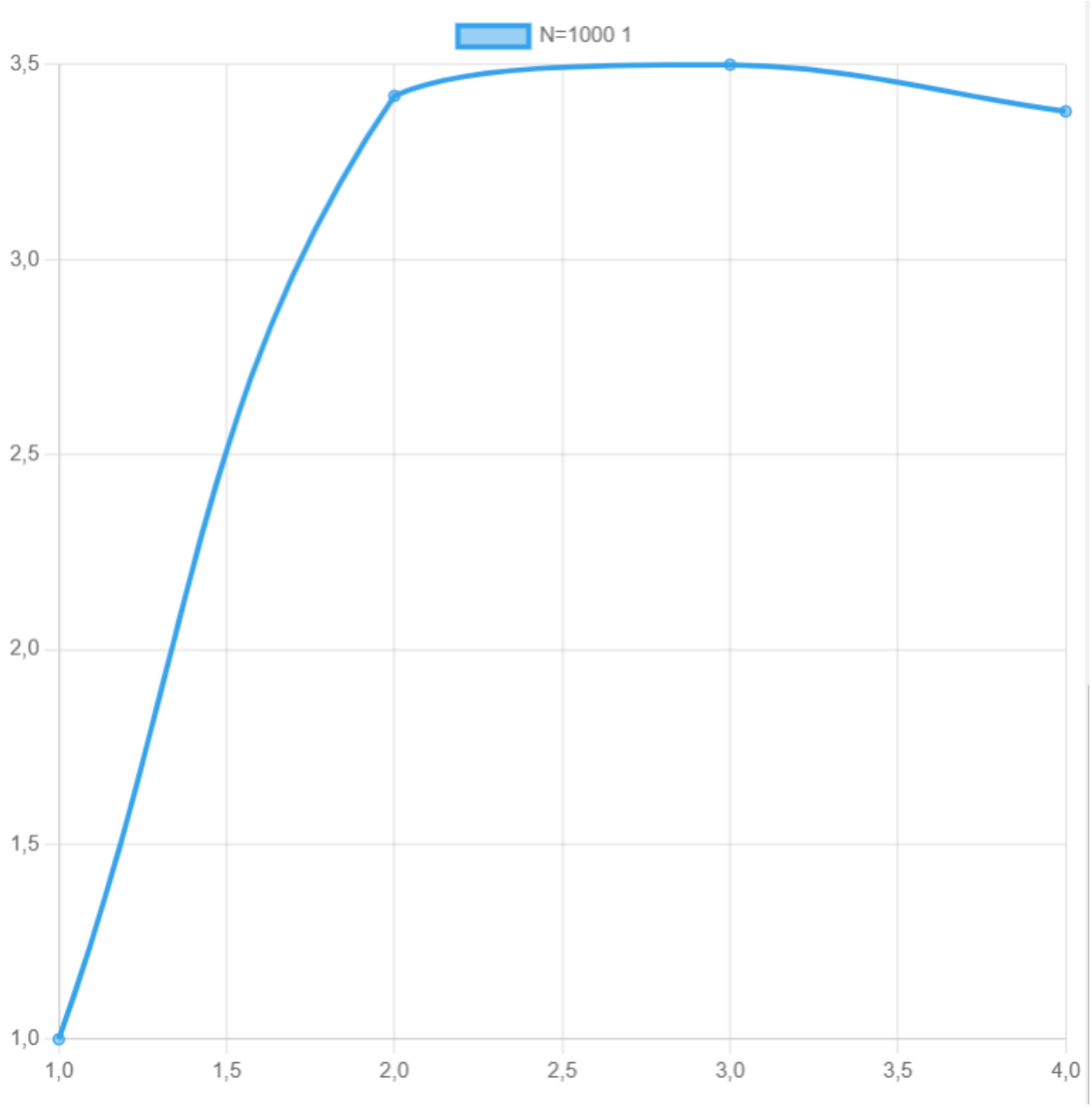


Рис. 3.8. Графік зміни коефіцієнту прискорення при N=1000,

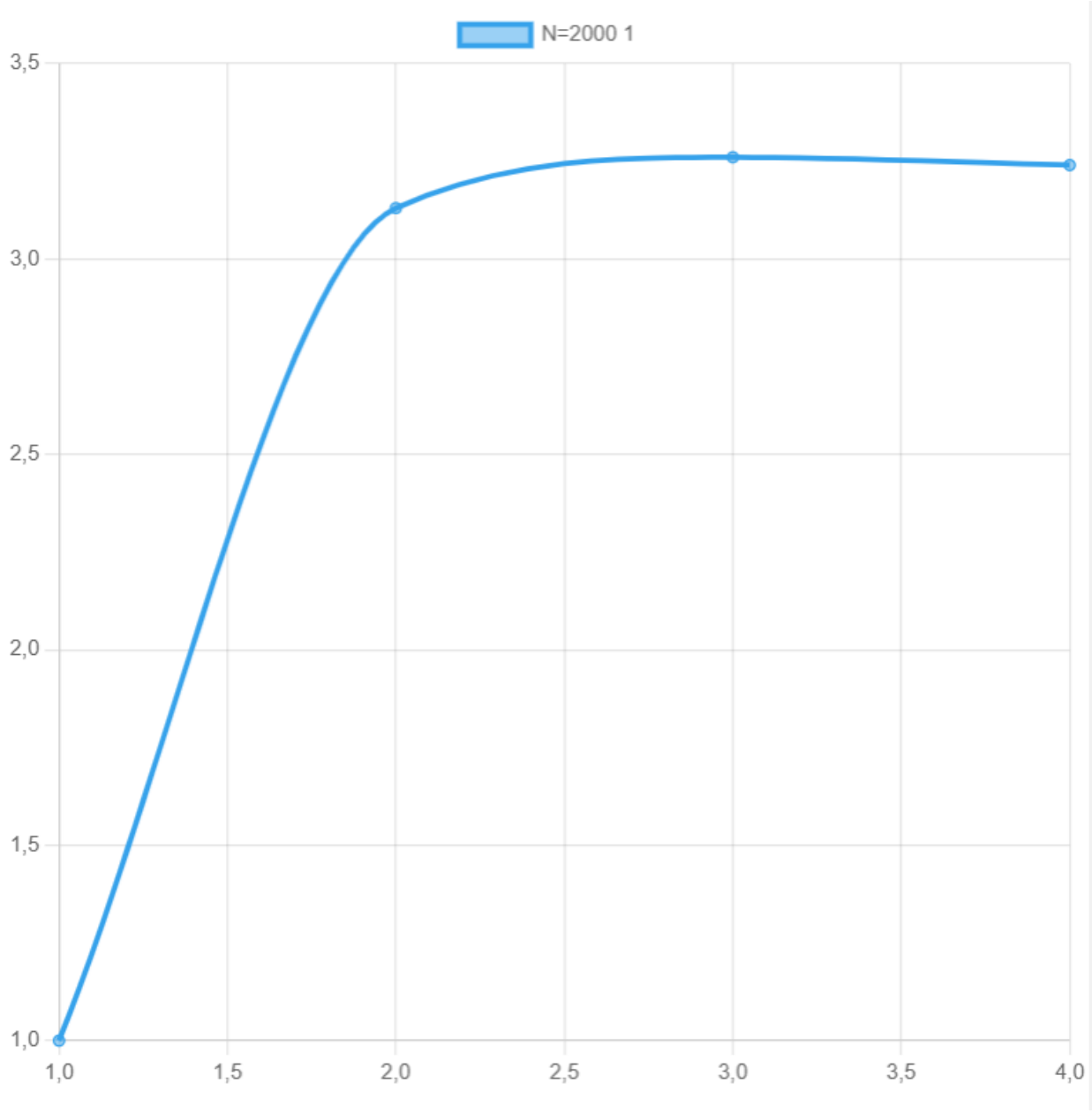


Рис. 3.9. Графік зміни коефіцієнту прискорення при N=2000

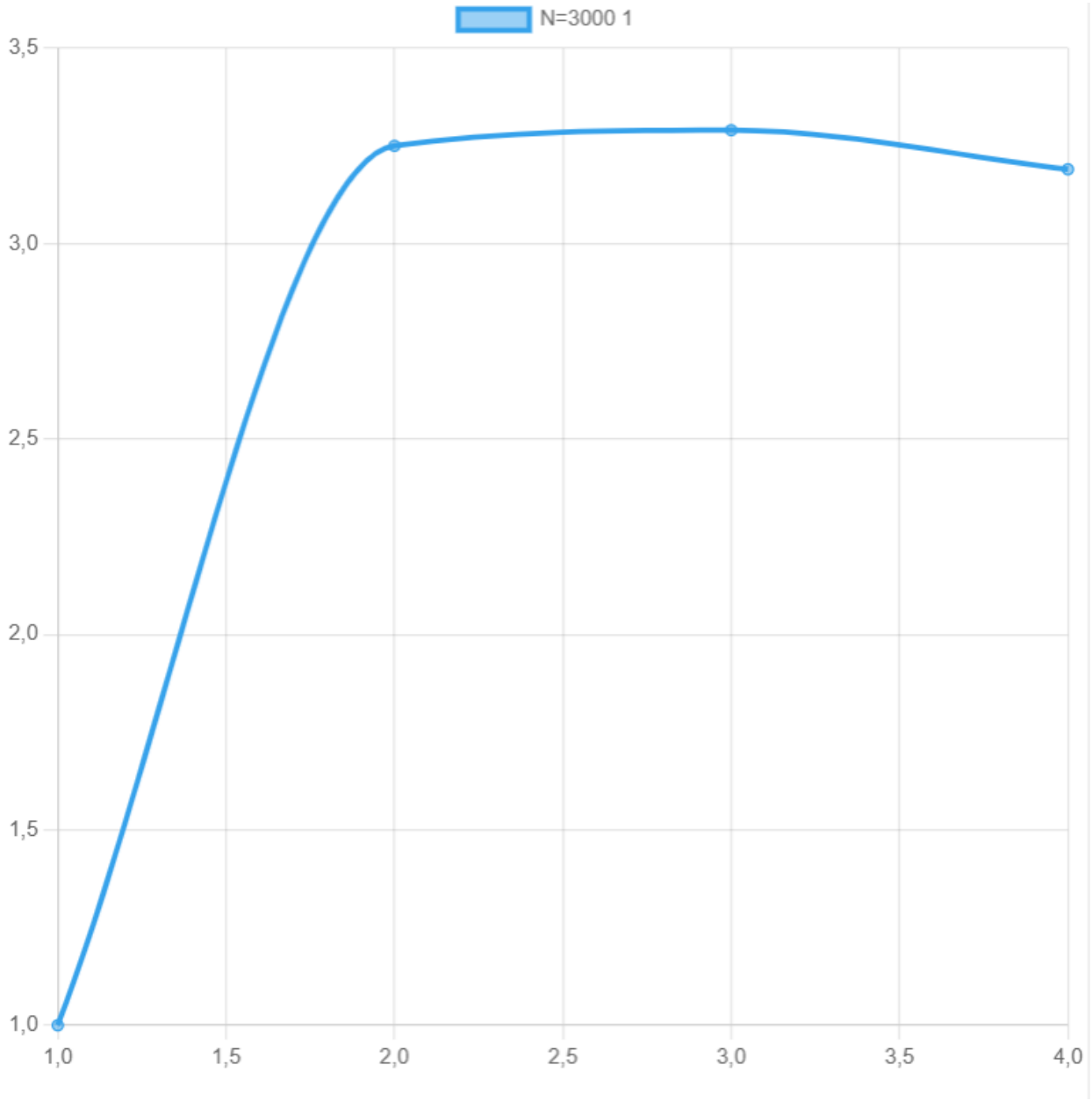


Рис. 3.10. Графік зміни коефіцієнту прискорення при N=3000

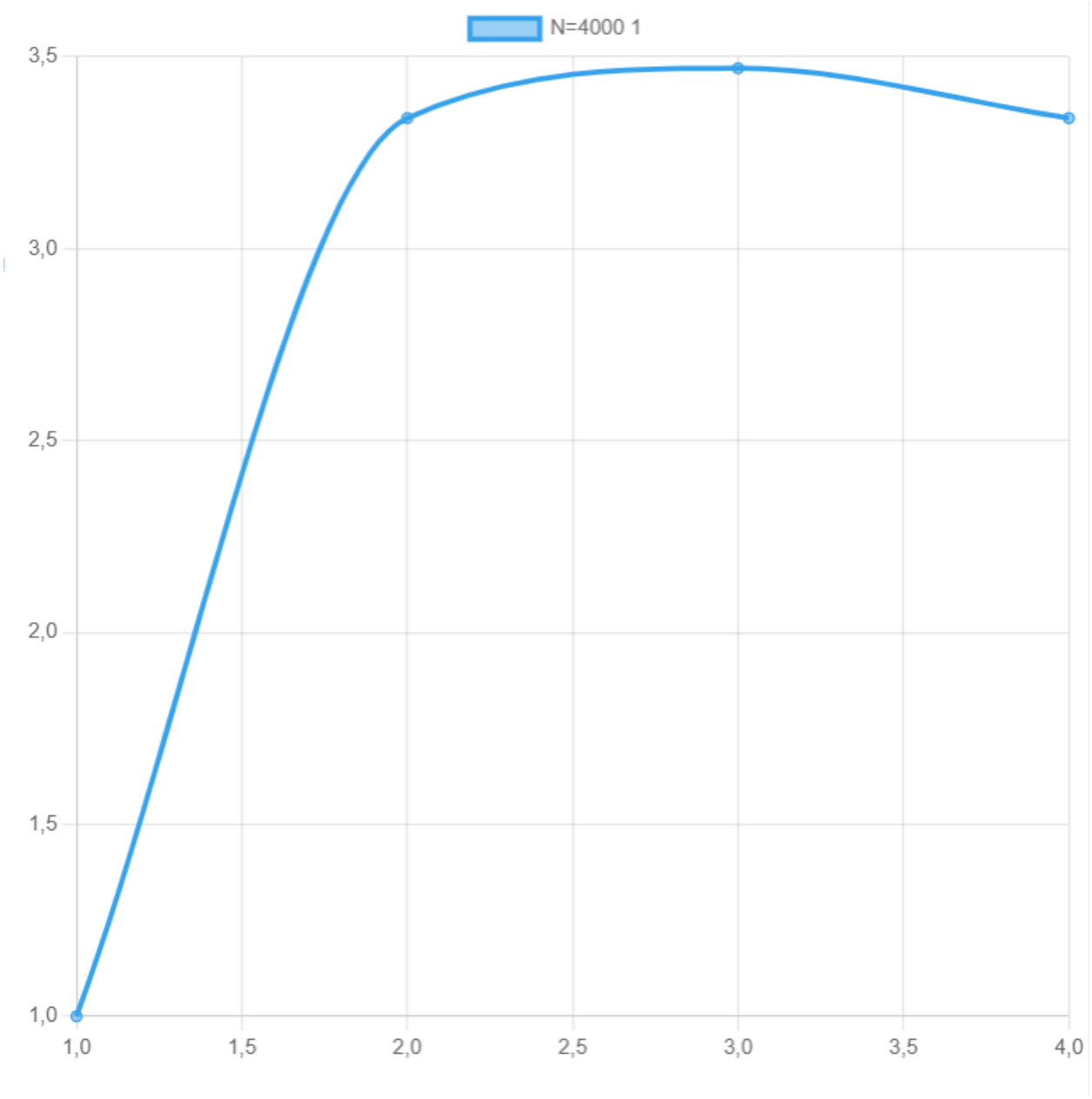


Рис. 3.11. Графік зміни коефіцієнту прискорення при N=4000

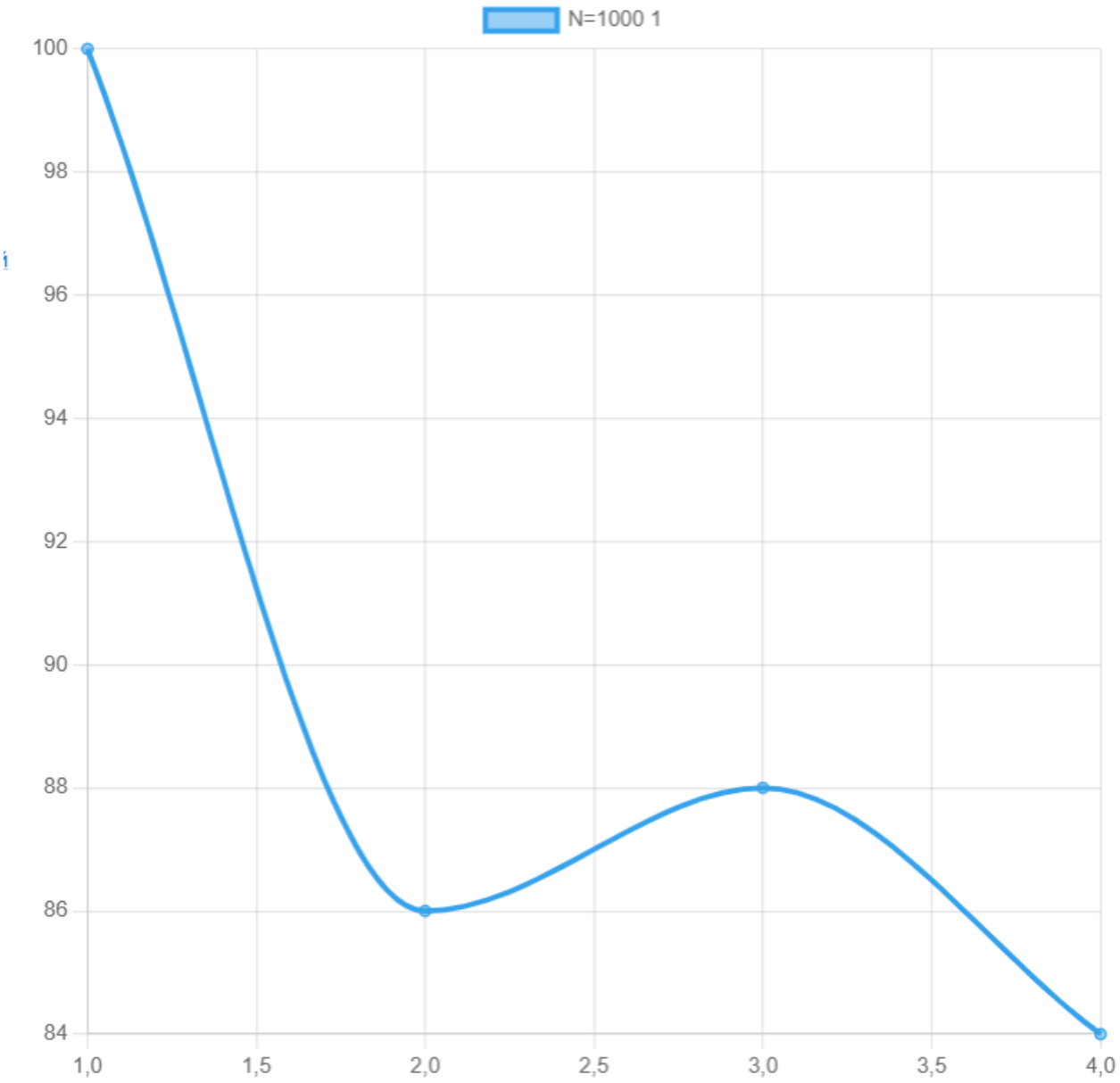


Рис. 3.12. Графік зміни коефіцієнту ефективності при N = 1000

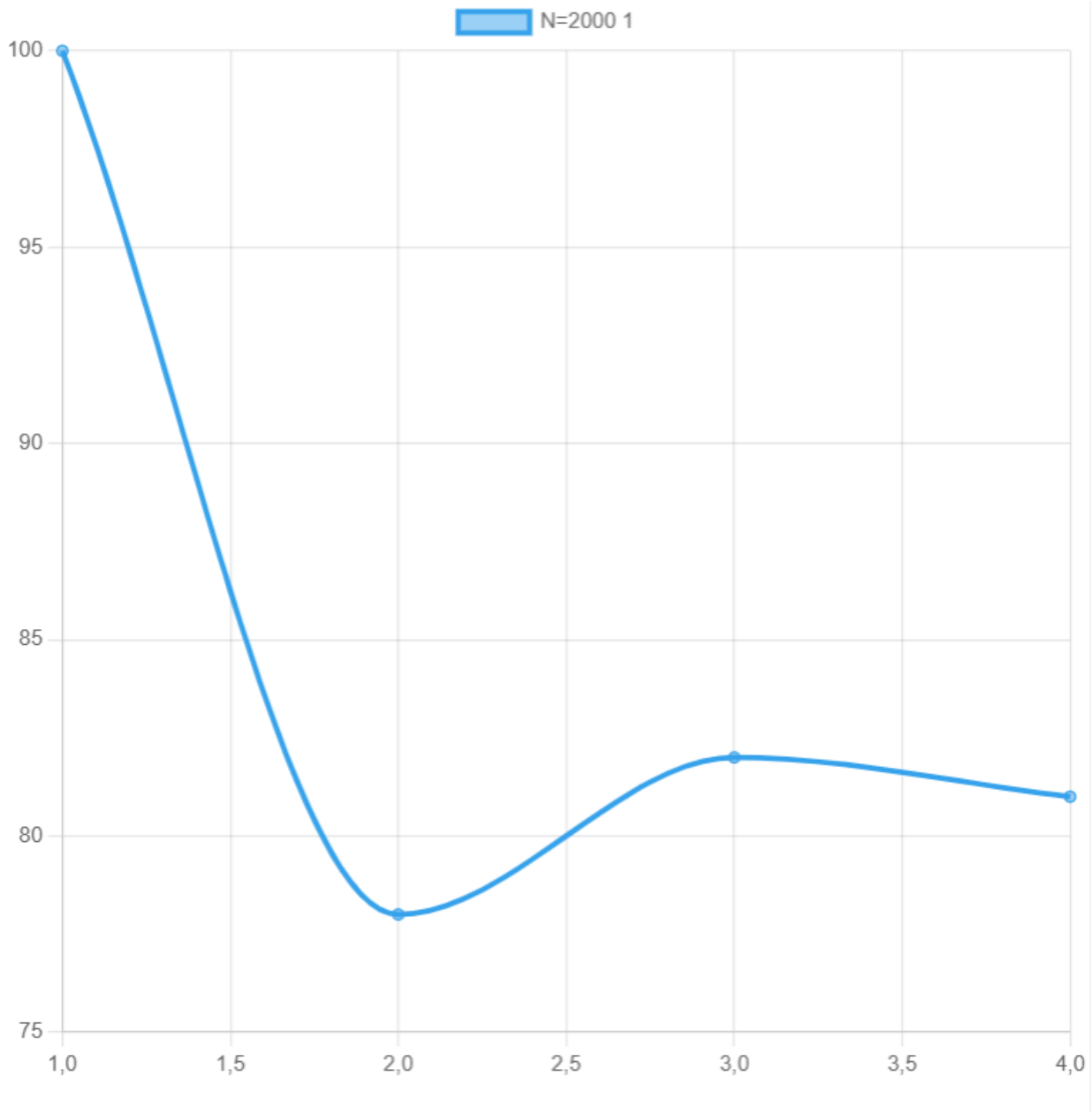


Рис. 3.13. Графік зміни коефіцієнту ефективності при N = 2000

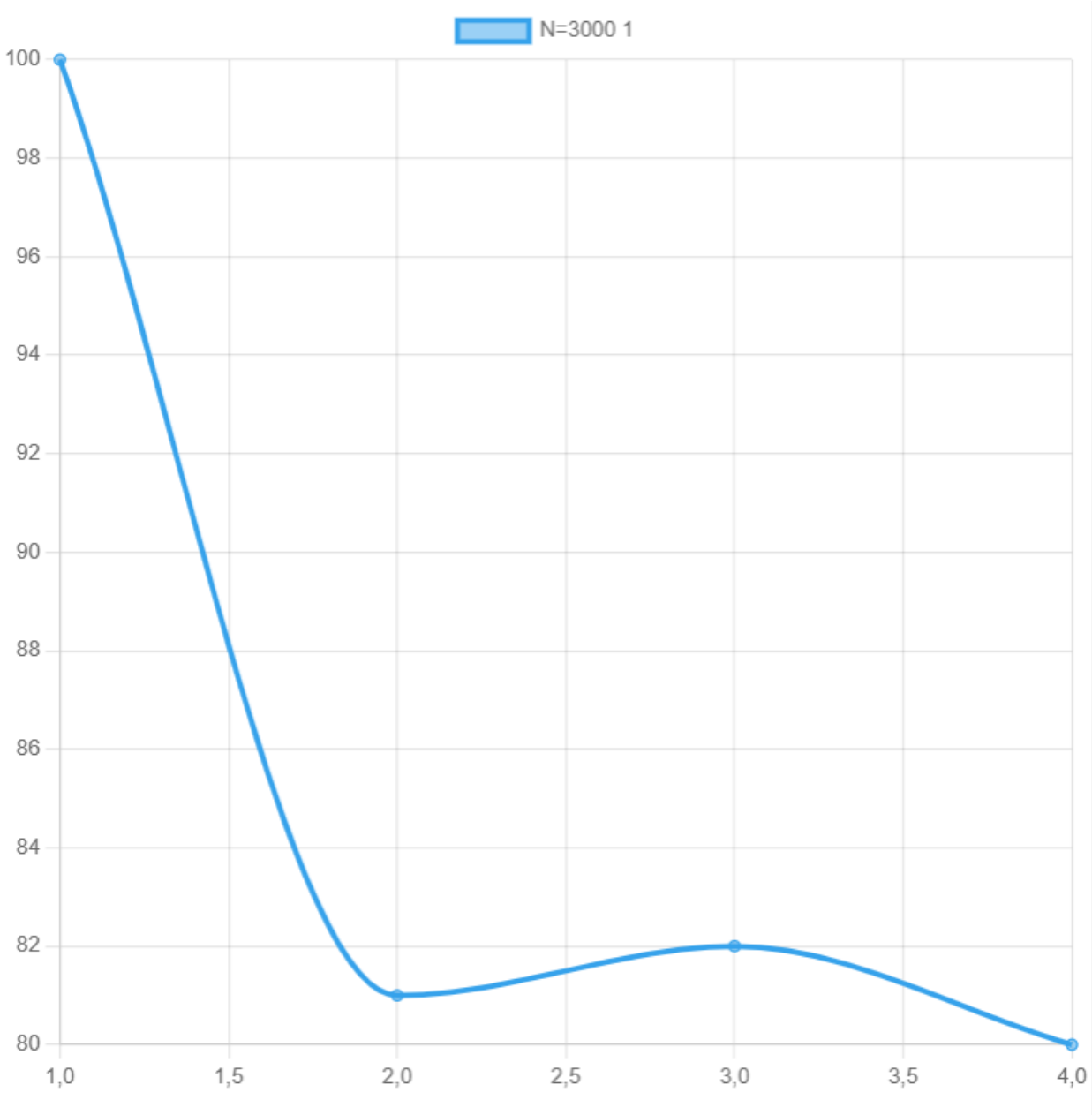


Рис. 3.14. Графік зміни коефіцієнту ефективності при N = 3000

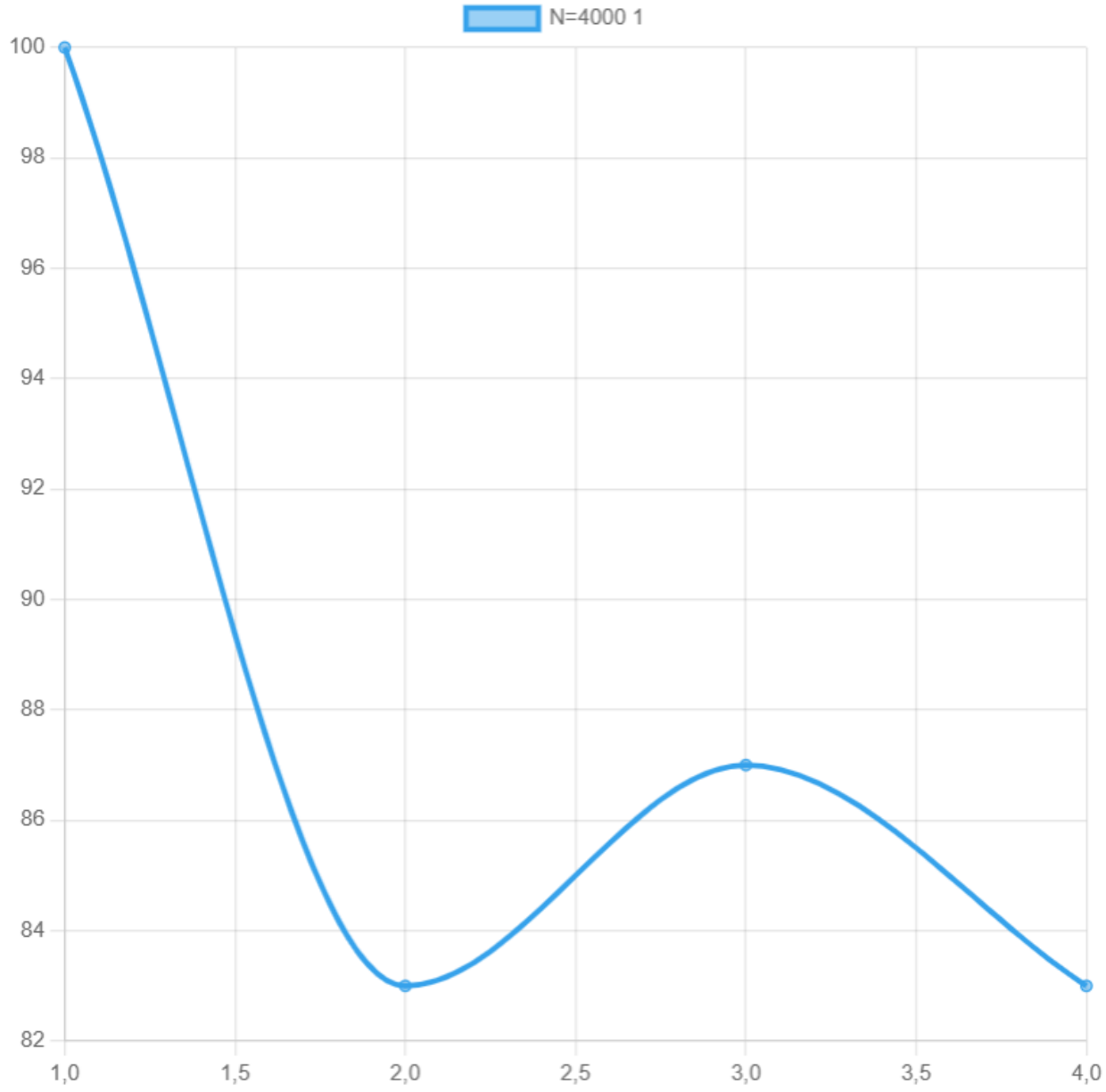


Рис. 3.15. Графік зміни коефіцієнту ефективності при N = 4000

Проаналізувавши отримані графіки можна зауважити, що результати, які є насправді дещо відрізняються від очікуваних результатів. Причиною цьому є накладні витрати, такі як час синхронізації, або інші, які не були враховані спочатку, але мають вплив на результат, хоч і інколи не дуже значний. Ще однією причиною може бути робота інших програм паралельно до виконуваного тестового запуску. Через це значення будуть дещо гіршими

очікуваних, а також час роботи програми на одній вибірці з різною кількістю запусків може відрізнятися.

Також варто зауважити, що використання 4 потоків із різними підходами до організації паралельних обчислень показало кращі результати, ніж просте розпаралелювання циклу. Найкращий результат був досягнутий завдяки оптимальному управлінню потоками, що включало використання гнучкого розподілу задач (дрібнозернистого паралелізму) та утворення логічних потоків для ефективного використання апаратних ресурсів. А саме використання 2 pragma for у циклі

ВИСНОВОК ДО РОЗДІЛУ 3

В результаті було розроблено апаратне та програмне забезпечення програмно-апаратного комплексу. До апаратної частини обрані компоненти, які відповідають складності поставленого завдання і одночасно не є занадто дорогими, що підвищує раціональність створеної системи. Також варто зауважити, що частина компонентів обрана таким чином, щоб не створювати проблем у майбутньому під час модифікування високопродуктивного ПАК.

Також розроблено програму для виконання складного математичного виразу, який вміщує в собі матриці. Для реалізації використано мову програмування C та набір директив OpenMP. Під час організації роботи потоків було використано бар'єри, для забезпечення правильного виконання завдання. Перший бар’єр використовується для синхронізації введення, другий для синхронізації обчислення фінального значення виразу.

Написаний код прокоментовано в самому коді, а також детально пояснено, що виконують окремі функції і для чого вони потрібні. В результаті найкращим підходом було викристання 4 потоків із використанням 2 pragma for для обчислення прикладу, що підтверджується графіками прискорення та ефективності

Після створення програмної та апаратно частини проведено тестування, яке показало, що створена система дійсно працює ефективно. Отримані дані дещо відрізняються від очікуваних. Це спричинено накладними витратами по ефективності під час виконання програми, а також роботою деяких інших програм, які могли повпливати на результати тестувань. Ці впливи є досить незначними, тому якщо немає крайньої необхідності у дуже точних вимірюваннях, ними можна знехтувати.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП ПРОЕКТУ

4.1 Основна ідея стартап-проекту

Основною ідеєю даного стартапу є масштабованої платформи для високопродуктивних обчислень, що надає програмно-апаратні рішення для бізнесу, наукових досліджень і технічних завдань. Основна мета — забезпечити клієнтів доступними інструментами для виконання складних обчислень з максимальною ефективністю та мінімальними витратами. Таблиці 4.1-4.2 містять більш детальну інформацію про особливості проєкту. У таблиці 4.2 визначені сильні, слабкі, та нейтральні сторони стартап проекту і порівняно їх із реальними конкурентами. В колонці оцінки позначені наступні результати: W-гірші значення, N- аналогічні значення, S- кращі значення.

Таблиця 4.1

Опис ідеї стартап-проєкту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Виконання складних обчислювальних завдань, із мінімальними витрати часу та ресурсів	Бізнес-аналітика та великі дані (Big Data)	Прискорення обробки даних
	Наукові обчислення та моделювання	Економія часу
	Розробка програмного забезпечення	Автоматизація оптимізації коду

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Токеномічні характеристики ідеї	(потенційні) товари/концепції конкурентів			Оцінка:
		Мій проєкт	AWS EC2 Instances	Azure HPC + AI	
1	Продуктивність	Забезпечує оптимізацію за рахунок інтеграції паралельних обчислень, адаптованих до конкретних задач.	Висока продуктивність із широкими можливостями для різних задач, але потребує налаштувань	Висока продуктивність для обчислювальних задач, особливо у сфері AI та моделювання.	N
2	Гнучкість використання	Дозволяє налаштовувати апаратні й програмні компоненти для потреб малого та середнього бізнесу.	Масштабованість від невеликих задач до великих, але потребує досвіду в конфігурації.	Висока масштабованість, оптимізована для великих обсягів даних.	N
3	Доступність (вартість послуг)	Орієнтований на доступні рішення для малих і середніх користувачів із помірною вартістю.	Висока вартість для ресурсомістких задач, є знижки для довготривалого використання.	Дещо нижча вартість у порівнянні з AWS, але висока для малого бізнесу.	S
4	Інтеграційні можливості	Просте інтегрування завдяки готовим рішенням для поширених інфраструктур.	Потребує значного налаштування для інтеграції в існуючі системи.	Забезпечує глибоку інтеграцію з екосистемою Microsoft.	S

1.2 Технологічний аудит ідеї проекту

Таблиця 4.3

Технологічна здійсненність проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технологій
1	Виконання складних обчислювальних завдань, із мінімальними витрати часу та ресурсів	Хмарні обчислення	Доступна, широко впроваджена	Середня, присутні тарифи
2		Технологія контейнеризації	Доступна	Висока, оплата базової інфраструктури
3		Паралельні обчислення з використанням GPU	Доступна	Низька(дорогі для постійного використання)
4		Серверні кластери	Доступна	Низька: висока початкова вартість створення та підтримки

1.3 Аналіз ринкових можливостей запуску стартап-проєкту

Запуск стартапу має значний потенціал завдяки зростаючому попиту на високопродуктивні обчислювальні потужності в галузях машинного навчання, аналізу великих даних та моделювання. Ринок B2B клієнтів (компанії з IT, наукових досліджень та фінансового сектора) демонструє сталий ріст, особливо в сегментах, де потрібна висока точність та швидкість обчислень. Конкурентна перевага проекту — поєднання оптимізації витрат і доступності

передових технологій — дозволяє залучати клієнтів із середнього та малого бізнесу, які не можуть дозволити собі дорогі інфраструктури.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проєкту

№	Показник стану ринку	Характеристика
1	Кількість головних гравців, од	8-10
2	Загальний обсяг продаж, usd/ум.од	$50-70 млрд (глобальний ринок хмарних і високопродуктивних обчислень)
3	Динаміка ринку (якісна оцінка)	Зростаюча
4	Наявність обмежень для входу (вказати характер обмежень)	Високі витрати на інфраструктуру, висока конкуренція
5	Специфічні вимоги до стандартизації та сертифікації	ISO, GDPR (залежить від регіону)
6	Середня норма рентабельності в галузі (або по ринку), %	15-20%
7	Попит	Високий у B2B, особливо в AI, FinTech, науці.
8	Технологія	Інноваційна, швидкозмінна
9	Зміни в міжнародній торгівлі і прямих інвестиціях	Сприятливі для експорту IT-рішень
10	Інтенсивність конкуренції	Висока (AWS, Google, Azure — головні гравці)
11	Швидкість зміни умов ринку	Висока (технології швидко оновлюються)
12	Доступність інвестицій	Середня (залежить від регіону та бізнес-плану)

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності поведінки потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Високопродуктивні обчислення для аналізу даних	IT-компанії, наукові установи, великі корпорації	IT-компанії шукають масштабовані рішення, наукові установи потребують точності, а корпорації — надійності та відповідності нормам	Висока продуктивність, стабільність роботи, гнучкість у конфігурації
2	Паралельна обробка великого обсягу завдань	Фінансові установи, страхові компанії, бізнес-аналітика	Фінансові установи акцентують увагу на швидкості, страхові компанії — на точності, бізнес-аналітики — на інтеграції з іншими системами	Швидка обробка задач, зручний інтерфейс, підтримка інтеграції з іншими платформами
3	Оптимізація витрат на інфраструктуру	Малі та середні підприємства, стартапи	Малі підприємства обирають економічні рішення, стартапи — швидкий запуск та масштабованість	Низька вартість, простота впровадження, можливість поступового масштабування
4	Використання передових технологій для підвищення конкурентоспроможності	Компанії у сфері штучного інтелекту (AI), медіа, та геймінгу	AI-компанії орієнтуються на спеціалізовані потужності, медіа шукають стабільності під час пікових навантажень, геймінг — низьку затримку обчислень	Передові технології, адаптивність до специфічних завдань, стабільність під час інтенсивного використання

Таблиця 4.6

Аналіз факторів загроз

№ п/п	Фактор загрози	Опис загрози	Планове реагування компанії
1	Висока конкуренція	Велика кількість сильних гравців (AWS, Google Cloud, Azure) може ускладнити вихід на ринок.	Розробка унікальної ціннісної пропозиції, фокус на нішевих ринках і персоналізація послуг.
2	Висока вартість інфраструктури	Значні витрати на придбання або оренду обладнання можуть зменшити рентабельність проекту.	Використання хмарних технологій для зменшення початкових витрат; співпраця з постачальниками на вигідних умовах.
3	Зміна регуляторних норм	Нові закони про захист даних (наприклад, GDPR) або локальні обмеження можуть вплинути на бізнес.	Адаптація послуг до міжнародних і локальних стандартів; регулярний моніторинг правового середовища.
4	Технологічні збої	Збої в роботі серверів або програмного забезпечення можуть призвести до втрати клієнтів.	Інвестування у резервні системи, регулярне тестування та обслуговування обладнання.
5	Зміна умов ринку	Швидкий розвиток технологій може зробити обрані рішення застарілими.	Постійний моніторинг технологій, регулярне оновлення пропозиції та гнучкість у розробці нових продуктів.

Таблиця 4.7

Аналіз можливостей

№ п/п	Фактор можливості	Зміст можливості	Можлива реакція компанії
1	Зростаючий попит на хмарні обчислення	Все більше компаній переходять на хмарні сервіси для оптимізації витрат та підвищення продуктивності.	Активна маркетингова кампанія, розробка гнучких тарифних планів і масштабованих послуг для різних сегментів.
2	Попит на спеціалізовані обчислювальні послуги	Галузі, такі як AI, FinTech, медична діагностика, потребують адаптованих рішень для обчислень.	Розробка спеціалізованих модулів і оптимізованих рішень для вузьких сегментів ринку.
3	Міжнародна експансія	Ринок високопродуктивних обчислень активно зростає у країнах, що розвиваються, та в Європі.	Вихід на нові ринки через партнерства, локалізацію послуг і адаптацію до місцевих умов.

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	Прояв характеристики конкурентного середовища	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції	Комбінація глобальної та локальної конкуренції в залежності від сегменту ринку.	Пошук нішевих рішень, адаптація продуктів до регіональних особливостей, диференціація послуг.
Рівень конкурентної боротьби	Велика кількість гравців, домінування великих корпорацій (AWS, Google Cloud, Microsoft Azure).	Створення унікальної ціннісної пропозиції, орієнтація на специфічні сегменти ринку, партнерства.
Галузева ознака	Швидкий розвиток технологій, необхідність постійного оновлення послуг.	Інвестиції в R&D, регулярний моніторинг ринку, швидке впровадження інновацій.
Конкуренція за видами товарів	Велика різноманітність пропозицій: від базових обчислень до спеціалізованих рішень для AI.	Розробка інтегрованих продуктів, гнучке ціноутворення, відповідність потребам клієнтів.
Характер конкурентних переваг	Перевага великих гравців у масштабованості, вартості, швидкості впровадження.	Оптимізація ресурсів, створення доступних за ціною та швидких рішень, акцент на якість послуг.
Інтенсивність конкуренції	Високий рівень конкуренції між глобальними та регіональними компаніями.	Посилення маркетингової стратегії, просування бренду, забезпечення високого рівня сервісу.

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товаризамінники
	Домінування великих гравців (AWS, Google Cloud, Azure), висока інтенсивність конкуренції, значна частка ринку лідерів.	Високий поріг входу через необхідність інвестицій у інфраструктуру, але можливість нових гравців із інноваційними рішеннями.	Постачальники серверів, процесорів (Intel, AMD), хмарних рішень впливають на вартість та доступність ресурсів.	Високі очікування щодо якості, швидкості та вартості послуг, потреба в адаптації продуктів під індивідуальні потреби.	Технології, що пропонують інші способи вирішення задач (локальні сервери, спеціалізовані обчислювальні пристрої).
Висновки	Необхідно створювати нішеві рішення, інвестувати в маркетинг та формувати лояльність клієнтів.	Інвестиції в інновації, створення високих бар'єрів входу через стандарти та репутацію.	Налагодження довгострокових партнерств для отримання вигідних умов постачання.	Розробка персоналізованих рішень, оптимізація процесів для зниження вартості.	Підвищення ефективності та конкурентоспроможності, акцент на гнучкості та масштабованості.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкуренто-спроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Інноваційність технологій	Використання сучасних алгоритмів паралельних обчислень та оптимізації дозволяє пропонувати унікальні рішення.
2	Низька вартість послуг	Оптимізація ресурсів і використання ефективного хмарного середовища знижує витрати на обчислення.
3	Швидкість обчислень	Висока продуктивність програмно-апаратного комплексу забезпечує мінімальні затримки у виконанні задач.
4	Персоналізація рішень	Можливість адаптації послуг під специфічні вимоги клієнтів забезпечує широку аудиторію користувачів.
5	Простота використання	Інтуїтивний інтерфейс та автоматизація процесів спрощують роботу кінцевих користувачів із платформою.
6	Висока масштабованість	Можливість обробляти як невеликі, так і великомасштабні завдання розширює спектр застосувань продукту.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін стартап-проекту

№ п/п	Фактор конкуренто-спроможності	Бали 1-20	Порівняння рейтингу товарів-конкурентів						
			-3	-2	-1	0	+1	+2	+3
1	Інноваційність технологій	11		+					
2	Низька вартість послуг	18					+		
3	Швидкість обчислень	16					+		
4	Персоналізація рішень	15				+			
5	Простота використання	17						+	
6	Висока масштабованість	17						+	

Таблиця 4.12

SWOT-аналіз стартап-проекту

Сильні сторони	**Слабкі сторони**
- Інноваційність технологій для складних обчислень. - Низька вартість послуг завдяки оптимізації ресурсів. - Висока швидкість обчислень для виконання складних завдань. - Простота використання продукту для кінцевих користувачів.	- Потенційно високі початкові витрати на розробку та підтримку інфраструктури. - Обмежене визнання бренду на початкових етапах розвитку. - Можливість низької лояльності клієнтів на ранніх етапах запуску проекту.

- Масштабованість рішення для різних потреб клієнтів.	
Можливості - Розширення ринку за рахунок підвищення попиту на хмарні обчислення та високопродуктивні рішення. - Співпраця з іншими компаніями для вдосконалення продукту та залучення нових клієнтів. - Зростаючий попит на персоналізовані рішення та автоматизацію складних задач.	**Загрози** - Висока конкуренція на ринку з боку великих гравців, таких як Amazon, Microsoft та Google. - Швидкий розвиток технологій може призвести до появи нових конкурентів із більш ефективними рішеннями. - Зміни в нормативно-правовому середовищі, які можуть вплинути на функціонування стартапу.

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Пошук інвестицій через венчурні фонди	Висока	6-12 місяців
2	Партнерство з великими технологічними компаніями	Середня	12-18 місяців
3	Самофінансування та запуск через краудфандингові платформи	Середня	6-8 місяців

4.4 Розроблення ринкової стратегії проекту

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Малий та середній бізнес (SMB)	Висока: прагнуть покращити ефективність обчислень.	Високий, оскільки багато таких компаній потребують технологічних рішень для обробки даних	Середня: вже є кілька основних постачальників послуг.	Середня: бізнеси з обмеженими бюджетами можуть стикатися з труднощами у старті..
2	Фінансові та консалтингові компанії	Дуже висока: ці компанії постійно шукають рішення для обробки фінансових даних.	Дуже висока: ці компанії постійно шукають рішення для обробки фінансових даних.	Дуже висока: ці компанії постійно шукають рішення для обробки фінансових даних.	Дуже висока: ці компанії постійно шукають рішення для обробки фінансових даних.

Продовження Таблиці 4.14

3	Університети та науково-дослідні установи	Організації, що проводять наукові дослідження та мають потребу в потужних обчислювальних можливостях для моделювання та аналізу великих наукових даних.	Організації, що проводять наукові дослідження та мають потребу в потужних обчислювальних можливостях для моделювання та аналізу великих наукових даних.	Організації, що проводять наукові дослідження та мають потребу в потужних обчислювальних можливостях для моделювання та аналізу великих наукових даних.	Організації, що проводять наукові дослідження та мають потребу в потужних обчислювальних можливостях для моделювання та аналізу великих наукових даних.
Обрані цільові групи: Малий та середній бізнес (SMB).					

Малий та середній бізнес (SMB) був обраний тому, що ці компанії часто стикаються з обмеженими ресурсами і шукають економічно ефективні технологічні рішення, конкуренція в цьому сегменті є середньою, простота входу на цей ринок є середньою, проте з правильними підходами, адаптованими рішеннями, стартап має потенціал отримати великий ринковий сегмент.

Таблиця 4.15

Визначення базової стратегії розвитку

п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
	Розвиток рішення для малого та середнього бізнесу (SMB)	зосередження на високій продуктивності за доступною ціною для малих та середніх підприємств.	Висока ефективність, Доступність ціни, Інтуїтивно зрозумілий інтерфейс	Стратегія диференціації

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першо-прохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	НІ	Компанія буде шукати нових споживачів, орієнтуючись на малий та середній бізнес	Компанія не буде копіювати товарів конкурентів, а замість цього орієнтуватиметься на інноваційний підхід та поліпшення ефективності обчислень у своїх рішеннях.	Стратегія диференціації

Таблиця 4.17

Визначення стратегії позиціонування

п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспро можні позиції власного стартап проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Висока ефективність, мінімальні витрати часу та ресурсів, зручність у використанні.	Виведення на ринок через унікальні обчислювальні рішення.	Інноваційність рішення, висока продуктивність, зниження витрат на обчислення при високих навантаженнях.	Швидкість**,** Інноваційність**,** Ефективність
2	Спеціалізація на високопродуктивних обчисленнях для специфічних галузей (наприклад, фінанси, наука).	Охоплення спеціалізованих сегментів ринку.	Глибоке розуміння потреб конкретних галузей, адаптовані продукти для вузьких сегментів ринку.	Продуктивність, Галузева орієнтованість, Висока точність

1.4 Розроблення маркетингової програми стартап-проєкту

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Швидкість обробки даних	Товар дозволяє здійснювати обчислення в реальному часі, забезпечуючи високопродуктивну обробку даних без затримок.	Висока ефективність обробки в порівнянні з традиційними рішеннями; мінімальні затрати часу на обчислення.
2	Оптимізація використання ресурсів	Програмне забезпечення забезпечує економію ресурсів при паралельних обчисленнях, знижуючи навантаження на систему.	Індивідуальні алгоритми оптимізації, що знижують витрати обчислювальних ресурсів; зниження енергоспоживання.
3	Масштабованість та гнучкість	Платформа легко адаптується до різних обчислювальних навантажень і може масштабуватися при збільшенні обсягів роботи.	Можливість масштабувати систему для специфічних потреб; гнучкість налаштувань для різних галузей та задач.

Таблиця 4.19

Формування системи збуту стартап-проекту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Пошук ефективності та економії ресурсів.	Прямі продажі, техпідтримка, навчання.	Прямі онлайн-продажі та партнерства.	Прямі онлайн-продажі та партнерства.

Таблиця 4.20

Концепція маркетингових комунікацій стартап-проекту

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
	Шукають інноваційні, високопродуктивні рішення для обробки даних.	Соціальні мережі, професійні форуми, вебінари, технічні блоги.	Висока продуктивність, економія часу та ресурсів, простота інтеграції.	Показати ефективність і швидкість рішення, акцент на інноваційність і оптимізацію	Максимальна продуктивність за мінімальний час і ресурси

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі були розглянуті ключові аспекти маркетингових комунікацій стартап-проекту, які є основою для ефективної взаємодії з цільовими клієнтами. Специфіка поведінки цільових клієнтів вказує на їх прагнення до інноваційних і високопродуктивних рішень, що є важливою основою для формування рекламних стратегій. Вибір правильних каналів комунікацій, таких як соціальні мережі та професійні платформи, дозволяє ефективно досягти цільову аудиторію.

Ключові позиції, обрані для позиціонування продукту, акцентують увагу на продуктивності та економії ресурсів, що є важливим для потенційних клієнтів.

Завдання рекламного повідомлення полягає в демонстрації ефективності продукту, а концепція рекламного звернення підкреслює швидкість і оптимізацію рішення. Все це сприятиме успішному впровадженню стартап-проекту на ринок, забезпечуючи його конкурентоспроможність.

ВИСНОВКИ

В результаті дослідження різних лінійок процесорів двох провідних компаній було прийнято рішення обрати процесор Intel Core i5 8300H. Базуючись навколо нього підібрані компоненти, які мають найкраще співвідношення ціни до необхідної потужності, а також впринципі можуть працювати разом, утворюючи високопродуктивний ПАК. Конкретні характеристики процесора, а також його тестів можна побачити у додатках №1 та №2.

У другому розділі проведено дослідження різних мов програмування та засобів, які дозволяють використовувати паралельне програмування. В результаті обрано мову програмування C та бібліотеку OpenMP. Конкретно показані методи, які наявні у бібліотеці і пояснено, чому вони підходять краще за інші.

Результатом третього розділу є створення програмної та апаратної частини. Апаратна частина базується навколо обраного процесора. Компоненти підібрані таким чином, аби забезпечити найкраще співвідношення ціни до ефективності, а також не створювати проблем при модифікації ПАКу у майбутньому. Кошторис Апаратної частини можна знайти у додатку №3

Створено програму із використанням мови програмування C та бібліотеки OpenMP. За основу завдання обрано приклад із лінійної алгебри. Створили кілька варіантів виконання завдання із різним використанням кількості директив #pragma for. Протестували результати кожного варіанту, показали отримані результати у вигляді таблиць з часом виконання, присоренням, та ефективності. Після проведеного аналізу обрано найкращий варіант із використанням 2 директив pragma for. Підтвердженням цьому є побудовані графіки прискорення та ефективності.

Четвертий розділ описує розробку стартап-проекту по просуванню високопродуктивного програмно-апаратного комплексу. Досліджено конкуренцію на ринку, а також ринок потенційних клієнтів. Основним напрямком було обрано нішу малого та середнього бізнесу, так як вони не завжди мають достатньо коштів для дорогих рішень. Окрім цього їм потрібні достатньо ефективні рішення для специфічних проблем, які зможе вирішити даний стартап.

Отже написана робота підтверджує що створений комплекс має наукову та практична значущість, яка полягає у розробці універсального підходу до створення високопродуктивних обчислювальних систем, який може бути адаптований для вирішення задач у різних галузях, таких як обробка великих даних, машинне навчання, наукові дослідження та інженерні розрахунки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Стан і тенденції розвитку сучасних процесорів. Stud. URL: https://stud.com.ua/94230/informatika/stan_tendentsiyi_rozvitku_suchasnih_protsesoriv
2. Тенденції розвитку сучасних комп'ютерних систем. URL: https://comsys.kpi.ua/upload/Тенденція_розвитку_сучасних_компютерних_систем_2022.pdf.
3. Дані про сучасні процесори та перспективи їх розвитку?. URL: https://znanija.com/task/53499208.
4. Огляд та тестування флагманських процесорів Intel Core i9-14900K та Core i9-14900KF. URL: https://www.overclockers.ua/ua/cpu/intel-core-i9-14900kf-i7-14700k-amd-ryzen-9-7950x3d-7-7800x3d/all/.
5. Огляд та тестування флагманських процесорів Intel Core i9-14900K та Core i9-14900KF. URL: https://www.overclockers.ua/ua/cpu/intel-core-i9-14900kf-i7-14700k-amd-ryzen-9-7950x3d-7-7800x3d/all/.
6. Порівняння процесорів. URL: https://www.overclockers.ua/ua/cpu/info/compare/1007/.
7. Огляд AMD Ryzen 7 8700G: найпотужніший у світі APU для домашніх комп'ютерів, або процесор для всього й відразу. ITC.ua. URL: https://itc.ua/ua/articles/oglyad-amd-ryzen-7-8700g-najpotuzhnishyj-u-sviti-apu-dlya-domashnih-komp-yuteriv-abo-protsesor-dlya-vsogo-j-vidrazu/
8. ProIT. Огляд AMD Ryzen 7 7800X3D: найкращий ігровий процесор сучасності. ProIT. URL: https://proit.com.ua/tech/oglyad-amd-ryzen-7-7800x3d-najkrashhyj-igrovyj-protsesor-suchasnosti/
9. Що краще процесори Intel та AMD ☑ порівняння продуктивності ☑ Огляд Artline 2022. Збірка ПК онлайн у Києві ☑ Магазин комп'ютерів Artline ► зібрати ПК. URL: https://artline.ua/uk/news/sravnenie-protsessorov-intel-i-

amd?srsltid=AfmBOorn9QcCjQgCw0LE2EJz7Ga57R0Qj3rYuvRe49FWiUp5hsPR0wiH

10. Порівняння процесорів ryzen і intel: який із них кращий – статті | itechua. iTechua - Новини про смартфони, гаджети і різні девайси. URL: https://itechua.com/articles/220860
11. Порівняння топових процесорів INTEL і AMD | основні відмінності | hotline.ua. Hotline.ua. URL: https://hotline.ua/ua/guides/porvnyannya-procesorv-intel-ta-amd/
12. Система електронного забезпечення навчання ЗНУ. URL: https://moodle.znu.edu.ua/pluginfile.php/693597/mod_resource/content/1/Лекция%203.pdf
13. Еволюція хмарних обчислень. Головна | EDU Blog. URL: https://edublog.com.ua/blog/id1335942996/posts/статті/еволюція-хмарних-обчислень
14. Паралельне програмування: особливості та застосування. FoxmindEd. URL: https://foxminded.ua/paralelne-prohramuvannia/
15. Система електронного забезпечення навчання ЗНУ. URL: https://moodle.znu.edu.ua/pluginfile.php?file=/624022/mod_resource/content/2/Навчальний%20посібник_Гоменюк-1.pdf
16. Примітиви синхронізації: семафори та м'ютекси. URL: https://vseosvita.ua/library/prymityvy-synkhronizatsii-semafory-ta-miuteksy-877107.html
17. Репозитарій ОНУ імені І.І.Мечникова :: elONUar :: Головна. URL: https://dspace.onu.edu.ua/server/api/core/bitstreams/de54e483-b53f-442b-a5ec-b3fd91eff98a/content
18. Що таке Semaphore? Підрахунок, двійкові типи з прикладом. Guru99. URL: https://www.guru99.com/uk/semaphore-in-operating-system.html
19. Ужгородський національний університет. URL: https://www.uzhnu.edu.ua/uk/infocentre/get/45151 (дата звернення

20. Intel® core™ i5-8300h processor.
URL: https://www.intel.com/content/www/us/en/products/sku/134876/intel-core-i58300h-processor-8m-cache-up-to-4-00-ghz/specifications.html
21. Intel core i5-8300h @ 2.30ghz.
URL: https://www.cpubenchmark.net/cpu.php?cpu=Intel+Core+i5-8300H+@+2.30GHz&id=3254

ДОДАТОК 1

Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри

Характеристика процесора Intel Core i58300H

Аркушів 4

Київ 2024

Essentials

Product Collection	8th Generation Intel® Core™ i5 Processors
Code Name	Products formerly Coffee Lake
Vertical Segment	Mobile
Processor Number	i5-8300H
Lithography	14 nm
Recommended Customer Price	$250.00

CPU Specifications

Total Cores	4
Total Threads	8
Max Turbo Frequency	4.00 GHz
Intel® Turbo Boost Technology 2.0 Frequency‡	4.00 GHz
Processor Base Frequency	2.30 GHz
Cache	8 MB Intel® Smart Cache
Bus Speed	8 GT/s
TDP	45 W
Configurable TDP-down Base Frequency	1.80 GHz
Configurable TDP-down	35 W

Supplemental Information

Marketing Status	Discontinued
Launch Date	Q2'18
Embedded Options Available	No
Datasheet	View now

Memory Specifications

Max Memory Size (dependent on memory type)	64 GB
Memory Types	DDR4-2666, LPDDR3-2133
Max # of Memory Channels	2
Max Memory Bandwidth	41.8 GB/s
ECC Memory Supported ‡	No

GPU Specifications

GPU Name‡	Intel® UHD Graphics 630
Graphics Base Frequency	350 MHz
Graphics Max Dynamic Frequency	1.00 GHz
Graphics Video Max Memory	64 GB
Graphics Output	eDP/DP/HDMI/DVI
4K Support	Yes, at 60Hz
Max Resolution (HDMI)‡	4096x2304@30Hz
Max Resolution (DP)‡	4096x2304@60Hz
Max Resolution (eDP - Integrated Flat Panel)‡	4096x2304@60Hz
Max Resolution (VGA)‡	N/A
DirectX* Support	12
OpenGL* Support	4.5
Intel® Quick Sync Video	Yes
Intel® InTru™ 3D Technology	Yes
Intel® Clear Video HD Technology	Yes
Intel® Clear Video Technology	Yes
# of Displays Supported ‡	3

Device ID	0x3E9B

Expansion Options

PCI Express Revision	3.0
PCI Express Configurations ‡	Up to 1x16, 2x8, 1x8+2x4
Max # of PCI Express Lanes	16

Package Specifications

Sockets Supported	FCBGA1440
Max CPU Configuration	1
$T_{JUNCTION}$	100°C
Package Size	42mm x 28mm

Advanced Technologies

Intel® Optane™ Memory Supported ‡	Yes
Intel® Speed Shift Technology	Yes
Intel® Turbo Boost Technology ‡	2.0
Intel® Hyper-Threading Technology ‡	Yes
Intel® Transactional Synchronization Extensions	No
Intel® 64 ‡	Yes
Instruction Set	64-bit
Instruction Set Extensions	Intel® SSE4.1, Intel® SSE4.2, Intel® AVX2
Intel® My WiFi Technology	Yes

Idle States	Yes
Enhanced Intel SpeedStep® Technology	Yes
Thermal Monitoring Technologies	Yes
Intel® Flex Memory Access	Yes
Intel® Identity Protection Technology ‡	Yes

Security & Reliability

Intel® Software Guard Extensions (Intel® SGX)	Yes with Intel® ME
Intel® AES New Instructions	Yes
Secure Key	Yes
Intel® Memory Protection Extensions (Intel® MPX)	Yes
Intel® OS Guard	Yes
Intel® Trusted Execution Technology ‡	No
Execute Disable Bit ‡	Yes
Intel® Stable IT Platform Program (SIPP)	No
Intel® Virtualization Technology (VT-x) ‡	Yes
Intel® Virtualization Technology for Directed I/O (VT-d) ‡	Yes
Intel® VT-x with Extended Page Tables (EPT) ‡	Yes

ДОДАТОК 2

Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри

Тести процесора Intel Core i58300H

Аркушів 2

Київ 2024

Intel Core i5-8300H @ 2.30GHz

Description: Intel UHD Graphics 630

Class: Laptop

Socket: FCLGA1151-2

Clockspeed: 2.3 GHz

Turbo Speed: 4.0 GHz

Cores: 4 **Threads:** 8

Typical TDP: 45 W

TDP Down: 35 W

Cache per CPU Package:
L1 Instruction Cache: 4 x 32 KB
L1 Data Cache: 4 x 32 KB
L2 Cache: 4 x 256 KB
L3 Cache: 8 MB

Memory Support: Max. Memory Size: 64 GB (DDR4-2666, LPDDR3-2133)

Other names: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz, Intel Core i5-8300H CPU @ 2.30GHz

CPU First Seen on Charts: Q2 2018

CPUmark/$Price: 29.83

Overall Rank:
1452nd fastest in multithreading out of 4901 CPUs
1111th fastest in single threading out of 4901 CPUs
341st fastest in out of 1435 Laptop CPUs

Last Price Change: $250.00 USD (2018-04-01)

Average CPU Mark

Multithread Rating

7456

Single Thread Rating

2274

Samples: 3812*
*Margin for error: Low

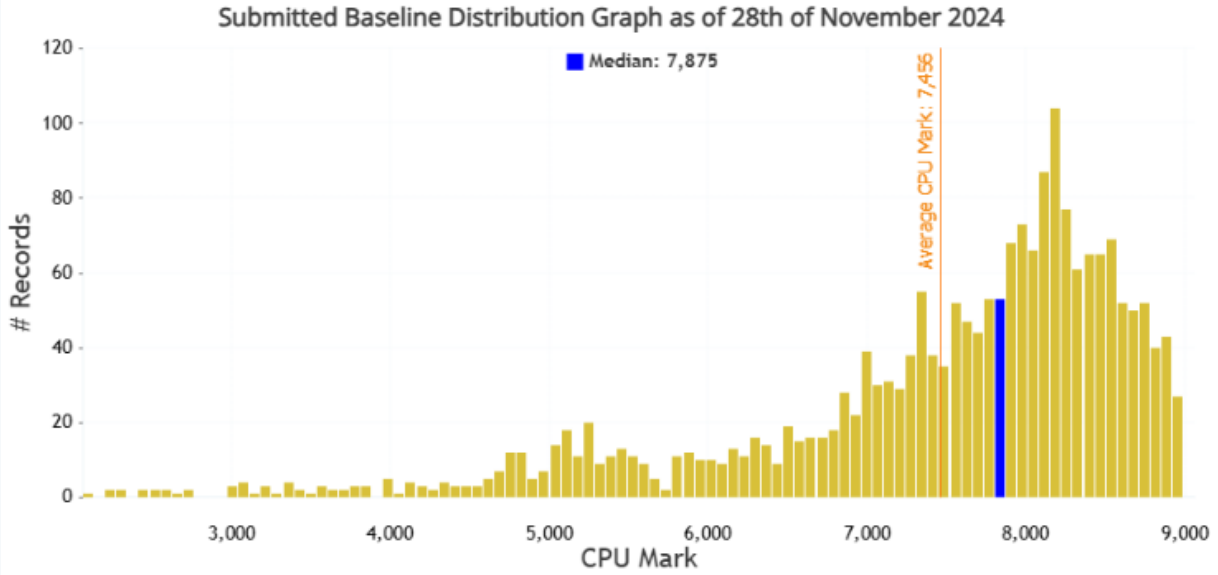
+ COMPARE

PerformanceTest V9
CPU Mark: 9,518
Thread: 2,299

CPU Test Suite Average Results for Intel Core i5-8300H @ 2.30GHz	
Integer Math	24,655 MOps/Sec
Floating Point Math	15,554 MOps/Sec
Find Prime Numbers	23 Million Primes/Sec
Random String Sorting	13,569 Thousand Strings/Sec
Data Encryption	2,562 MBytes/Sec
Data Compression	103,413 KBytes/Sec
Physics	498 Frames/Sec
Extended Instructions	6,494 Million Matrices/Sec
Single Thread	2,274 MOps/Sec

From submitted results to PerformanceTest V10 as of 1st of December 2024.

CPU Mark Distribution for Intel Core i5-8300H @ 2.30GHz



From submitted results to PerformanceTest V10 as of 28th of November 2024.
For distribution graph only: Results are trimmed to exclude outliers by disregarding the bottom 1% and top 1% of submissions.

ДОДАТОК З

Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри

Вартість апаратної частини високопродуктивного ПАК

Аркушів 1

Київ 2024

Компонент	Назва	Ціна, грн.
Процесор	Intel Core i5-8300H	4000
Система охолодження	DeepCool Gammaxx 400EX	1328
Материнська плата	MSI MPG Z790 Carbon WiFi	16632
Оперативна пам’ять	Kingston Beast RGB DDR5	2820
Запам’ятовуючий пристрій	Seagate Barracuda 500GB 3.5	4*650
Відеокарта	GeForce GTX 1660 Super	7173
Блок живлення	Seasonic A12-500 500W	3270
Корпус	Deepcool Matrexx 50	2139
	Сума:	**36362**

ДОДАТОК 4

Високопродуктивний програмно-апаратний комплекс для задач лінійної алгебри

Лістинг основної програми

Аркушів 6

Київ 2024

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

const int N = 4000; // Розмір матриц

// Функція для створення динамічної матриці
double** create_matrix() {
    double** matrix = (double**)malloc(N *
sizeof(double*));
    if (!matrix) {
        fprintf(stderr, "Memory allocation failed for
matrix rows\n");
        exit(EXIT_FAILURE);
    }
    matrix[0] = (double*)malloc(N * N * sizeof(double));
    if (!matrix[0]) {
        fprintf(stderr, "Memory allocation failed for
matrix data\n");
        free(matrix);
        exit(EXIT_FAILURE);
    }
    for (int i = 1; i < N; i++) {
        matrix[i] = matrix[0] + i * N;
    }
    return matrix;
}

// Функція для звільнення пам'яті динамічної матриці
void free_matrix(double** matrix) {
    free(matrix[0]);
    free(matrix);
}

// Функція для заповнення матриць випадковими числами
void fill_matrix(double** matrix) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = rand() % 11; // Випадкове
число від 0 до 10
```

```
        }
    }
}

// Функція для множення матриц
void multiply_matrices(double** result, double** mat1,
double** mat2, int num_threads, int num_pragmas) {
    omp_set_num_threads(num_threads);  // Встановлення
кількості потоків

    // Ініціалізація масиву result
    #pragma omp parallel for collapse(2) if (num_pragmas
>= 1)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = 0.0;
        }
    }

    // Основний цикл для множення матриц
    if (num_pragmas == 1) {
        #pragma omp parallel for
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++) {
                    result[i][j] += mat1[i][k] *
mat2[k][j];
                }
            }
        }
    } else if (num_pragmas == 2) {
        #pragma omp parallel for collapse(2)
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++) {
                    result[i][j] += mat1[i][k] *
mat2[k][j];
                }
            }
        }
    } else if (num_pragmas == 3) {
        #pragma omp parallel for collapse(3)
        for (int i = 0; i < N; i++) {
```

```
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++) {
                    result[i][j] += mat1[i][k] *
mat2[k][j];
                }
            }
        }
    } else {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                for (int k = 0; k < N; k++) {
                    result[i][j] += mat1[i][k] *
mat2[k][j];
                }
            }
        }
    }
}

// Функція для складання двох матриц
void add_matrices(double** result, double** mat1,
double** mat2) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result[i][j] = mat1[i][j] + mat2[i][j];
        }
    }
}

void multiply_matrices_with_timing(double** result,
double** mat1, double** mat2, int num_threads, int
num_pragmas, const char* label) {
    double start_time, end_time;

    // Ініціалізація для запобігання оптимізації
    volatile double sum = 0.0;

    start_time = omp_get_wtime();
    multiply_matrices(result, mat1, mat2, num_threads,
num_pragmas);
    end_time = omp_get_wtime();
```

```
    // Фіктивний підрахунок для запобігання оптимізації
компілятором
    for (int i = 0; i < N; i++) {
        sum += result[i][i];
    }

    printf("%s - Time: %f seconds (Check: %f)\n", label,
end_time - start_time, sum);
}

void add_matrices_with_timing(double** result, double**
mat1, double** mat2, const char* label) {
    double start_time, end_time;
    start_time = omp_get_wtime();
    add_matrices(result, mat1, mat2);
    end_time = omp_get_wtime();
    printf("%s - Time: %f seconds\n", label, end_time -
start_time);
}

int main() {
    double** MB = create_matrix();
    double** MC = create_matrix();
    double** MD = create_matrix();
    double** ME = create_matrix();
    double** MA = create_matrix();
    double** MB_MC = create_matrix();
    double** MD_ME = create_matrix();

    double start_time, end_time;

    // Ініціалізація генератора випадкових чисел
    srand(42);

    // Заповнення матриц випадковими числами
    fill_matrix(MB);
    fill_matrix(MC);
    fill_matrix(MD);
    fill_matrix(ME);

    // Варіант 1: Однопотокова реалізація
    printf("Variant 1: Single core\n");
    start_time = omp_get_wtime();
```

```
    multiply_matrices_with_timing(MB_MC, MB, MC, 1, 0,
"MB x MC");
    multiply_matrices_with_timing(MD_ME, MD, ME, 1, 0,
"MD x ME");
    add_matrices_with_timing(MA, MB_MC, MD_ME, "MA =
MB_MC + MD_ME");
    end_time = omp_get_wtime();
    printf("Total time: %f seconds\n", end_time -
start_time);

    // Варіант 2: Паралельна реалізація з одним pragma
for
    printf("\nVariant 2: 4 cores with 1 pragma for\n");
    start_time = omp_get_wtime();
    multiply_matrices_with_timing(MB_MC, MB, MC, 4, 1,
"MB x MC");
    multiply_matrices_with_timing(MD_ME, MD, ME, 4, 1,
"MD x ME");
    add_matrices_with_timing(MA, MB_MC, MD_ME, "MA =
MB_MC + MD_ME");
    end_time = omp_get_wtime();
    printf("Total time: %f seconds\n", end_time -
start_time);

    // Варіант 3: Паралельна реалізація з двома pragma
for
    printf("\nVariant 3: 4 cores with 2 pragma for\n");
    start_time = omp_get_wtime();
    multiply_matrices_with_timing(MB_MC, MB, MC, 4, 2,
"MB x MC");
    multiply_matrices_with_timing(MD_ME, MD, ME, 4, 2,
"MD x ME");
    add_matrices_with_timing(MA, MB_MC, MD_ME, "MA =
MB_MC + MD_ME");
    end_time = omp_get_wtime();
    printf("Total time: %f seconds\n", end_time -
start_time);

    // Варіант 4: Паралельна реалізація з трьома pragma
for
    printf("\nVariant 4: 4 cores with 3 pragma for\n");
    start_time = omp_get_wtime();
```

```
    multiply_matrices_with_timing(MB_MC, MB, MC, 4, 3,
"MB x MC");
    multiply_matrices_with_timing(MD_ME, MD, ME, 4, 3,
"MD x ME");
    add_matrices_with_timing(MA, MB_MC, MD_ME, "MA =
MB_MC + MD_ME");
    end_time = omp_get_wtime();
    printf("Total time: %f seconds\n", end_time -
start_time);

    // Звільнення пам'яті
    free_matrix(MB);
    free_matrix(MC);
    free_matrix(MD);
    free_matrix(ME);
    free_matrix(MA);
    free_matrix(MB_MC);
    free_matrix(MD_ME);

    printf("\nCalculation complete.\n");
    return 0;
}
```