Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

на тему «Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій»

КвРКІП. 2301147.23.01.24 ПЗ

Виконав: студент 2 курсу, група КІ2м-23-1 _____ Олексій БОНДАР
Підпис Ім'я, прізвище

Керівник д-р. техн. наук, професор _____ Сергій ЛИСЕНКО
Науковий ступінь, вчене звання Підпис Ім'я, прізвище

До захисту допускаю:
Зав. кафедри КІІС, доктор філософії, доцент
Ольга. ПАВЛОВА
29 04 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень МАГІСТР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма ОСВІТНЬО-НАУКОВА ПРОГРАМА «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ ТА ПРОГРАМУВАННЯ»

ЗАТВЕРДЖУЮ

Зав. кафедри Ольга ПАВЛОВА

" 01 " 09 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

БОНДАРУ Олексію Миколайовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій
Керівник проекту (роботи) Сергій ЛИСЕНКО, д.т.н., професор
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2025 №8

2. Строк подання студентом проекту (роботи) на кафедру 01.05.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз відомих методів високопродуктивних обчислень

Модель системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Реалізація системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

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

6. Консультанти розділів кваліфікаційної роботи магістра

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сергій ЛИСЕНКО, професор кафедри КІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІС		

7. Дата видачі завдання «_01_» 09 2024р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) кваліфікаційної роботи магістра	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики КвРМ з керівником	01.09.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.10.2024	виконано
3	Робота над розділом 1 – аналіз відомих моделей, методів за темою; постановка задачі	01.11.2024	виконано
4	Робота над розділом 2 – розробка моделей для вирішення поставленої задачі	01.12.2024	виконано
5	Робота над науковою статтею	01.02.2025	виконано
6	Робота над розділом 3 – розробка методів для вирішення поставленої задачі	15.02.2025	виконано
7	Робота над розділом 4 – проектування та розробка ПЗ для вирішення поставленої задачі, експериментальна частина	01.04.2025	виконано
8	Оформлення пояснювальної записки згідно вимог	18.04.2025	виконано
9	Попередній захист ДРМ	29.04.2025	виконано
10	Захист ДРМ на засіданні ЕК	До 15.05.2025	

Студент _____ Олексій БОНДАР
Підпис Ім'я, прізвище

Керівник роботи _____ Сергій ЛИСЕНКО
Підпис Ім'я, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

Автор роботи: Бондар Олексій Миколайович.

Керівник роботи: Лисенко Сергій Миколайович.

Пояснювальна записка: 86 с., 10 рис., 10 табл., 2 дод., 93 джерел.

QoS, БД, СПЗ, ПЗ, ГРАФ, ДЕКОМПОЗИЦІЯ, ІНДЕНТИФІКАЦІЯ.

Об’єктом дослідження є використання хмарних технологій для забезпечення високопродуктивних обчислень на мобільних пристроях.

Предметом дослідження є методи та системи забезпечення високопродуктивних обчислень на мобільних пристроях за допомогою хмарних технологій.

Метою кваліфікаційної роботи магістра є вдосконалення обчислювальної продуктивності мобільних пристроїв за допомогою хмарних технологій.

Для розв’язання поставлених задач використовувалися методи принципу теорії комп’ютерних мереж і систем, системного аналізу, моделювання, методів аналізу даних, математичної статистики, дискретної математики та інших подібних методів вирішень.

Наукова новизна отриманих результатів:

− набув подальшого розвитку метод використання хмарних технологій для високопродуктивних обчислень на мобільних пристроях дозволяє зменшити повторні обчислення через локалізацію змін у графі залежностей, узгоджено виконувати потоки, ефективно розподіляти навантаження та знижувати витрати на комунікацію й обчислення при збереженні даних;

− набула подальший розвиток інформаційна технологія системи високопродуктивних обчислень на мобільних пристроях із застосуванням хмарних технологій.

На основі проведених досліджень розроблена архітектура і компоненти програмного забезпечення, що дозволяють ефективно розподіляти обчислювальні ресурси між мобільними пристроями та хмарною інфраструктурою.

Практична цінність отриманих результатів полягає у створенні системи високопродуктивних обчислень на мобільних пристроях із застосуванням хмарних технологій.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

КДГЗ – конкурентний динамічний граф залежностей

БД – база даних

СПЗ – системне програмне забезпечення

БПР – блок прийняття рішень

ОС – операційна система

ПЗ – програмне забезпечення

ВСТУП

У сучасному світі інформаційних технологій обсяг даних, які необхідно обробляти, зростає в геометричній прогресії. Для вирішення складних наукових, інженерних та бізнес-завдань традиційні обчислювальні методи часто виявляються недостатньо ефективними [1-3]. Саме тому високопродуктивні обчислення відіграють ключову роль у багатьох сферах, забезпечуючи необхідну швидкість та потужність обробки інформації [4-6].

Ця технологія охоплює паралельні та розподілені системи, що дозволяють одночасно виконувати мільярди операцій, використовуючи ресурси суперкомп'ютерів, кластерів або хмарних платформ [7]. Високопродуктивні обчислення застосовуються в наукових дослідженнях, фінансовому аналізі, моделюванні складних процесів у медицині, фізиці, штучному інтелекті та інших сферах, де швидкість обробки даних визначає успішність рішень [8, 9].

Дослідження цього напряму відкриває перспективи для розвитку нових технологій та методологій, що дозволяють ефективніше використовувати обчислювальні ресурси [10-12]. Саме тому аналіз застосування високопродуктивних обчислень є важливим кроком для розуміння їхньої ролі в сучасних технологічних процесах [13].

У сучасному світі мобільні пристрої стали невід'ємною частиною повсякденного життя, а їхні обчислювальні можливості стрімко зростають. Завдяки потужним процесорам, графічним прискорювачам та спеціалізованим нейронним модулям смартфони та планшети здатні виконувати завдання, які ще кілька років тому були доступні лише для настільних комп'ютерів [14]. Проте, коли йдеться про виконання ресурсоємних операцій, таких як машинне навчання, складні симуляції чи обробка великих масивів даних, можливостей окремого мобільного пристрою все ще недостатньо. Саме тут у гру вступають високопродуктивні обчислення (High-Performance Computing, HPC), які відкривають нові горизонти для мобільних технологій [15].

Застосування HPC на мобільних пристроях може відбуватися через розподілені обчислення, хмарні платформи або гібридні рішення, де смартфон використовується як інтерфейс для потужніших серверів. Наприклад, сучасні мобільні додатки для доповненої реальності, складної обробки зображень або аналізу великих даних нерідко покладаються на віддалені обчислювальні ресурси, забезпечуючи баланс між продуктивністю та енергоефективністю [16-18].

Розвиток 5G-мереж і вдосконалення алгоритмів розподілених обчислень дозволяють ще швидше передавати та обробляти дані, що робить HPC ще більш доступним для мобільних пристроїв [19-20]. У той же час постає питання оптимізації – як ефективно використовувати доступні ресурси, мінімізуючи затримки та енергоспоживання [21]. Це є ключовим викликом для інженерів та дослідників, які прагнуть зробити високопродуктивні обчислення невід'ємною частиною мобільних екосистем.

Застосування високопродуктивних обчислень (HPC) на мобільних пристроях відкриває нові можливості для складних обчислювальних завдань, таких як штучний інтелект, обробка великих даних та моделювання фізичних процесів. Проте, незважаючи на всі переваги, ця сфера стикається з численними викликами та обмеженнями, які гальмують повноцінне впровадження HPC у мобільні екосистеми [22].

Однією з головних проблем є обмеженість обчислювальних ресурсів. На відміну від серверів чи суперкомп'ютерів, мобільні пристрої мають значно меншу потужність процесорів та графічних прискорювачів. Хоча сучасні мобільні чипи оснащені багатоядерними процесорами та спеціалізованими модулями для нейронних обчислень, вони все ще не можуть конкурувати з повноцінними HPC-системами, коли йдеться про масштабні розрахунки [23].

Ще одним серйозним викликом є енергоспоживання. Високопродуктивні обчислення потребують значних обсягів енергії, що створює проблему для мобільних пристроїв, які працюють від акумулятора. Виконання складних обчислень на смартфоні або планшеті може призвести до швидкого розряду батареї, а також до перегріву пристрою, що негативно впливає на його

продуктивність та довговічність. Саме тому існує необхідність у розробці ефективних механізмів енергозбереження, які б дозволили виконувати ресурсомісткі операції без значного впливу на автономність пристрою [24].

Не менш важливим є питання затримок при передачі даних. У випадку розподілених обчислень, коли частина задач передається на сервери або хмарні платформи, швидкість та стабільність з'єднання відіграють ключову роль. Хоча розвиток 5G-мереж суттєво зменшує цей недолік, затримки все ще залишаються проблемою, особливо у регіонах із нестабільним інтернет-з'єднанням. Якщо пристрій залежить від зовнішнього HPC-ресурсу, перебої в мережі можуть зупинити виконання важливих завдань або зробити роботу системи нестабільною [25-27].

Також існує проблема сумісності програмного забезпечення. Багато алгоритмів і додатків, розроблених для високопродуктивних обчислень, орієнтовані на традиційні архітектури серверів та кластерів, що ускладнює їх адаптацію під мобільні пристрої. Оптимізація програмного забезпечення для енергоефективної роботи на мобільних чипах вимагає значних зусиль з боку розробників, що може уповільнювати впровадження HPC-рішень у мобільному середовищі [29-30].

Окрім цього, безпека та конфіденційність даних залишаються серйозною загрозою. Використання хмарних обчислень для підвищення продуктивності мобільних пристроїв означає передачу даних через мережу, що створює ризики перехоплення або несанкціонованого доступу. Захист чутливої інформації та забезпечення надійного зберігання даних у розподілених системах є складним завданням, яке потребує удосконалення криптографічних методів і підходів до аутентифікації [31].

Попри всі ці виклики, дослідження у сфері HPC для мобільних пристроїв активно тривають. Удосконалення мобільних процесорів, розвиток хмарних технологій та нові методи оптимізації енергоспоживання поступово зменшують існуючі обмеження. У майбутньому високопродуктивні обчислення можуть стати

невід’ємною частиною мобільних екосистем, забезпечуючи ще більшу продуктивність і відкриваючи нові можливості для користувачів [32-33].

З розвитком обчислювальних технологій та мобільних пристроїв високопродуктивні обчислення (HPC) поступово виходять за межі традиційних суперкомп’ютерів та серверних кластерів, проникаючи у сферу мобільних рішень [34]. Одним із ключових напрямів цього розвитку є інтеграція HPC із хмарними технологіями, що відкриває перед мобільними пристроями нові можливості у сфері штучного інтелекту, великих даних, наукових розрахунків та інших складних обчислювальних задач [35].

Завдяки хмарним обчисленням мобільні пристрої можуть отримувати доступ до потужних ресурсів у віддалених центрах обробки даних, компенсуючи обмеження своєї апаратної частини [36]. Це особливо актуально для додатків, що потребують інтенсивної обробки інформації, таких як машинне навчання, обробка відео в реальному часі, складне моделювання та візуалізація. Наприклад, сучасні мобільні асистенти на основі штучного інтелекту здатні виконувати базову обробку даних локально, але складніші обчислення, такі як глибокі нейронні мережі, передаються на хмарні сервери, що дозволяє досягати високої швидкодії без навантаження на локальні ресурси пристрою [37].

Однією з ключових переваг хмарно-орієнтованих HPC-рішень є можливість динамічного масштабування ресурсів залежно від потреб користувача. Якщо мобільному додатку потрібно виконати складне обчислення, воно може отримати доступ до відповідних ресурсів у хмарі, використовуючи їх лише на час виконання завдання, що значно підвищує ефективність використання обчислювальних потужностей [38-39]. Це також дозволяє оптимізувати витрати, оскільки користувачі та організації можуть оплачувати лише фактично використані ресурси, замість інвестування в дороге обладнання.

Перспективи розвитку цієї технології тісно пов’язані з удосконаленням мережевих інфраструктур [40]. Зокрема, поширення 5G значно зменшує затримки при передаванні даних між мобільними пристроями та хмарними серверами, забезпечуючи стабільне з'єднання з високою пропускною здатністю. Це відкриває

нові можливості для застосування HPC у таких сферах, як автономні транспортні системи, де обробка величезних потоків даних у реальному часі є критично важливою [41-42].

Окрім того, стрімкий розвиток концепції Edge Computing дає можливість поєднувати локальні обчислення мобільних пристроїв із ресурсами периферійних серверів, які розташовані ближче до кінцевого користувача [43]. Це дозволяє зменшити затримки ще більше та підвищити ефективність мобільних рішень. Наприклад, у сфері медицини мобільні пристрої, підключені до хмарних платформ, можуть виконувати попередній аналіз медичних зображень або біометричних даних пацієнта, а складніший аналіз відправляти на віддалені HPC-сервери [44].

Попри всі переваги, існують також виклики, пов’язані із впровадженням HPC через хмарні сервіси. Одним із них є безпека передавання та зберігання даних [45, 46]. Використання хмарних платформ означає, що конфіденційні дані користувачів можуть передаватися через мережу, що створює потенційні ризики витоку інформації. Для вирішення цієї проблеми активно розробляються нові методи шифрування, анонімізації даних та безпечного зберігання інформації у хмарних сховищах [47].

Інтеграція високопродуктивних обчислень із хмарними технологіями дозволяє мобільним пристроям значно розширити свої можливості та брати участь у виконанні складних обчислювальних задач без необхідності використання дорогого апаратного забезпечення. Це відкриває нові перспективи у різних галузях науки, бізнесу, медицини та технологій, наближаючи мобільні обчислення до рівня традиційних HPC-систем. З подальшим розвитком бездротових мереж, штучного інтелекту та хмарних платформ мобільні пристрої зможуть ще активніше використовувати потужності суперкомп’ютерів, що зробить високопродуктивні обчислення доступнішими та ефективнішими для широкого кола користувачів.

Таким чином постає задача підвищення продуктивності функціонування систем, що використовують хмарні технології, за рахунок застосування мультизадачності.

Метою кваліфікаційної роботи магістра є підвищення продуктивності обчислень на мобільних пристроях із використанням хмарних технологій.

Поставлена мета досягається розв’язанням таких основних завдань:

- проаналізувати відомі методи продуктивності обчислень на мобільних пристроях із використанням хмарних технологій;

- розробити моделі здійснення високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

- розробити метод здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій;

- здійснити дослідження методу здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій.

Об’єктом дослідження є високопродуктивні обчислення на мобільних пристроях із використанням хмарних технологій.

Предметом дослідження є метод та система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

Наукова новизна отриманих результатів:

- розроблено новий метод здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій, уможливлює зменшення обсяг повторного обчислення шляхом локалізації змін у графі залежностей, забезпечує узгоджене виконання потоків, розподіляє обчислювальне навантаження між мобільними пристроями з використанням хмарної інфраструктури, що знижує затрати на комунікацію та обчислення при збереженні даних.

- удосконалено систему для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

Практична значимість отриманих результатів полягає у розробленому системі для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

Для розв’язання поставлених задач використовуються основні положення теорії комп’ютерних мереж та систем, системного аналізу, моделювання, методів

аналізу даних, теорії математичної статистики, теорії дискретної математики, теорії.

За темою кваліфікаційної роботи опубліковано одну публікацію [93] у Збірнику наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2024». (Хмельницький 2024. С. 303-305).

1 АНАЛІЗ ВІДОМИХ МЕТОДІВ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЕНЬ

1.1 Огляд та поняття високопродуктивних обчислень

Високопродуктивні обчислення (High-Performance Computing, HPC) є основою сучасних наукових досліджень, інженерних розробок та аналізу великих даних, забезпечуючи можливість виконання надскладних обчислень у найкоротший термін [48]. Ця технологія виникла як відповідь на потребу людства у швидкому та ефективному розв’язанні завдань, що виходять за межі можливостей традиційних комп’ютерів.

Суть високопродуктивних обчислень полягає у використанні потужних апаратних та програмних рішень, які дозволяють здійснювати паралельну обробку великих обсягів даних. У традиційних комп’ютерах виконання задач відбувається послідовно, що накладає обмеження на швидкість обчислень [49].

Натомість HPC використовує паралельні архітектури, де одночасно працюють сотні, тисячі або навіть мільйони процесорних ядер, що дозволяє суттєво прискорити розрахунки [50].

Ключовим компонентом високопродуктивних обчислень є суперкомп’ютерні системи – складні системи, що складаються з великої кількості взаємопов’язаних процесорів, графічних прискорювачів та спеціалізованих модулів, які можуть виконувати мільярди операцій за секунду.

Однак HPC не обмежується лише суперкомп’ютерами: концепція розподілених обчислень дозволяє об’єднувати тисячі звичайних серверів у єдину систему, яка може працювати як єдиний організм.

Такий підхід використовується у хмарних технологіях та грід-обчисленнях, коли ресурси кількох віддалених центрів обробки даних поєднуються для розв’язання надскладних задач [50, 51].

Однією з ключових особливостей високопродуктивних обчислень є їхній вплив на наукові дослідження та промисловість [52-54].

Наприклад, у медицині HPC допомагає створювати точні моделі білкових структур, що прискорює розробку нових ліків.

У кліматології високопродуктивні обчислення дозволяють здійснювати моделювання змін клімату, прогнозуючи природні катастрофи з високою точністю. Інженери та дизайнери використовують HPC для моделювання складних фізичних процесів, наприклад, поведінки аеродинамічних конструкцій літаків або міцності матеріалів у будівництві [55].

Програмне забезпечення, що використовується у високопродуктивних обчисленнях, також відрізняється від традиційних комп'ютерних рішень. Операційні системи, алгоритми та бібліотеки оптимізовані для роботи у багатопотоковому середовищі, а програмісти застосовують спеціальні методи розпаралелювання коду, щоб максимально ефективно використовувати доступні ресурси.

Використання графічних процесорів (GPU) та спеціалізованих прискорювачів, таких як Tensor Processing Units (TPU), дозволяє ще більше збільшити продуктивність, особливо у завданнях, пов'язаних із машинним навчанням та аналізом великих масивів даних [56-58].

Розвиток високопродуктивних обчислень стає дедалі важливішим у контексті штучного інтелекту, оскільки сучасні моделі нейронних мереж потребують обробки терабайтів даних у реальному часі.

Без потужних HPC-систем тренування таких моделей зайняло б роки або десятиліття, тоді як сучасні суперкомп'ютери можуть виконати ці завдання за лічені дні або навіть години [59].

Завдяки стрімкому розвитку технологій високопродуктивні обчислення поступово стають доступнішими.

Якщо раніше HPC асоціювалися виключно із суперкомп'ютерами в урядових лабораторіях та дослідницьких інститутах, то сьогодні навіть мобільні пристрої можуть використовувати їхні можливості через хмарні сервіси.

Це означає, що майбутнє HPC не обмежується лише великими науковими проєктами. Вони стають частиною повсякденного життя, відкриваючи нові перспективи для технологій та суспільства в цілому [60-62].

1.2 Використання високопродуктивних обчислень на мобільних пристроях

Сучасні мобільні пристрої вже давно перестали бути лише засобом зв’язку чи інструментом для виконання базових завдань.

Завдяки значному розвитку апаратного забезпечення та програмних оптимізацій вони перетворилися на повноцінні обчислювальні платформи, здатні виконувати складні завдання, які раніше були доступні лише для серверів та суперкомп’ютерів [63].

Одним із ключових напрямів цієї еволюції стало використання високопродуктивних обчислень (HPC) у мобільному середовищі, що дозволяє смартфонам, планшетам та іншим портативним пристроям вирішувати завдання, пов’язані з аналізом великих обсягів даних, машинним навчанням, обробкою зображень та відео, а також симуляціями фізичних процесів [64].

Фундаментальною зміною, що сприяла поширенню HPC у мобільних пристроях, стало покращення їхніх процесорних можливостей.

Сучасні мобільні чипи містять багатоядерні центральні процесори (CPU), графічні процесори (GPU) та навіть нейронні обчислювальні блоки (NPU), які спеціально оптимізовані для роботи з відомими використовуваними алгоритмами штучного інтелекту.

Наприклад, топові смартфони здатні виконувати мільярди операцій за секунду, що дозволяє їм обробляти складні задачі прямо на пристрої, без необхідності підключення до серверів.

Це особливо важливо для таких застосувань, як розпізнавання облич, автоматичний переклад у реальному часі, покращена фотографія з використанням штучного інтелекту та аналіз медичних даних [65].

Попри значні досягнення в продуктивності мобільного заліза, існують обмеження, пов’язані з енергоспоживанням та тепловиділенням. Мобільні пристрої не можуть дозволити собі витрачати стільки ж енергії, як традиційні HPC-системи, адже це призведе до швидкого розряду акумулятора та перегріву.

Саме тому мобільні високопродуктивні обчислення часто поєднуються з хмарними технологіями, які дозволяють передавати складні розрахунки на віддалені сервери.

Завдяки цьому користувач отримує потужність суперкомп’ютера у своєму кишеньковому пристрої, при цьому споживання енергії залишається мінімальним [66-68].

Хмарні сервіси, такі як Google Cloud, AWS та Microsoft Azure, забезпечують мобільним пристроям доступ до ресурсів, необхідних для обробки великих обсягів даних та виконання обчислювально інтенсивних задач [69].

Наприклад, у мобільних іграх зі складною графікою використовується стрімінг зображень із серверів, що дозволяє запускати високоякісні ігри безпосередньо на смартфонах, навіть якщо вони не мають достатньо потужного обладнання.

Аналогічно, у сфері доповненої та віртуальної реальності мобільні пристрої можуть отримувати складно обчислені 3D-моделі з хмари, значно покращуючи якість взаємодії з користувачем [70].

Ще однією важливою сферою застосування мобільних HPC є штучний інтелект. Завдяки інтеграції спеціалізованих модулів (NPU) у сучасні процесори мобільні пристрої можуть виконувати нейронні обчислення локально, що прискорює роботу додатків, які потребують глибокого навчання.

Наприклад, камери смартфонів використовують ШІ для покращення фотографій у реальному часі, застосовуючи складні алгоритми аналізу сцени, шумозаглушення та корекції освітлення.

Також розумні голосові асистенти, такі як Google Assistant чи Siri, навчаються безпосередньо на пристрої, що дозволяє їм працювати швидше та зберігати конфіденційність даних користувача [71].

У сфері медицини високопродуктивні обчислення на мобільних пристроях дозволяють аналізувати біометричні показники в реальному часі.

Сучасні розумні годинники та фітнес-трекери оснащені сенсорами, що вимірюють серцевий ритм, рівень кисню в крові та інші параметри, використовуючи потужні алгоритми для прогнозування можливих проблем зі здоров'ям.

Завдяки HPC такі пристрої можуть виявляти ознаки аритмії або апное сну без необхідності звернення до лікаря, що відкриває нові можливості у сфері персоналізованої медицини [72].

Інженерія та наука також виграють від поєднання мобільних пристроїв із HPC.

У геології, наприклад, мобільні додатки можуть аналізувати дані з датчиків землетрусів та прогнозувати можливі підземні поштовхи, використовуючи потужні алгоритми, що працюють у хмарі.

В астрономії смартфони можуть брати участь у розподілених обчисленнях, допомагаючи аналізувати космічні знімки для пошуку нових екзопланет або чорних дір [73].

Незважаючи на всі досягнення, мобільні високопродуктивні обчислення стикаються з певними викликами.

Наприклад, швидкість передачі даних між пристроєм і сервером може бути критичним фактором для продуктивності, тому розвиток 5G-мереж є важливим для подальшого прогресу цієї технології.

Крім того, безпека даних залишається значною проблемою, оскільки передача конфіденційної інформації до хмарних платформ потребує надійного шифрування та захисту від кібератак [74].

Загалом, високопродуктивні обчислення на мобільних пристроях стають дедалі важливішими, змінюючи уявлення про можливості портативної електроніки.

Поєднання потужного апаратного забезпечення, розподілених обчислень та хмарних сервісів дозволяє мобільним пристроям виконувати завдання, які ще

кілька років тому вимагали наявності суперкомп’ютерів [75]. Цей напрямок продовжує розвиватися, відкриваючи нові перспективи у сферах штучного інтелекту, медицини, науки, ігор та інженерії, роблячи світ ще більш технологічним та взаємопов’язаним [76].

1.3 Хмарні технології як платформа для високопродуктивних обчислень на мобільних пристроях

Мобільні пристрої, такі як смартфони та планшети, стають дедалі потужнішими, проте їхні обчислювальні можливості все ще обмежені через компактні розміри, енергоспоживання та тепловиділення.

Одним із найперспективніших рішень для подолання цих обмежень є використання хмарних технологій як платформи для високопродуктивних обчислень.

Завдяки цьому підходу мобільні пристрої можуть отримувати доступ до практично необмежених ресурсів хмарних центрів обробки даних, що відкриває нові можливості для складних обчислювальних завдань [77].

Суть інтеграції мобільних пристроїв із хмарними HPC-платформами полягає у розподілі обчислювальних навантажень.

Мобільний пристрій виконує функцію клієнта, який збирає дані, здійснює базову обробку та відправляє їх до хмарного сервера. У хмарі потужні кластери серверів проводять складні розрахунки, після чого результати повертаються назад на мобільний пристрій.

Цей підхід дозволяє виконувати навіть найскладніші завдання, такі як обробка відео у реальному часі, моделювання фізичних процесів або аналіз великих даних без значного навантаження на локальний пристрій [78].

Однією з ключових сфер застосування такого підходу є машинне навчання. Навіть найсучасніші смартфони мають обмежені можливості у тренуванні нейронних мереж, адже цей процес вимагає величезних обсягів обчислень.

Завдяки хмарним сервісам, таким як Google Cloud AI, Microsoft Azure Machine Learning та AWS SageMaker, мобільні додатки можуть використовувати потужні графічні та тензорні процесори у віддалених дата-центрах. Це означає, що штучний інтелект може навчатися на величезних наборах даних та одразу передавати оновлені моделі на мобільні пристрої [79].

Ще однією важливою перевагою є можливість обробки зображень та відео на рівні, який неможливо реалізувати на мобільному процесорі.

Наприклад, у додатках для редагування фотографій та відео високої якості користувачі можуть застосовувати складні ефекти та алгоритми покращення зображення, які зазвичай потребують великих обчислювальних ресурсів.

Мобільний пристрій лише передає вихідні файли до хмари, де сервери виконують усі розрахунки, а потім надсилають назад готовий результат з даними [80-82].

Хмарні HPC-рішення також революціонізують мобільний геймінг. Хмарні ігрові платформи, такі як NVIDIA GeForce Now, Google Stadia або Xbox Cloud Gaming, дозволяють запускати графічно насичені ігри безпосередньо на смартфоні або планшеті, використовуючи лише інтернет-з’єднання.

Гра фактично виконується на віддаленому сервері, а мобільний пристрій отримує лише потокове зображення, що дає змогу запускати навіть найвимогливіші проєкти на слабкому апаратному забезпеченні [83].

У медицині хмарні високопродуктивні обчислення дозволяють мобільним пристроям швидко аналізувати складні медичні знімки, такі як МРТ або КТ, безпосередньо в лікарняних умовах.

Лікар може використовувати планшет для сканування зображень, після чого штучний інтелект у хмарі миттєво аналізує дані та пропонує можливі діагнози. Це значно пришвидшує процес прийняття рішень та покращує якість медичного обслуговування [84].

Однак ефективність такого підходу залежить від швидкості та стабільності інтернет-з’єднання.

З появою 5G мобільні пристрої отримують можливість обмінюватися даними з хмарними сервісами майже миттєво, що значно покращує інтерактивність обчислювальних процесів.

Мобільні додатки, що використовують HPC у хмарі, можуть працювати в режимі реального часу, забезпечуючи плавний користувацький досвід навіть у найскладніших сценаріях [85].

Безпека даних також відіграє важливу роль у використанні хмарних високопродуктивних обчислень.

Передача конфіденційної інформації, наприклад, у фінансових або медичних додатках, потребує надійного шифрування та захисту від кібератак.

Сучасні хмарні сервіси пропонують багаторівневі системи безпеки, включаючи шифрування даних у процесі передачі та зберігання, а також технології аутентифікації та контролю доступу.

Перспективи застосування хмарних технологій для високопродуктивних обчислень у мобільних пристроях продовжують зростати.

З розвитком квантових обчислень, покращенням алгоритмів розподіленої обробки та впровадженням більш швидкісних бездротових мереж ця технологія стане ще більш доступною та потужною.

У майбутньому користувачі зможуть отримати доступ до ресурсів, які раніше були ексклюзивними для наукових лабораторій та суперкомп’ютерів, прямо зі своїх мобільних пристроїв [86].

Зрештою, хмарні HPC-рішення для мобільних платформ змінюють спосіб взаємодії з технологіями, відкриваючи нові горизонти в науці, медицині, розвагах та бізнесі.

Це не лише підвищує ефективність обчислень, але й робить складні технології доступними кожному користувачеві, незалежно від апаратних обмежень їхнього пристрою.

1.4 Дослідження методів організації високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

В останні роки високопродуктивні обчислення (HPC) з використанням хмарних технологій стали важливою темою наукових досліджень.

У статті [87] розглядаються останні досягнення в квантових хмарних обчисленнях, включаючи різні моделі хмар, платформи та технології. Обговорюються проблеми управління ресурсами, безпеки та конфіденційності, а також пропонуються напрямки майбутніх досліджень.

Квантові хмарні обчислення — це нова парадигма, яка дозволяє розгортати та виконувати квантові програми на квантових обчислювальних ресурсах без потреби в спеціалізованих середовищах для розміщення та роботи з фізичними квантовими комп'ютерами.

Цей підхід дозволяє користувачам отримувати доступ до можливостей квантових обчислень через хмарні платформи, полегшуючи дослідження та розробку квантових алгоритмів і програм. Стаття містить вичерпний огляд останніх досягнень квантових хмарних обчислень. У роботі визначено відкриті проблеми та запропоновано майбутні напрямки у цій новій галузі. Автори обговорюють різні хмарні моделі та платформи, які були розроблені для підтримки квантових обчислень. Ці платформи пропонують користувачам можливість виконувати квантові схеми та віддалено отримувати результати обчислень, тим самим зменшуючи потребу у спеціальному фізичному квантовому обладнанні. Ця модель є особливо корисною в сучасну епоху квантових пристроїв проміжного масштабу (NISQ), які, незважаючи на свої обмеження, забезпечують багатообіцяючу платформу для дослідження квантових алгоритмів і додатків. У статті також розглядаються різні аспекти квантових хмарних обчислень, включаючи управління ресурсами, квантові безсерверні обчислення, а також питання безпеки та конфіденційності.

Ефективне управління ресурсами має вирішальне значення для оптимізації продуктивності та ефективності квантових хмарних платформ. Квантові

безсерверні обчислення, які дозволяють користувачам запускати квантові додатки без керування основною інфраструктурою, визначені як потенційна область для розвитку. Крім того, автори підкреслюють важливість вирішення питань безпеки та конфіденційності для забезпечення безпечного та конфіденційного виконання квантових програм у хмарних середовищах. На закінчення в статті розглядаються відкриті проблеми квантових хмарних обчислень і пропонуються майбутні напрямки досліджень і розробок. Він підкреслює необхідність постійних досліджень для подолання поточних проблем і повної реалізації потенціалу квантових хмарних обчислень у різних програмах. Автори припускають, що вирішення цих проблем відкриє нові можливості для використання квантових обчислювальних ресурсів через хмарні платформи, тим самим просуваючи сферу та її практичні застосування.

Стаття [88] аналізує зміни в галузі HPC, зокрема вплив хмарних технологій та штучного інтелекту. Обговорюються виклики та можливості, пов'язані з інтеграцією HPC у хмарні середовища, а також необхідність переосмислення традиційних підходів до обчислень. Стаття розглядає розвиток високопродуктивних обчислень (HPC) та їх значення для наукових досліджень. Автори підкреслюють значний перехід від традиційних суперкомп’ютерних систем до більш диверсифікованої глобальної екосистеми HPC. Ця трансформація зумовлена декількома факторами, зокрема зростаючими витратами появою індивідуальних високопродуктивних систем, а також глобальним дефіцитом напівпровідників, який вплинув на всі країни.

Ключова зміна, визначена в документі, полягає в значних інвестиціях великих постачальників хмарних послуг у великомасштабні обчислювальні інфраструктури. Ці хмарні системи, які часто використовують розроблені на замовлення напівпровідники, не тільки перевершують традиційні високопродуктивні системи за масштабом, але й досягають прориву в таких сферах, як штучний інтелект, ігри та комп’ютерне зір. Ця подія змінює підхід до наукових обчислень, вказуючи на необхідність зміни парадигми в тому, як обчислювальні ресурси використовуються та інтегруються в дослідницькі робочі

процеси. Щоб подолати ці проблеми та використати нові можливості, автори виступають за комплексне переосмислення дизайну та розгортання систем HPC. Вони пропонують охопити наскрізне спільне проектування, яке передбачає тісну співпрацю між розробкою апаратного та програмного забезпечення з самого початку. Крім того, вони підкреслюють важливість індивідуальних конфігурацій апаратного забезпечення та упаковки для задоволення конкретних обчислювальних вимог. Автори також закликають до широкомасштабного прототипування, що нагадує практику тридцятирічної давнини, для тестування та вдосконалення нових технологій перед повномасштабним впровадженням. Крім того, вони наголошують на необхідності формування спільних партнерських відносин із провідними компаніями, що займаються обчислювальною екосистемою, у тому числі в секторах смартфонів і хмарних обчислень, щоб сприяти інноваціям і гарантувати, що системи високопродуктивних вимірів відповідають постійним потребам наукових досліджень. Підводячи підсумок, у статті підкреслюється критична потреба спільноти HPC адаптуватися до швидко мінливого технологічного та економічного середовища. Застосовуючи більш інтегрований та спільний підхід до проектування та розробки системи, автори вважають, що наступне покоління систем HPC може бути більш ефективним, економічно ефективним і краще відповідати складним обчислювальним завданням сучасних наукових досліджень.

У статті [89] розглядаються принципи проєктування та виклики при створенні масштабованих систем та програмних архітектур для HPC на хмарних платформах. Особлива увага приділяється паралелізму, ієрархії пам'яті, комунікаційним затримкам та відмовостійкості. Високопродуктивні обчислення (HPC) необхідні для вирішення складних обчислювальних проблем у різних областях. Оскільки масштаб і складність додатків HPC продовжують зростати, потреба в масштабованих системах і архітектурах програмного забезпечення стає першорядною. У статті надається вичерпний огляд архітектури для HPC на місці, зосереджуючись як на апаратних, так і на програмних аспектах, а також детально описуються проблеми, пов’язані зі створенням кластера HPC на місці. Робота

досліджує принципи проектування, виклики та нові тенденції у створенні масштабованих систем HPC і програмного забезпечення, вирішуючи такі проблеми, як паралелізм, ієрархія пам'яті, накладні витрати на зв'язок і відмовостійкість на різних хмарних платформах. Узагальнюючи результати досліджень і технологічні досягнення, ця стаття має на меті надати розуміння масштабованих рішень для задоволення зростаючих вимог до додатків HPC у хмарі.

Стаття [90] пропонує класифікацію та огляд HPC у хмарі, обговорюючи переваги та недоліки використання хмарних ресурсів для наукових та бізнес-додатків. Стаття досліджує інтеграцію високопродуктивних обчислень (HPC) із хмарними обчисленнями для вдосконалення наукових і бізнес-додатків. Автори обговорюють еволюцію HPC-хмар як життєздатної альтернативи традиційним локальним кластерам, підкреслюючи їхній потенціал для більш ефективного виконання ресурсомістких програм. У роботі представлено комплексну класифікацію хмарних досліджень HPC. В роботі подано оцінку фінансових наслідків міграції ресурсомістких програм із локальних середовищ на публічні хмарні платформи.

Досліджено поєднання локальних і хмарних ресурсів для оптимізації продуктивності та економічної ефективності. Такий підхід дозволяє виконувати стабільні робочі навантаження локально, тоді як пікові навантаження можуть використовувати хмарні ресурси за принципом оплати за використання. Також в статті описано процес оптимізації продуктивності систем, пов'язаних із отриманням оптимальної продуктивності з різноманітних і часто невідомих основних хмарних платформ, визначено основні служби, які полегшують використання хмар HPC, роблячи їх більш доступними та зручними для різних програм.

Також представлено обговорення розробки відповідних стратегій ціноутворення та контрактних угод, які обслуговують як малих, так і великих користувачів, забезпечуючи стабільність хмарних послуг HPC. Автори також пропонують бачення майбутнього HPC-хмар, підкреслюючи швидке зростання

нових додатків, що керуються великими даними та штучним інтелектом. Вони наголошують на необхідності продовження досліджень для вирішення виявлених проблем, що може призвести до значних успіхів як у бізнесі, так і в науковій сферах.

В статті підкреслюється трансформаційний потенціал інтеграції HPC із хмарними обчисленнями, пропонується структурований огляд поточних дослідницьких зусиль і окреслюються критичні проблеми, які необхідно вирішити, щоб повністю реалізувати переваги HPC-хмар у різних програмах.

У статті [91] аналізується прогрес хмарних обчислень у досягненні продуктивності, порівняно з традиційними HPC-системами. Розглядаються фактори, що впливають на продуктивність, та пропонуються рекомендації щодо оптимізації хмарних ресурсів для наукових застосувань. У статті автори досліджують, чи можуть інфраструктури хмарних обчислень зрівнятися з продуктивністю традиційних систем високопродуктивних обчислень (HPC) для широкого спектру наукових застосувань. Це питання є важливим для майбутнього проектування системи та для демократизації доступу до високопродуктивних обчислювальних ресурсів. Автори використовують багаторівневий підхід для оцінки розриву в продуктивності між HPC і хмарними обчисленнями. Їх методологія включає апаратні та системні мікротестування, а також проксі-сервери додатків користувача. Отримані дані показують, що сучасні хмарні обчислювальні платформи високого класу можуть забезпечити продуктивність, конкурентоспроможну HPC, не тільки для додатків, що потребують інтенсивних обчислень, але також і для додатків, що потребують інтенсивного використання пам’яті та зв’язку, принаймні в скромних масштабах. Це покращення пояснюється прогресом у високошвидкісних системах пам’яті, з’єднаннях і спеціальному пакетному плануванні, доступному на деяких хмарних платформах. Дослідження підкреслює, що хоча хмарні обчислення досягли значного прогресу в продуктивності, певні фактори, такі як моделі витрат, час очікування роботи, наявність попередньо встановленого програмного забезпечення та інші аспекти зручності використання, також відіграють вирішальну роль в оцінці

конкурентоспроможності хмарних платформ для наукових обчислень. Однак ці фактори виходять за рамки цього конкретного дослідження. Підсумовуючи, у дослідженні показано, що хмарні обчислення досягли значного прогресу в усуненні розриву в продуктивності з традиційними високопродуктивними системами, що робить їх життєздатним варіантом для більш широкого кола наукових застосувань. Ця розробка має потенціал для демократизації доступу до високопродуктивних обчислювальних ресурсів, що принесе користь науковому співтовариству в цілому.

У статті [92] розглядається потенціал хмарних обчислень для наукових обчислень, зокрема для обробки великих обсягів даних. Обговорюються переваги та обмеження використання хмарних ресурсів у наукових дослідженнях.

1.5 Висновки до першого розділу

У розділі розглянуто високопродуктивні обчислення (HPC) та їхнє застосування на мобільних пристроях, особливо у поєднанні з хмарними технологіями.

Аналіз високопродуктивних обчислень показав, що HPC використовується для складних обчислень у науці, медицині, фінансах, інженерії та інших сферах. Традиційно HPC застосовується у суперкомп’ютерах, однак сучасні тенденції орієнтовані на мобільні пристрої та хмарні сервіси.

Впровадження HPC у мобільні пристрої дозволяє виконувати складні задачі, такі як машинне навчання та моделювання. Мобільні пристрої використовують багатоядерні CPU, GPU та спеціалізовані нейронні модулі.

Мобільні пристрої можуть використовувати хмарні сервіси для складних обчислень, що допомагає мінімізувати енергоспоживання та зменшити затримки.

Використання хмарних технологій дозволяє масштабувати систему без значних витрат на апаратне забезпечення.

На основі виконаного аналізу було виявлено основні недоліки відомих рішень, зокрема:

1. Обмежена обчислювальна потужність мобільних пристроїв, тобто мобільні пристрої не можуть конкурувати з традиційними HPC-системами.

2. Виконання інтенсивних обчислень на мобільному пристрої швидко розряджає акумулятор та викликає перегрів.

3. Використання хмарних платформ та розподілених обчислень може призводити до затримок, особливо у регіонах із нестабільним інтернет-з’єднанням.

4. Алгоритми HPC часто розробляються для серверних платформ, що ускладнює їхню адаптацію для мобільних пристроїв.

5. Використання хмарних сервісів потребує передачі чутливих даних через мережу, що створює ризики витоку інформації.

6. Використання потужних хмарних ресурсів може бути дорогим, особливо для індивідуальних користувачів або малих підприємств.

Ці недоліки вказують на необхідність подальших досліджень у напрямку розроблення систем для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

1.6 Постановка задачі

Для вирішення поставленої мети підвищення продуктивності обчислень на мобільних пристроях із використанням хмарних технологій необхідно розв’язати такі основні завдання:

- формалізувати математичну модель функціонування системи з вивченням вхідних параметрів, зазначенням розподілу задач між пристроєм і хмарою, часу виконання задачі, енергоспоживання, ресурсних обмежень, обмеження якості обслуговування (QoS), врахування накладних витрат на безпеку, задання цільової функції оптимізації;

- формалізувати компоненти системи у вигляді множин та функцій взаємодії;

- розробити метод здійснення високопродуктивних обчислень, зокрема описати процес декомпозиції на підобчислення, побудову конкурентного динамічного графа залежностей, поширення змін і поступове повторне виконання;

- описати застосування методу високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій;

- реалізувати системне програмне забезпечення системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій;

- здійснити дослідження методу здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій.

2 МОДЕЛЬ СИСТЕМИ ДЛЯ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЕНЬ НА МОБІЛЬНИХ ПРИСТРОЯХ ІЗ ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

2.1 Формалізована математичну модель системи

Для побудови формалізованого опису системи розглянемо перелік основних компонентів такої системи.

Система включає множину мобільних пристроїв-клієнтів (Mobile Client Device), серед яких: смартфон, планшет або інший пристрій із доступом до комп'ютерної мережі.

Мобільні пристрої відповідають за збір вхідних даних, попередню обробку та ініціацію обчислювальних запитів до хмари.

Також вони можуть використовувати вбудовані сенсори, GPU, або CPU для локальної обробки.

У моделі системи високопродуктивних обчислень на мобільних пристроях з використанням хмарних технологій ключовим компонентом є множина мобільних пристроїв-клієнтів, яку позначимо як $\mathcal{M}$.

$$\mathcal{M} = \{m_1, m_2, \dots m_k\}, \tag{2.1}$$

де кожен елемент $m_i \in \mathcal{M}$ - це індивідуальний мобільний пристрій, який може мати одну з наступних форм:

$$m_i \in \mathcal{M}_{type}, \mathcal{M}_{type} = \{\text{смартфон}, \text{планшет}, \text{інший мереживий пристрій}\}$$

Кожен пристрій m_i характеризується множиною локальних апаратних ресурсів:

$$R_{m_i} = \{r_{GPU}, r_{GPU}, r_{Sensors}, r_{RAM}\}, \tag{2.2}$$

де:

- r_{GPU} — центральний процесор пристрою,

- r_{GPU} — графічний процесор (за наявності),

- $r_{Sensors}$ — вбудовані сенсори (GPS, камери, IMU тощо),

- r_{RAM} — оперативна пам’ять.

Кожен мобільний пристрій виконує наступні функції:

- Збір вхідних даних. Формально, нехай $\mathcal{I}_i$ - множина вхідних даних, отриманих пристроєм m_i через сенсори або інтерфейс користувача.

- Попередня обробка здійснюється:

$$\mathcal{P}_i: \mathcal{I}_i \to \mathcal{I}_i', \tag{2.3}$$

де $\mathcal{P}_i$ - функція локальної обробки, яка трансформує вхідні дані до форми, придатної для подальшого обчислення або передачі.

- Ініціація обчислювального запиту:

$$\mathcal{R}_i: \mathcal{I}_i' \to \mathcal{T}$$

де $\mathcal{R}_i$ - функція ініціації задачі $t \in \mathcal{T}$, що створюється на основі попередньо оброблених даних.

Таким чином, для кожного $m_i \in \mathcal{M}$, формально визначається структура:

$$m_i = \langle \mathcal{I}_i, \mathcal{P}_i, \mathcal{R}_i, R_{m_i} \rangle \tag{2.4}$$

де:

- $\mathcal{I}_i$ — потік вхідних даних;

- $\mathcal{P}_i$ — модуль попередньої обробки;

- $\mathcal{R}_i$ — модуль генерації обчислювального запиту;

- R_{m_i} — апаратні ресурси пристрою.

Система включає мобільну обчислювальну платформу (Mobile HPC Middleware / Framework), а саме програмне забезпечення, що виконує розподіл задач між мобільним пристроєм і хмарою.

Обчислювальна платформа може включати підтримку iThreads, OpenCL, CUDA (на підтримуваних пристроях) або власні засоби обробки задач.

Мобільна обчислювальна платформа, яка виступає програмним посередником між мобільним пристроєм та хмарною інфраструктурою, моделюється як множина існуючих програмних середовищ високопродуктивних обчислень:

$$\mathcal{F} = \{f_1, f_2, \dots, f_n\}, \tag{2.5}$$

де кожен елемент $f_j \in \mathcal{F}\}$ - це програмний модуль або фреймворк, який забезпечує:

- розподіл задач $t \in \mathcal{T}$ між пристроєм $m_i \in \mathcal{M}$ та хмарною платформою $c_k \in \mathcal{C}$,

- використання паралельних обчислювальних засобів локально або через мережу.

Для кожного $f_j \in \mathcal{F}$ визначається множина підтримуваних технологій:

$$T_{f_j} = \{iThreads, OpenCL, CUDA, CustomRuntime\},$$

де:

- iThreads - програмна модель інкрементального багатопотокового виконання,

- OpenCL - кросплатформенна паралельна обчислювальна платформа,

- CUDA - фреймворк для паралельних обчислень на GPU (від NVIDIA),

- CustomRuntime - власні специфічні засоби обробки задач, розроблені під певну платформу.

Кожен фреймворк $f_j \in \mathcal{F}$ реалізує функцію розподілу задач:

$$\mathcal{D}_{f_j}: \mathcal{T} \times \mathcal{M} \to \mathcal{M} \cup \mathcal{C}, \tag{2.6}$$

де:

- $\mathcal{D}_{f_j}(t, m_i) = x$ визначає, чи задача t має виконуватись на локальному пристрої m_i, чи передаватись на обчислення у хмару $x \in \mathcal{C}$.

Кожен компонент обчислювальної платформи має наступну формальну структуру:

$$f_j = \langle T_{f_j}, \mathcal{D}_{f_j} \rangle, \tag{2.7}$$

де:

- T_{f_j} — підтримувані обчислювальні середовища,

- $\mathcal{D}_{f_j}$ — функція прийняття рішень про розміщення задач.

Система також містить модуль адаптивного планування задач, який аналізує тип задачі, ресурсну завантаженість, мережеві умови, приймає рішення про виконання задачі локально або на хмарній інфраструктурі, а також забезпечує баланс між енергоспоживанням, часом відгуку та точністю обчислень.

Модуль адаптивного планування задач виконує ключову роль у прийнятті рішень щодо ефективного розподілу обчислювальних задач між локальними пристроями та хмарною інфраструктурою.

У математичній моделі він описується як множина адаптивних планувальників:

$$\mathcal{S} = \{s_1, s_2, \ldots, s_p\}, \tag{2.8}$$

де кожен елемент $s_k \in \mathcal{S}$ — це інстанція модуля, який працює в контексті конкретного мобільного пристрою $m_i \in \mathcal{M}$ або у межах хмарної платформи.

Кожен планувальник s_k здійснює аналіз таких множин характеристик.

Нехай тип задачі представимо як:

$$\mathcal{T}_{type} = \{GPU - bound, GPU - bound, Data - intensive, Real - time, \dots\}$$

Стан ресурсів мобільного пристрою представимо як:

$$\mathcal{R}_{m_i}^{\text{стан}} = \{GPU_{usage}, RAM_{free}, Battery_{level}, \dots\}$$

Мережеві умови представимо як:

$$\mathcal{N}_{m_i} = \{Latency, Bandwidth, PacketLoss, \dots\}$$

Планувальник реалізує функцію адаптивного планування:

$$\mathcal{A}_{s_k}: \mathcal{T} \times \mathcal{R}_{m_i}^{\text{стан}} \times \mathcal{N}_{m_i} \to \mathcal{M} \cup \mathcal{C}$$

де:

- $\mathcal{A}_{s_k}(t,\cdot,\cdot) = x$ — це вибір середовища виконання задачі t (локальний пристрій або хмара), з урахуванням поточного стану системи.

Кожен $s_k \in \mathcal{S}$ виконує планування із забезпеченням балансу між множинами метрик:

$$\mathcal{Q} = \{q_1, q_2, q_3\},$$

де:

q_1 – енергоспоживання;

q_2 - час відгуку;

q_3 - точність обчислень.

Це реалізується шляхом багатокритеріальної оптимізації:

$$\min_{x \in \mathcal{M} \cup \mathcal{C}} [w_1 \cdot q_1(x) + w_2 \cdot q_2(x) + w_3 \cdot q_3(x)]$$

де w_i — вагові коефіцієнти, що задають пріоритети для критеріїв.

Формально кожен елемент модуля можна подати у вигляді:

$$s_k = \langle \mathcal{A}_{s_k}, \mathcal{T}_{type}, \mathcal{R}_{m_i}^{\text{стан}}, \mathcal{N}_{m_i}, \mathcal{Q} \rangle$$

Система включає мережеву інфраструктуру, яка включає канали зв'язку між мобільним пристроєм і хмарною платформою (Wi-Fi, 4G/5G, LTE, тощо).

Мережеву інфраструктура забезпечує QoS, шифрування та мінімізацію затримок при передаванні даних.

Мережева інфраструктура виконує функції забезпечення якісного обміну даними між мобільними пристроями $m_i \in \mathcal{M}$ та хмарними ресурсами $c_j \in \mathcal{C}$, з урахуванням характеристик мережі, захищеності передавання, рівня QoS і затримок.

Опишемо модуль як множину мережевих каналів:

$$\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_r\}, \tag{2.9}$$

де кожен канал $\ell_r \in \mathcal{L}$ є логічним або фізичним каналом зв'язку між елементами $m_i \in \mathcal{M}$ і $c_j \in \mathcal{C}$, що характеризується множиною параметрів.

Кожен канал ℓ_r описується множиною характеристик як:

$$\mathcal{P}_{\ell_r} = \{L, Ba, P, J, E, Q.\} \tag{2.10}$$

де:

- L — середня затримка (мс),
- B — пропускна здатність (Мбіт/с),
- P — втрати пакетів (%),
- J — флуктуації затримки (мс),
- E — рівень шифрування (наприклад, AES-256),
- Q — клас сервісу (наприклад, Best Effort, Assured Forwarding).

Модуль реалізує функцію оцінки мережевої якості для кожного каналу:

$$\mathcal{F}_{\ell_r} : \mathcal{P}_{\ell_r} \to \mathbb{R}, \tag{2.11}$$

Ця функція оцінює якість каналу, наприклад, як агреговану метрику:

$$\mathcal{F}_{\ell_r} = \alpha_1 \cdot Latency^{-1} + \alpha_2 \cdot Bandwidth - \alpha_3 \cdot PacketLoss - \alpha_4 \cdot Jitter$$

де α_i — вагові коефіцієнти, що визначають важливість відповідної характеристики в контексті задачі.

Функція маршрутизації трафіку або вибору каналу визначається як:

$$\mathcal{R}_{net} : \mathcal{T} \times \mathcal{M} \times \mathcal{C} \to \mathcal{L}, \tag{2.12}$$

де $\mathcal{R}_{net}(t, m_i, c_j) = \ell^*$ - це вибір найоптимальнішого каналу зв'язку для задачі t, що виконується на маршруті $m_i \to c_j$, відповідно до функції оцінки.

Кожен канал зв'язку можна подати у вигляді:

$$\ell_r = \langle \mathcal{P}_{\ell_r}, \mathcal{F}_{\ell_r}, \mathcal{R}_{net} \rangle, \tag{2.13}$$

Система прагне мінімізувати комунікаційні затрати та забезпечити необхідний рівень QoS:

$$\min_{\ell_r \in \mathcal{L}} [\beta_1 \cdot Latency(\ell_r) + \beta_2 \cdot PacketLoss(\ell_r) - \beta_3 \cdot Bandwidth(\ell_r)]$$

де β_i — ваги, що задають пріоритет для кожного критерію в конкретному контексті задачі.

Система має хмарну інфраструктура, тобто високопродуктивні сервери (CPU, GPU, TPU) для обробки складних обчислювальних задач.

Її основна роль – це підтримка горизонтального масштабування, оркестрації контейнерів (Kubernetes), віртуалізації (VMs, Docker).

Хмарну інфраструктура може бути реалізована у вигляді приватної, публічної або гібридної хмари.

Хмарна інфраструктура забезпечує виконання обчислювально-інтенсивних задач на віддалених високопродуктивних ресурсах.

Вона включає засоби масштабування, віртуалізації, контейнеризації, а також підтримує гнучку оркестрацію задач у хмарному середовищі.

Хмарні вузли опишемо множиною:

$$\mathcal{C} = \{c_1, c_2, \ldots, c_n\}, \tag{2.14}$$

де кожен елемент $c_j \in \mathcal{C}$ - це обчислювальний ресурс хмарної інфраструктури, який має певні характеристики обробки задач.

Хмарна інфраструктура забезпечує виконання задач у середовищах віртуалізації:

$$\mathcal{V} = \{VM, Container, Bare - metal\}, \tag{2.15}$$

Кожен вузол c_j підтримує певні типи середовищ:

$$\mathcal{V}_{c_j}\theta \subseteq \mathcal{V}, \tag{2.16}$$

Характеристики хмарного вузла описуються множиною ресурсів:

$$\mathcal{R}_{c_j}^{\text{ресурси}} = \{GPU_{cores}, GPU_{units}, TPU_{units}, RAM, Storage, Load_{current}, \ldots\}, \tag{2.2}$$

Тип розгортання хмарної інфраструктури задамо як:

$$\mathcal{H} = \{Public, Private, Hybrid\}, \tag{2.17}$$

Для кожного вузла $c_j \in \mathcal{C}$ визначено тип хмари:

$$Type_{c_j} \in \mathcal{H}, \tag{2.18}$$

Хмарна система включає оркестратор, що виконує розподіл задач по віртуалізованих середовищах:

$$\mathcal{O} : \mathcal{T} \times \mathcal{R}_{c_j}^{\text{ресурси}} \times \mathcal{V}_{c_j} \to Instance_t, \tag{2.19}$$

де:

- $\mathcal{T}$ — множина задач,

- $\mathcal{O}$ створює середовище виконання задачі t у вузлі c_j із заданим типом віртуалізації та необхідними ресурсами.

Хмарна інфраструктура підтримує горизонтальне масштабування (додавання нових вузлів), яке формалізується як функція:

$$\mathcal{S}_{scale} : \mathcal{L}_{load} \to \Delta\mathcal{C}, \tag{2.20}$$

де $\mathcal{L}_{load}$ — рівень навантаження системи, а $\Delta\mathcal{C}$ — множина доданих або вилучених вузлів.

Кожен хмарний вузол c_j можна подати як:

$$c_j = \langle \mathcal{R}_{c_j}^{\text{ресурси}}, \mathcal{V}_{c_j}, Type_{c_j}, \mathcal{O}, \mathcal{S}_{scale} \rangle, \tag{2.21}$$

Система виконує розміщення задач із мінімізацією витрат і максимізацією продуктивності:

$$\min_{c_j \in \mathcal{C}} \left[y_1 \cdot ExecutionTime(t, c_j) + y_2 \cdot Cost(c_j) - y_3 \cdot Utilization(c_j) \right], \tag{2.22}$$

де y_i — вагові коефіцієнти, які відображають важливість кожної метрики.

Система включає обчислювальні модулі у хмарі - спеціалізовані модулі для паралельних обчислень, глибокого навчання, обробки відео тощо.

Обчислювальні модулі у хмарі застосовуються для для використання Big Data платформ (наприклад, Apache Spark, Hadoop), ML-фреймворків (TensorFlow, PyTorch).

Обчислювальні модулі у хмарі забезпечують виконання ресурсоємних задач, таких як обробка великих даних, глибоке навчання, відеоаналіз, з використанням спеціалізованих фреймворків і технологій.

Вони використовують високопродуктивні платформи обчислень, що підтримують паралелізм, розподілену обробку та апаратне прискорення.

Опишемо множину обчислювальних рушіїв як:

$$\mathcal{E} = \{e_1, e_2, \ldots, e_q\}, \tag{2.23}$$

де кожен рушій $e_k \in \mathcal{E}$ — це спеціалізований обчислювальний модуль, що виконує задачі певного типу в хмарному середовищі.

Кожен модуль належить до певного класу відповідно до призначення:

$$\mathcal{E}_{\text{тип}} = \{BigData, DeepLearning, VideoProcessing, ScientificSim, \ldots\}, \tag{2.2}$$

Кожен рушій e_k виконує задачі певного типу:

$$Type_{e_k} \in \mathcal{E}_{\text{тип}}, \tag{2.24}$$

Кожен рушій підтримує певні програмні технології або фреймворки:

$$\mathcal{F}_{e_k} = \{TensorFlow, PyTorch, Apache\ Spark, Hadoop, OpenCV, FFmpeg, \ldots\}, \tag{2.25}$$

Рушій описується характеристиками доступних обчислювальних ресурсів:

$$\mathcal{R}_{e_k}^{\text{ресурси}} = \{GPU_{cores}, GPU_{units}, TPU_{units}, Memory, Storage, \ldots\}, \tag{2.26}$$

Задачі призначаються рушіям відповідно до їхньої спеціалізації та ресурсної доступності:

$$\mathcal{M}_{engine} : \mathcal{T} \times \mathcal{E}_{\text{тип}} \times \mathcal{F}_{e_k} \times \mathcal{R}_{e_k}^{\text{ресурси}} \to \mathcal{E}, \tag{2.27}$$

де:

- $\mathcal{T}$ - множина задач,

- $\mathcal{M}_{engine}(t,\cdot,\cdot,\cdot) = e_k$ - вибір рушія для задачі t на основі відповідності її типу та вимог.

Кожен рушій описується як:

$$e_k = \langle Type_{e_k}, \mathcal{F}_{e_k}, \mathcal{R}_{e_k}^{\text{ресурси}}, \mathcal{M}_{engine} \rangle, \tag{2.28}$$

Призначення задач у модулі здійснюється з урахуванням критеріїв обчислювальної ефективності:

$$\min_{e_k \in \mathcal{E}} [\delta_1 \cdot ExecutionTime(t, e_k) + \delta_2 \cdot ResourceUsage(e_k) + \delta_3 \cdot FrameworkSwitchCost(e_k)]$$

де:

- δ_i - вагові коефіцієнти, що визначають пріоритети між швидкістю, ресурсною ефективністю та витратами на перемикання між фреймворками.

Система включає сховище даних. Це надійне збереження проміжних та кінцевих результатів обчислень.

Сховище забезпечує підтримку розподілених файлових систем (HDFS, Ceph, Amazon S3), можливість кешування результатів для зменшення повторних обчислень.

Сховище даних відповідає за збереження, доступність і цілісність проміжних та кінцевих результатів обчислень. Воно підтримує розподілені файлові системи, ефективне кешування і взаємодію з обчислювальними рушіями.

Позначимо множину доступних сховищ як:

$$\mathcal{D} = \{d_1, d_2, \ldots, d_r\}, \tag{2.29}$$

де кожне сховище $d_l \in \mathcal{D}$ є логічною або фізичною одиницею зберігання даних у хмарі.

Кожне сховище реалізоване з використанням певної платформи:

$$\mathcal{D}_{\text{тип}} = \{HDFS, Ceph, Amazon\ S3, Azure\ Blob, Local1FS, \ldots\}$$

Для кожного сховища визначено тип:

$$Type_{d_l} \in \mathcal{D}_{\text{тип}}, \tag{2.30}$$

Сховище може включати кешуючі механізми для уникнення повторних обчислень:

$$\mathcal{C}_{d_l} = \{Enabled, Disabled\}, \tag{2.31}$$

де $\mathcal{C}_{d_l} = Enabled$ означає, що кешування активне на рівні сховища d_l.

Взаємодію сховища з результатами обчислень формалізуємо як функцію:

$$\mathcal{S}_{save} : \mathcal{T} \times \mathcal{E} \to \mathcal{D}, \tag{2.32}$$

де:

- $\mathcal{T}$ - множина задач,

- $\mathcal{E}$ - обчислювальні рушії,

- $\mathcal{S}_{save}(t, e_k) = d_l$ - означає, що результати задачі t, виконаної рушієм e_k, зберігаються у сховищі d_l.

Множина характеристик якості сховища:

$$\mathcal{Q}_{storage} = \{q_1, q_2, q_3\} =, \quad (2.2)$$

де:

q_1 - час доступу;

q_2 - пропускна здатність;

q_3- надійність.

Оптимізаційна ціль:

$$\min_{d_l \in \mathcal{D}} [\lambda_1 \cdot q_1(d_l) - \lambda_2 \cdot q_2(d_l) - \lambda_3 \cdot q_3(d_l)], \quad (2.33)$$

де λ_i - вагові коефіцієнти для оцінки ефективності вибору сховища.

Кожен елемент d_l описується як:

$$d_l = \langle Type_{d_l}, \mathcal{C}_{d_l}, \mathcal{Q}_{storage}, \mathcal{S}_{save} \rangle, \quad (2.34)$$

Сховище дозволяє паралельний доступ з різних модулів системи (планувальники, рушії, аналітичні модулі):

$$\mathcal{A}_{access} : \mathcal{M} \cup \mathcal{C} \cup \mathcal{E} \to 2^{\mathcal{D}}, \quad (2.35)$$

де $\mathcal{A}_{access}(x)$ - набір сховищ, доступних для модуля x (мобільний пристрій, хмарний вузол або рушій).

Система включає модуль безпеки та автентифікації (Security & Privacy Module), яка забезпечує автентифікацію користувачів, шифрування даних при передачі та зберіганні.

Модуль безпеки та автентифікації також призначений для використання протоколів OAuth, TLS, VPN, Zero Trust Architectures.

Модель системи включає модуль безпеки та автентифікації.

Цей компонент критично важливий для захисту конфіденційності, цілісності та доступності даних, а також контролю доступу до хмарної інфраструктури.

Модуль забезпечує автентифікацію, авторизацію, шифрування, а також реалізацію сучасних протоколів і архітектур безпеки в комп'ютерних межах всієї системи.

Основна мета - запобігання несанкціонованому доступу, витокам та атакам на дані й сервіси.

Позначимо множину агентів безпеки як:

$$\mathcal{Z} = \{z_1, z_2, \ldots, z_s\}, \tag{2.36}$$

де кожен агент $z_j \in \mathcal{Z}$ відповідає за безпековий контекст певного компонента системи (мобільного пристрою, хмари, мережі, сховища).

Модуль підтримує множину протоколів автентифікації:

$$\mathcal{A}_{auth} = \{OAuth2, OpenID\ Connect, SAML, Biometric, \ldots\}, \tag{2.2}$$

Для кожного агента z_j визначено автентифікаційний механізм:

$$Auth_{z_j} \in \mathcal{A}_{auth}, \tag{2.37}$$

Протоколи безпечного з'єднання використовуються для забезпечення шифрування при передаванні даних:

$$\mathcal{P}_{conn} = \{TLS, VPN, IPSec, HTTPS, \ldots\}, \tag{2.38}$$

$$SecureConn_{z_j} \subseteq \mathcal{P}_{conn}$$

Визначимо множину моделей політик доступу:

$$\mathcal{A}_{access} = \{RBAC, ABAC, Zero\ Trust, \dots\}$$

де:

- RBAC — керування доступом на основі ролей;
- ABAC — керування доступом на основі атрибутів;
- Zero Trust — динамічна модель постійної перевірки.

Формалізуємо як функцію:

$$\mathcal{S}_{sec} : \mathcal{U} \times \mathcal{R} \times \mathcal{C} \to \{0,1\}, \quad (2.39)$$

де:

- $\mathcal{U}$ - множина користувачів,
- $\mathcal{R}$ - ресурси (дані, обчислення, інтерфейси),
- $\mathcal{C}$ - контекст доступу (роль, час, локація, пристрій),
- $\mathcal{S}_{sec}(u, r, c) = 1$ означає дозвіл доступу, $= 0$ — заборона.

Описується як: $\mathcal{G}_{security} = \{g_1, g_2, g_3\} =$ {Конфіденційність, Цілісність, Доступність}.

Оптимізаційна ціль — мінімізувати ризики для кожної мети:

$$\min_{z_j \in \mathcal{Z}} \left[\mu_1 \cdot Risk_{g_1}(z_j) + \mu_2 \cdot Risk_{g_2}(z_j) + \mu_3 \cdot Risk_{g_3}(z_j)\right]$$

Кожен агент описується як:

$$z_j = \langle Auth_{z_j}, SecureConn_{z_j}, \mathcal{A}_{access}, \mathcal{S}_{sec}, \mathcal{G}_{security} \rangle$$

Система включає інтерфейс користувача - це мобільний застосунок або вебінтерфейс для керування обчисленнями, перегляду результатів, а також для підтримки асинхронної взаємодії, push-сповіщень про завершення задач.

Інтерфейс користувача забезпечує взаємодію людини з системою, зокрема подання задач, відображення результатів та отримання сповіщень.

Формалізуємо основні дії користувача:

$$\mathcal{F}_{i_k} = \{SubmitTask, ViewResult, ReceiveNotification, ManageSettings\}$$

$$\mathcal{U}_{interaction} : \mathcal{U} \times \mathcal{F}_{i_k} \to \mathcal{S}, \quad (2.40)$$

де:

- $\mathcal{U}$ - множина користувачів,

- $\mathcal{S}$ - система або її окремі сервіси.

Інтерфейс реалізує підтримку асинхронних подій:

$$\mathcal{N}_{push} : \mathcal{T} \times \mathcal{E} \to \mathcal{U}, \quad (2.2)$$

де:

- задача $t \in \mathcal{T}$, виконана рушієм $e \in \mathcal{E}$, тригерить сповіщення користувачу $u \in \mathcal{U}$.

$$i_k = \langle Type_{i_k}, \mathcal{F}_{i_k}, \mathcal{U}_{interaction}, \mathcal{N}_{push} \rangle, \quad (2.41)$$

Система включає модуль моніторингу і логування для відстеження продуктивності, навантаження, споживання ресурсів, а також для інтеграції з інструментами на кшталт Prometheus, Grafana, ELK Stack.

Модуль відповідає за спостереження за станом компонентів системи, збір логів, а також візуалізацію продуктивності.

Задамо множину моніторингових елементів:

$$\mathcal{M}_L = \{m_1, m_2, \ldots, m_z\}, \quad (2.42)$$

де кожен m_z відслідковує ресурси або сервіси в системі.

Задамо функція моніторингу:

$$\mathcal{M}_{observe} : \mathcal{E} \cup \mathcal{M} \cup \mathcal{C} \to 2^{\mathcal{Q}monitor}, \quad (2.243$$

Множина логів описується так:

$$\mathcal{L} = \{l_1, l_2, \ldots, l_n\}, \quad (2.44)$$

де кожен лог формується відповідно до події:

$$\mathcal{G}_{log} : \mathcal{T} \cup \mathcal{U} \cup \mathcal{E} \to \mathcal{L}, \quad (2.45)$$

Формальна структура моніторингового агента така:

$$m_z = \langle \mathcal{Q}_{monitor}, \mathcal{M}_{observe}, \mathcal{G}_{log}, \mathcal{T}_{mon} \rangle, \quad (2.46)$$

2.1.1 Вхідні параметри

Подамо формалізовану математичну модель системи, що поєднує мобільні пристрої та хмарну інфраструктуру для ефективного виконання високопродуктивних обчислень.

Така система передбачає гібридний підхід до обробки задач, у якому рішення про локальне або віддалене (у хмарі) виконання задачі приймається на основі адаптивного аналізу вхідних параметрів, ресурсної доступності, енергоспоживання та обмежень затримки.

Приймемо нехай $T = \{t_1, t_2, \ldots, t_n\}$ - множина задач, які надходять до системи.

Кожна задача t_i характеризується обсягом вхідних даних $D(t_i)$, вимогами до якості обслуговування $Q(t_i)$, а також ресурсом виконання $R(t_i)$.

У моделі розглядаються два основні виконавці: мобільний пристрій (M) та хмарна інфраструктура (C), з відповідними ресурсами R^m та R^c, пропускною здатністю каналу зв'язку B, затримкою передачі даних L і параметрами енергоспоживання E^M,E^C.

2.1.2 Розподіл задач між пристроєм і хмарою

Для моделювання вибору місця обробки задачі введемо бінарну змінну x_i:

$$x_i = \begin{cases} 1, \text{якщо задача } t_i \text{виконується у хмарі} \\ 0, \text{якщо задача } t_i \text{ виконується локально на мобільному пристрої} \end{cases}$$

2.1.3 Час виконання задачі

Загальний час обробки задачі визначається як:

$$T_{total}(t_i) = (1 - x_i) \cdot T_M(t_i) + x_i \cdot (T_{net}(t_i) + T_C(t_i)) , \quad (2.47)$$

де:

$$T_{net}(t_i) = \frac{D(t_i)}{B} + L$$

є часом передавання даних через мережу, що включає затримку передачі та мережеву латентність.

2.1.4 Енергоспоживання

Загальне енергоспоживання на обробку задачі t_i визначається як:

$$E_{total}(t_i) = (1 - x_i) \cdot E_M(t_i) + x_i \cdot (E_{t_x}(t_i) + E_{r_x}(t_i)) , \quad (2.48)$$

де $E_{t_x}(t_i)$ та $E_{r_x}(t_i)$ — енергетичні витрати на передавання та отримання даних при віддаленому виконанні задачі.

2.1.5 Ресурсні обмеження

Для забезпечення коректного розподілу задач встановлюються обмеження на обчислювальні ресурси:

$$\sum_{i=1}^{n}(1 - x_i) \cdot R_M(t_i) \leq R_M^{max}, \quad (2.49)$$

$$\sum_{i=1}^{n} x_i \cdot R_C(t_i) \leq R_C^{max}, \quad (2.50)$$

де R_M^{max}, R_C^{max} — максимальна доступна кількість обчислювальних ресурсів для мобільного пристрою та хмари відповідно.

2.1.6 Обмеження якості обслуговування (QoS)

Кожна задача повинна бути виконана у визначений термін:

$$T_{total}(t_i) \leq Q(t_i), \forall_i \, , \quad (2.60)$$

2.1.7 Врахування накладних витрат на безпеку

Параметр безпеки системи (наприклад, шифрування даних) впливає на час обробки задачі. Це можна врахувати як додаткову затримку $\delta(S)$:

$$T_{total}(t_i) = T_{total}(t_i) + \delta(S) \, , \quad (2.61)$$

2.1.8 Цільова функція оптимізації

Метою є мінімізація загального часу обробки задач при мінімальному енергоспоживанні.

Формально цільова функція має вигляд:

$$\min_{x_i}\{\sum_{i=1}^{n}[\alpha \cdot T_{total}(t_i) + \beta \cdot E_{total}(t_i)]\} \, , \quad (2.62)$$

де $\alpha, \beta \in \mathbb{R}^+$ — вагові коефіцієнти, що відображають пріоритетність метрик продуктивності та енергоефективності.

2.1.9 Розширення моделі

У разі реалізації підтримки багатопотокових обчислень або інкрементального виконання задач, модель можна розширити за допомогою множини потоків $P_j \subset T$, що реалізуються паралельно.

Додатково для підвищення адаптивності системи може бути використана функція передбачення на основі машинного навчання $f_0(\cdot)$, яка формує рішення про офлоадінг задач залежно від поточних параметрів системи:

$$x_i = f_0(D(t_i), R_M, B, L, Q(t_i), \dots) , \tag{2.63}$$

2.2 Формалізація компонентів системи у вигляді множин та функцій взаємодії

У межах побудови узагальненої моделі системи високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій, введемо множини, що відповідають основним функціональним компонентам системи, а також визначимо функції взаємодії між ними.

Розглянемо компоненти системи. Позначимо множини компонентів системи наступним чином:

$\mathcal{M}$ — множина мобільних пристроїв-клієнтів,

$\mathcal{M} = \{m_1, m_2, \dots, m_k\}$;

$\mathcal{F}$ – множина мобільних обчислювальних фреймворків (middleware),

$\mathcal{F} = \{f_1, f_2, \dots, f_r\}$;

$\mathcal{S}$ — множина модулів адаптивного планування задач,

$\mathcal{S} = \{s_1, s_2, \dots, s_r\}$;

$\mathcal{N}$ — множина мережевих каналів зв'язку,

$\mathcal{N} = \{n_{WiFi}, n_{4G}, n_{5G}, \dots\}$;

$\mathcal{C}$ — множина хмарних інфраструктур;

$\mathcal{C} = \{c_1, c_2, \dots, c_l\}$;

$\mathcal{E}$ — множина хмарних обчислювальних модулів;

$\mathcal{E} = \{e_1, e_2, \dots, e_q\}$;

$\mathcal{D}$ — множина хмарних сховищ даних,

$\mathcal{D} = \{d_1, d_2, \dots, d_v\}$;

$\mathcal{P}$ — множина модулів безпеки та автентифікації,

$\mathcal{P} = \{p_1, p_2, \dots, p_u\}$;

$\mathcal{U}$ — множина інтерфейсів користувача,

$\mathcal{U} = \{u_1, u_2, \dots, u_s\}$.;

$\mathcal{L}$ — множина систем моніторингу і логування,

$\mathcal{L} = \{l_1, l_2, \dots, l_t\}$.;

Визначимо множини задач:

$\mathcal{T} = \{t_1, t_2, \dots, t_n\}$ — множина задач, які необхідно обробити в системі.

Задамо функцію розподілу задач:

$$\phi : \mathcal{T} \times \mathcal{M} \times \mathcal{S} \times \mathcal{N} \to \mathcal{C} \cup \mathcal{M}$$

де функція $\phi(t_i, m_j, s_k, n_l)$ визначає, чи задача t_i буде виконуватись локально на пристрої m_j, чи передана до хмари c_p з урахуванням рішення модуля планування s_k та мережевого стану n_l.

Задамо функцію виконання задач у хмарі

$\psi : \mathcal{T} \times \mathcal{C} \to \mathcal{E}$

Функція ψ визначає, який хмарний обчислювальний модуль $e_q \in \mathcal{E}$ відповідає за виконання задачі, призначеної на хмару.

Задамо функцію збереження результатів:

$\delta : \mathcal{T} \to \mathcal{D}$

Кожна задача після виконання має бути збережена в одному з елементів системи зберігання $\mathcal{D}$, що забезпечує сталість, доступність та повторне використання результатів.

Задамо функцію доступу до результатів

$\omega : \mathcal{D} \times \mathcal{U} \to$ Вивід користувачу

Ця функція описує передачу результатів з хмарного сховища до користувача через відповідний інтерфейс $u \in \mathcal{U}$.

Задамо функцію безпеки при обробці

$\sigma : (\mathcal{T} \times \mathcal{M} \times \mathcal{C}) \to \mathcal{P}$

Функція σ асоціює із кожною задачею, що передається між компонентами системи, відповідний механізм безпеки та політику автентифікації.

Задамо функцію моніторингу:

$$\mu : (\mathcal{T}, \mathcal{M}, \mathcal{C}, \mathcal{N}) \to \mathcal{L}\ , \tag{2.64}$$

Функція μ визначає зв’язок задач із системами логування та моніторингу для аналізу навантаження, часу виконання, ефективності використання ресурсів.

Задамо функцію функція взаємодії системи.

Повну поведінку системи можна представити через композицію взаємодіючих функцій:

$$\Omega : \mathcal{T} \to \mathcal{U}, \text{де } \Omega(t_i) = \omega(\delta\left(\psi\left(\phi(t_i, m_j, s_k, n_l)\right)\right), u_s). \tag{2.65}$$

Таким чином, задача t_i проходить через етапи планування, обчислення, зберігання і доступу, які координуються за допомогою модулів, визначених у множинах компонентів системи.

2.3 Висновки

У розділі було розроблено формалізовану модель системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

Запропонована модель охоплює всі ключові компоненти такої системи від мобільних клієнтських пристроїв до хмарної інфраструктури, адаптивного планування задач, модулів безпеки, сховищ даних та інтерфейсів користувача.

Формалізація здійснена через визначення множин об’єктів та функцій взаємодії, що дозволяє однозначно описати поведінку системи в умовах змінних обчислювальних ресурсів, мережевих характеристик і вимог до якості обслуговування.

В рамках даного підходу враховано вплив енергоспоживання, затримок, безпекових витрат і динамічного розподілу задач на ефективність роботи гібридної обчислювальної архітектури.

Розроблена модель є основою для розроблення системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

3 СИСТЕМА ДЛЯ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЕНЬ НА МОБІЛЬНИХ ПРИСТРОЯХ ІЗ ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

3.1 Метод здійснення високопродуктивних обчислень

Для вирішення задачі підвищення продуктивності обчислень на мобільних пристроях із використанням хмарних технологій було удосконалено метод здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій .

У дослідженні було використано поняття потоків (threads), які виконують роботу зі спільною пам'яттю та надають вичерпний набір примітивів їх синхронізації.

Використання потоків мотивоване їх здатністю до масштабування та перенесення на різні архітектури та операційні системи, а також їх роллю як фундаментального рівня для численних високорівневих абстракцій паралельного програмування.

Приймемо, нехай $T = \{t_1, t_2, \dots, t_n\}$ позначає множину потоків, що виконуються одночасно у спільному адресному просторі $\mathcal{M}$.

Кожен потік $t_1 \in T$ уможливлює координацію за допомогою механізмів синхронізації $S = \{s_1, s_2, \dots, s_m\}$, де $S \subset \mathcal{S}_{POSIX}$ позначає набір доступних примітивів синхронізації POSIX, таких як м'ютекси і змінні умов.

Такий набір положень забезпечує сумісність з широким спектром існуючих паралельних програмних систем, тим самим максимізуючи інтероперабельність і зменшуючи складність інтеграції.

Під час початкового виконання, позначеного як базовий запуск, потік обчислює вихідні дані з нуля та записує трасування виконання $\mathcal{T}_0$.

Нехай P – багатопотокове програмне забезпечення, а I_0 – її початковий вхід. Результат $O_0 = P(I_0)$ обчислюється, а траса $\mathcal{T}_0 = trace(P, I_0)$ зберігається для подальшого використання.

Для всіх наступних виконань, тобто поступових запусків, вхідні дані змінюються на $I' = I_0 + \Delta I$, де ΔI позначає вхідну дельту, яку можна вказати шляхом вказівки зсувів і довжини змінених сегментів.

Враховуючи $\mathcal{T}_0$ і ΔI, метод виконує поступове оновлення для обчислення нового виводу $O' = P(I')$ без повторного виконання всієї програми з нуля.

Замість цього він вибірково повторно виконує лише необхідні частини обчислення на основі записаного трасування та обсягу змін, тобто

$$O' = \mathcal{F}(O_0, \mathcal{T}_0, \Delta I), \tag{3.1}$$

де $\mathcal{F}$ позначає функцію інкрементального оцінювання.

Метод не лише покращує ефективність робочих навантажень із незначними варіаціями вхідних даних, але й підтримує коректність завдяки використанню детермінованого відтворення та детального відстеження залежностей, зафіксованих у результатах його застосування.

Запропонований удосконалений метод адаптує принципи самоналаштовуваних обчислень для мультипоточності з спільною пам’яттю, а також використовує техніки з систем запису-відтворення, що застосовуються для надійної мультипоточності.

Метод включає такі кроки:

1. Декомпозиція на підобчислення.
2. Побудова графа конкурентних динамічних залежностей.
3. Поширення змін і поступове повторне виконання.

Розглянемо кроки детальніше.

3.1.1 Декомпозиція на підобчислення

Обчислення програми P розділено на набір підобчислень, позначених $N = \{n_1, n_2, \dots, s_k\}$, де кожен $n_i \in N$ представляє логічно атомарну одиницю роботи

процесом (наприклад, базовий блок або локальну область потоку виконання процесу).

3.1.2 Побудова конкурентного динамічного графа залежностей

Під час початкового (базового) виконання потік записує трасування виконання для побудови конкурентний динамічний граф залежностей (КДГЗ) $\mathcal{G} = (N, \rightarrow)$, де частковий порядок $\rightarrow$ кодує динамічні зв'язки «відбувається до» між підобчисленнями. Ключовий інваріант правильності такий:

Для будь-якого підобчислення $n \in N$, приймемо, що:

$$\mathrm{M} = \{n \in N | m \rightarrow n\}, \qquad (3.2)$$

позначають набір безпосередніх динамічних попередників n.

Якщо під час наступного інкрементального запуску вхід до всіх $m \in M$ залишається незмінним, і порядок виконання відповідає початковому частковому порядку $\rightarrow$, тоді вхідні дані для n також гарантовано залишаться незмінними.

Отже, мемоізований ефект n - запам'ятовування результату виконання якоїсь операції, щоб повторно не виконувати цю ж операцію, якщо її вхідні дані не змінилися - можна безпечно повторно використовувати без повторного виконання процесу.

3.1.3 Поширення змін і поступове повторне виконання.

Під час інкрементного виконання, враховуючи модифікований вхід $I' = I_0 + \Delta I$, здійснюється поширення ефекту ΔI через графік $\mathcal{G}$.

Оновлене виконання дотримується топологічного порядку, що відповідає $\rightarrow$, повторно використовує попередньо запам'ятовані підобчислення, де вхідні дані не змінені, і вибірково повторює лише ті підобчислення, вхідні дані яких визнано недійсними.

Формально для кожного $n \in N$ здійснюється визначення предикату дійсності valid(n), такий, що:

$$\text{valid(n)} = \left(\forall \text{m} \in \text{M}, valid(m) \wedge input(m) = input_0(m)\right) \Rightarrow$$

$$\Rightarrow nput(n) = input_0(n). \quad (3.3)$$

Підобчислення, що задовольняють цей предикат, можна безпечно повторно використовувати, забезпечуючи правильність і мінімізуючи надлишкові обчислення.

Метод забезпечує основу для ефективного поступового виконання в багатопоточних середовищах, використовуючи аналіз динамічних залежностей для максимального повторного використання та мінімізації накладних витрат.

3.2 Застосування методу високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій

Розглянемо роботу методу на прикладі, який рзглядається, ілюструє поетапне виконання в контексті багатопоточного програмного забезпечення, що включає два потоки, T_1 і T_2 , які отримують доступ і змінюють три спільні змінні (x, y, z), використовуючи взаємне виключення через примітиви блокування.

Цей приклад конкретно демонструє побудову та корисність графа конкурентної динамічної залежності, як було представлено раніше.

3.2.1 Крок1. Ідентифікація підобчислень

Кожне виконання потоку розбивається на підобчислення на межах примітивів синхронізації, зокрема на кожній парі `lock()`/`unlock()`.

Цей вибір межі відображає деталізацію атомарних секцій, які синхронізують доступ до спільного стану.

Позначимо ідентифіковані субобчислення так:

- T_1.a: критичний розділ, що виконується потоком T_1;

- T_2.a, T_2.b: дві критичні секції, що виконуються потоком T_2.

Припустимо, що під час початкового (базового) запуску планувальник потоків вибирає чергування:

$$T_1.a \rightarrow T_2.a \rightarrow T_2.b, \tag{3.4}$$

представляючи загальний порядок, що відповідає хронології отримання блокування.

3.2.2 Крок2. Побудова графа конкурентних динамічних залежностей

Щоб побудувати граф конкурентних динамічних залежностей $\mathcal{G} = (N, \rightarrow)$, система повинна фіксувати як порядок, викликаний синхронізацією, так і залежності даних між субобчисленнями.

Розглянемо випадок, коли вхідні дані змінено — наприклад, змінну у змінено.

Нехай $R(n)$ і $W(n)$ позначають набір для читання і набір для запису, відповідно, підобчислення $n \in N$.

Граф конкурентних динамічних залежностей здійснення поширення змін шляхом аналізу цих наборів:

- Оскільки $T_1.a \in$ N читається y і $y \in \Delta$I, необхідно перерахувати $T_1.a$

- $T_2.a$ не залежить від y (тобто $y \notin R(T_2.a)$), і на жодну зі змінних, які він читає або записує, не впливає $T_1.a$; отже, його можна повторно використовувати.

- $T_2.b$ потрібно перерахувати, незважаючи на відсутність прямої залежності від y через транзитивну залежність даних: $T_1.a$ записує в z, а $T_2.b \in$ R(z), встановлюючи непрямий шлях впливу.

Таким чином, дані залежності форми

$$\exists m \in \mathrm{N} : \mathrm{W(m)} \cap \mathrm{R}(n) \neq \emptyset$$

повинні бути захоплені транзитивно, щоб забезпечити правильне поширення змін.

Багатопотокові програми, як правило, є недетермінованими, оскільки планувальник ОС може по-різному чергувати підобчислення між виконаннями. Розглянемо альтернативний розклад:

$$T_2.a \rightarrow T_2.b \rightarrow T_1.a,$$

у якому T_1 отримує блокування після $T_2.b$.

Навіть без будь-яких модифікацій вхідних даних стан спільних змінних (наприклад, $\mathcal{y}$, що спостерігаються під час обчислень, може відрізнятися від базового прогону.

У результаті як $T_1.a$, так і $T_2.b$ може знадобитися повторне виконання поточної задачі.

Тому, щоб уникнути фальшивих повторних обчислень, події синхронізації повинні бути явно зафіксовані в →, щоб забезпечити еквівалентність розкладу.

Якщо всі обмеження синхронізації збережено, а вхідні дані не змінені, запам’ятовані підобчислення є дійсними та безпечними для повторного використання.

3.2.3 Крок 3. Розповсюдження змін в інкрементному виконанні

Поступове виконання відбувається шляхом відвідування підобчислень у топологічному порядку, який відповідає →.

Для кожного підобчислення n алгоритм використовує свої записані набори читання і запису для визначення повторного використання:

Нехай:

- $changed(d) = True$, якщо розташування даних d було змінено після базового запуску.

- $recompute(n) = \exists d \in R(n) : changed(d)$.

Потім:

- Якщо $recompute(n) = True$, ми повторно виконуємо n і оновлюємо $W(n)$ у пам'яті та наборах відстеження.

- Якщо $recompute(n) = False$, ми пропускаємо виконання n і матеріалізуємо його наслідки, записуючи запам’ятовані значення $W(n)$ безпосередньо в адресний простір.

Формально нехай $V: N \to \{True, False\}$ буде картою дійсності. Тоді правило поширення таке:

$$V(n) = \begin{cases} False, & if \exists m \to n: \neg V(m) \vee changed\big(W(m)\big) \cap R(n) \neq \emptyset \\ True, otherwise \end{cases}$$

Це забезпечує точне й ефективне повторне використання, зберігаючи семантичну коректність всупереч змінам вхідних даних і обмеженням упорядкування, спричиненим синхронізацією.

Приклад демонструє необхідність запису як залежностей даних (через набори читання/запису), так і порядків синхронізації (через →) у CDDG.

Це подвійне відстеження забезпечує точне розповсюдження змін і ефективне поступове повторне виконання навіть у складних, недетермінованих багатопоточних середовищах.

3.3 Модель системи

3.3.1 Модель узгодженості пам'яті

Частковий порядок виконання підобчислень, позначений як →, неявно фіксує залежності типу читання-після-запису (read-after-write, RAW) між підобчисленнями, використовуючи множини читання $R(n) \subseteq D$ та запису $W(n) \subseteq D$ для кожного підобчислення $n \in$ N, де D — множина адрес доступу до пам'яті.

Визначення і збереження відношення → є критичним для коректності інкрементального виконання та залежить від обраної моделі пам'яті, яка, у свою чергу, визначає допустиму гранулярність підобчислень.

У системі було обрано модель узгодженості пам’яті Release Consistency [81], яка забезпечує баланс між коректністю та ефективністю.

На відміну від моделі Sequential Consistency (SC) [101], яка передбачає, що всі операції пам’яті з усіх потоків повинні бути тотально впорядковані, RC дозволяє відтерміновану видимість оновлень пам’яті, при цьому гарантує, що оновлення, зроблені потоком, стають видимими для інших потоків лише в точках синхронізації.

Приймемо, що $S \subset N$ – множина підобчислень, що містять операції синхронізації (наприклад, lock(), unlock()), тоді модель RC визначає, що:

$$\forall m, n \in N: (m \notin S \wedge n \notin S \wedge W(m) \cap R(n) \neq \emptyset) \nRightarrow m \rightarrow n.$$

Інакше кажучи, за відсутності синхронізації між підобчисленнями mm та nn, навіть наявність конфлікту доступу до пам’яті (RAW) не встановлює між ними залежності у частковому порядку.

Це дозволяє суттєво зменшити кількість фіксованих залежностей у КДГЗ і обмежити межі підобчислень точками синхронізації, тобто:

$$Granularity(n) = Boundary(Synchronization\ Events.$$

На противагу цьому, у моделі SC кожна інструкція, що здійснює доступ до спільної пам’яті, потенційно має міжпотоковий ефект, а тому повинна трактуватись як окреме підобчислення:

$$GranularitySC(n) = Instruction\ Level.$$

Це призводить до значного накладного обліку та втрати переваг інкрементального виконання.

Застосування моделі RC дозволяє зменшити обсяг інформації, яку потрібно відстежувати, забезпечуючи при цьому семантичну еквівалентність стандартам POSIX Threads (pthreads), де також вимагається, щоб усі звернення до спільних структур даних були узгоджені через примітиви синхронізації.

Таким чином, семантика доступу до пам’яті гарантує, що всі зміни стають видимими іншим потокам лише після виконання відповідних примітивів синхронізації.

Отже, використання моделі RC як основи для побудови часткового порядку $\rightarrow$ у графі залежностей забезпечує коректну та ефективну основу для інкрементального багатопотокового виконання з керованими накладними витратами.

3.3.2 Модель виконання

Нехай $P = \{T_1, T_2, \dots, T_k\}$ – множина потоків програми. Кожен потік T_i виконує послідовність інструкцій, які ми групуємо у підобчислення $n \in N$, що обмежуються точками синхронізації відповідно до моделі Release Consistency.

Таким чином, виконання програми представлено як множина підобчислень $N = U_{i=1}^{k} N_i$, де N_i – впорядковане виконання підобчислень у потоці T_i.

Виконання програми моделюється як побудова часткового порядку $O = (N, \rightarrow)$, де $\rightarrow \subseteq N \times N$ – відношення залежності.

Це відношення фіксує як синхронізаційні залежності, які виникають між підобчисленнями, пов'язаними через lock()/unlock(), join() або інші бар'єри; а також, як дані залежностей: якщо $m \rightarrow n$, то $W(m) \cap R(n) \neq \emptyset$, тобто підобчислення nn читає значення, записані у m.

КДГЗ підтримує обидва види залежностей і використовується для аналізу впливу змін на результат виконання.

Формально, якщо для деякого підобчислення $n \in N$ множина всіх його попередників (прямих та транзитивних) $M = \{m \in N \mid m \to* n\}$ не зазнала змін (ані вхідних даних, ані розкладу), то підобчислення nn може бути повторно використане без переобчислення:

$$Reuse(n) \Leftrightarrow \forall \mathrm{m} \in M : Input(m) = Input_{prev}(m).$$

3.3.3 Модель синхронізації

Для досягнення поставленої мети необхідним є повна підтримка стандартного набору примітивів синхронізації.

Така підтримка передбачена API pthreads, зокрема, pthread_mutex_t, pthread_cond_t, pthread_barrier_t, pthread_spinlock_t тощо.

Ці примітиви є точками узгодження згідно з Release Consistency (RC) моделлю, визначаючи межі підобчислень, у яких можлива міжпотокова взаємодія.

Важливим зауваженням є те, що пропонований метод не підтримує спеціалізовані (нестандартні) механізми синхронізації, зокрема визначені користувачем спін-блокування, що реалізуються вручну через атомарні інструкції, наприклад:

```
while (!atomic_compare_and_swap(...))
    {}
```

Це обмеження пов’язане з тим, що такі механізми не генерують явних точок синхронізації, які можна було б відстежити для побудови коректного порядку → у КДГЗ.

RC-модель вимагає явної фіксації подій узгодження, без чого міжпотокові залежності залишаються невизначеними, що порушує гарантії інкрементальності та детермінованості.

Хоча нестандартні методи синхронізації іноді використовуються задля гнучкості або продуктивності, це може призводити до збоїв, гонок або неочевидних деградацій продуктивності.

У багатьох випадках такі механізми можуть бути безболісно замінені на стандартні примітиви.

Таким чином, метод передбачає розширення моделі механізмами для визначення користувацьких точок синхронізації на основі низькорівневих примітивів, таких як вбудовані атомарні операції, що дозволить визначати власні межі підобчислень і забезпечить прозору підтримку структур даних без блокування (lock-free) або без очікування (wait-free).

Інкрементальне виконання запускається після початкового прогону, в ході якого будується CDDG=(N,→), і запам’ятовуються множини читання та запису кожного підобчислення:

$$R : N \to 2^D, W : N \to 2^D$$

Під час інкрементального запуску користувач вказує змінені області вхідних даних, тобто множину $D\Delta \subseteq D$, де відбулись модифікації. Система ідентифікує всі підобчислення nnn, для яких:

$$R(n) \cap D_\Delta \neq \emptyset \text{ або } \exists m \to * n \text{ де } W(m) \cap D_\Delta \neq \emptyset$$

Такі підобчислення маркуються як застарілі й здійснюється процес переобчислення.

Інші підобчислення, чия множина читання залишилась незмінною, використовують запам'ятовані результати, що істотно знижує обсяг необхідних обчислень.

3.4 Формалізація інкрементального багатопотокового виконання

Розглянемо формалізацію обчислювальної моделі, що лежить в основі системи методу, та опишемо ключові алгоритми, які реалізують підтримку інкрементального багатопотокового виконання.

Ці алгоритми відображають два режими роботи системи: початкове (повне) виконання, що здійснює запуск програми з нуля, та інкрементальне виконання, яке повторно використовує результати попередніх обчислень на основі змін у вхідних даних.

Центральним компонентом нашого підходу є конкурентний динамічний граф залежностей (КДГЗ), який визначає частковий порядок $\rightarrow \subseteq N \times N$ між множиною підобчислень *N*.

Для кожного підобчислення $n \in N$ з відповідними множинами читання $R(n) \subseteq \mathcal{A}$ та запису W(n)$\subseteq \mathcal{A}$, де $\mathcal{A}$ – універсум адрес пам’яті, ми фіксуємо інформацію про залежності даних та синхронізації.

Під час початкового запуску відбувається виконання програми у звичайному режимі з нуля.

Усі підобчислення $n \in N$ виконуються у порядку, сумісному з міжпотоковими подіями синхронізації, які формують граф порядку $\rightarrow$.

В процесі виконання для кожного підобчислення зберігаються множини R(n), W(n) та ефекти виконання E(n), що включають відповідні значення змінних у вихідному стані.

Під час інкрементального запуску виконується поширення змін через застосування КДГЗ.

Задані модифікації вхідних даних, наприклад у вигляді змін у файлах або вказівки зсуву та довжини змінених байтів, використовуються для визначення множини підобчислень N′⊆N, для яких щонайменше одне значення в R(n) зазнало змін порівняно з попереднім запуском.

Ці підобчислення обчислюються повторно, а всі інші — пропускаються з відновленням ефектів із E(n).

Розглянемо формальне представлення алгоритму побудови КДГЗ під час початкового запуску, а також алгоритму його використання для ефективного інкрементального виконання.

3.5 Одночасний динамічний граф залежностей (КДГЗ)

КДГЗ - це орієнтований ациклічний граф з вершинами, що представляють підобчислення, і двома типами ребер для запису залежностей між тунками: ребрами, що передують ребрам, і ребрами, що залежать від даних. Далі ми пояснимо, як отримати вершини та ребра.

Визначимо підобчислення як послідовність інструкцій, що виконуються потоком між двома викликами синхронізації.

Моделюватимемо виконання потоку t як послідовність підобчислень (L_t). Підобчислення у потоці повністю впорядковані на основі порядку їх виконання з використанням монотонно зростаючого лічильника.

Конкурентний динамічний граф залежностей (КДГЗ, англ. Concurrent Dynamic Dependency Graph, CDDG) є орієнтованим ациклічним графом G=(V,E), де множина вершин VVV представляє підобчислення, а множина ребер E складається з двох підтипів: ребер порядку виконання та ребер залежності від даних.

Граф кодує як внутрішньопотоковий порядок виконання, так і міжпотокові залежності, що виникають внаслідок синхронізації або доступу до наявної спільної пам’яті.

Кожне підобчислення є атомарною одиницею обчислення і визначається як послідовність інструкцій, що виконується потоком t між двома послідовними викликами API синхронізації.

Виконання потоку t моделюється як впорядкована послідовність підобчислень:

$$L(t) = \langle L(t)[0], L(t)[1], \dots, L(t)[n] \rangle,$$

де кожне підобчислення $L(t)[\alpha]$ маркується монотонно зростаючим лічильником $\alpha \in N$.

Модель синхронізації інтерпретується через примітиви як операції захоплення (acquire) та звільнення (release).

Якщо потік виконує, наприклад, unlock(S) на об'єкті синхронізації S, це інтерпретується як операція звільнення, тоді як відповідний lock(S) в іншому потоці є операцією захоплення.

Для будь-якого об'єкта синхронізації S, якщо підобчислення $\mathrm{L}(\mathrm{t}_1)[\alpha]$ виконує release(S), а підобчислення $\mathrm{L}(\mathrm{t}_2)[\beta]$ виконує acquire(S), то виконується відношення порядку виконання $\mathrm{L}(\mathrm{t}_1)[\alpha] \rightarrow \mathrm{L}(\mathrm{t}_2)[\beta]$.

Це відношення також включає внутрішньопотокове впорядкування: якщо $t \ \ 1 = t_2$ і $\beta = \alpha + 1$, то також виконується:

$$L(t_1)[\alpha] \rightarrow L(t_1)[\beta].$$

Таким чином, ребра $E_{ctrl} \subseteq E$ задають частковий порядок над V, сформований як об'єднання контролю потоку та подій синхронізації.

Залежності від даних визначаються через множини читання та запису для кожного підобчислення.

Для підобчислення $L(t)[\alpha]$ множини $\mathrm{R(L(t)[\alpha])} \subseteq \mathrm{A}$ та $W(L(t)[\alpha]) \subseteq A$ відображають відповідно адреси, з яких читалося та до яких записувалося під час його виконання, де A - простір адрес пам’яті.

Формально, якщо існують два підобчислення $L(t_1)[\alpha]$та $L(t_2)[\beta]$, для яких виконується умова

$$W(L(t_1)[\alpha]) \cap R(L(t_2)[\beta]) \neq \emptyset \text{ і } L(t_1)[\alpha] \rightarrow L(t_2)[\beta],$$

то встановлюється ребро залежності від даних $L(t_1)[\alpha] \rightarrow_{data} L(t_2)[\beta]$, що приєднується до множини $E_{data} \subseteq E$.

Таким чином, повна множина ребер $E = E_{ctrl} \cup E_{data}$ визначає всі залежності між підобчисленняами, забезпечуючи точне моделювання причинно-наслідкових зв'язків у паралельному виконанні програмного забезпечення.

Це дозволяє використовувати CDDG як основу для інкрементального переобчислення: зміни у вхідних даних викликають інвалідацію лише тих підобчислень, які мають залежність від змінених даних або від інвалідацій, що до них ведуть.

Метод підтримує два основні типи виконання:

1) первинне (ініціалізаційне) виконання, що створює граф CDDG з нуля;

2) інкрементальне виконання, яке поширює зміни через уже побудований граф.

Під час первинного запуску виконується інструментоване трасування виконання програми, у межах якого кожному потоку t відповідає послідовність підобчислень $L(t) = \langle L(t)[0], L(t)[1], \dots \rangle$.

Для кожного підобчисленняа L(t)[α] під час виконання фіксується його множина читання R(L(t)[α]) та множина запису W(L(t)[α]), а також точки синхронізації, які визначають його взаємозв’язки з іншими підобчисленнями.

На основі цих даних побудова графа CDDG відбувається за правилами, наведеними в попередньому розділі: додаються ребра залежності від даних та ребра порядку виконання.

Формально, будується граф G=(V,E), де кожна вершина $v \in V$ відповідає одному підобчислення, а ребра $e \in E$ кодують причинно-наслідкові залежності.

Інкрементальне виконання використовує вже побудований граф G як основу для поширення змін.

Нехай Δ⊆A - множина адрес, які зазнали змін у вхідних даних.

Алгоритм інвалідації визначає підмножину підобчислень I⊆V, які потенційно залежать від адрес у Δ, тобто

$$I = \{L(t)[\alpha] \in V \mid R(L(t)[\alpha]) \cap \Delta \neq \emptyset\}.$$

Залежно від структури графа, ці інвалідації можуть поширюватися рекурсивно через ребра залежностей. Формально, виконується замикання по графу:

$$\text{Invalidated(I)}=\text{TransitiveClosureG(I)},$$

де $\text{TransitiveClosure}_{\text{G}}(\text{I})$ - це найменша множина підобчисленнь, яка включає I та всі підобчислення, до яких веде шлях з будь-якого $v \in I$ по ребрах E.

Після інвалідації відповідних підобчислень, обчислення перезапускається тільки для підобчислень із множини Invalidated(I), зберігаючи решту результатів повністю незмінними.

Таким чином, інкрементальне виконання дозволяє уникнути повного переобчислення програми при незначних змінах у вхідних даних, що є ключовою перевагою моделі СПЗ.

3.6 Формальна модель початкового запуску

Під час початкового запуску виконується трасування багатопотокової програми з метою побудови конкурентного детермінованого графа залежностей (КДГЗ).

Нехай $T = \{t_1, t_2, \dots, t_n\}$ - множина потоків, що виконуються повністю паралельно.

Кожен потік $t \in T$ виконує послідовність підобчислень

$$L(t)=\langle L(t)[0],L(t)[1],\dots \rangle,$$

де кожне підобчислення визначається як відрізок виконання між двома викликами примітивів синхронізації.

Виконання програми супроводжується динамічним аналізом пам’яті, при якому кожна інструкція доступу до пам’яті (load/store) обробляється процедурою onMemoryAccess(a,op), де a - адреса доступу, а $op \in \{load, store\}$.

Ці операції додають адресу aa до множини читання R(L(t)[α]) або запису W(L(t)[α]) поточного підобчислення L(t)[α].

Початок кожного підобчислення ініціюється процедурою startThunk(t), яка оновлює векторний годинник потоку $C_t \in N^{|T|}$ таким чином:

$$C_t[t] := \alpha,$$

де α - поточний порядковий номер підобчислення потоку t.

Цей векторний годинник відображає логічний час потоку та записується у структуру $C(L(t)[\alpha]) := C_t$ як часовий штамп підобчислення.

Завершення підобчислення реєструється процедурою endThunk(t), яка фіксує поточні множини читання/запису та забезпечує завершення спостереження за пам’яттю до наступної точки синхронізації.

Синхронізація між потоками виконується за допомогою механізму векторних годинників.

Кожен об’єкт синхронізації $s \in S$ асоціюється з вектором $C_S \in N^{|T|}$, який представляє його логічний час.

Для підтримки часткового порядку подій «відбулося до» ($\prec$), ми визначаємо правила оновлення годинників.

При виклику операції звільнення потоком t над об’єктом S, оновлюється:

$$C_S := max\ (C_S, C_t),$$

де максимум визначається поелементно: $max\ (u, v)[i] = max\ (u[i], v[i])$.

При виклику операції отримання об’єкта S потоком t, оновлюється:

$$C_t := max\ (C_t, C_S),$$

що гарантує, що всі події в потоці tt, які відбуваються після захоплення SS, будуть логічно пізнішими за будь-який попередній звільняючий підобчислення іншого потоку.

Таким чином, зв’язки «happens-before» між підобчисленнями формалізуються через відношення:

$$L(t_1)[\alpha] \prec L(t_2)[\beta] \Leftrightarrow C(L(t_1)[\alpha]) < C(L(t_2)[\beta]),$$

де оператор «<» означає поелементне домінування:

$$\forall i, C(L(t_1)[\alpha])[i] \leq C(L(t_2)[\beta])[i],$$
$$\exists j, C(L(t_1)[\alpha])[j] < C(L(t_2)[\beta])[j].$$

Після завершення виконання всіх потоків і фіксації відповідних множин R, W, а також векторних годинників для кожного підобчислення, будується орієнтований ациклічний граф $G = (V, E)$ - множина підобчислень, а ребра E створюються згідно з правилами залежностей за даними та синхронізаційного порядку.

Отриманий граф G використовується для подальшого інкрементального виконання, яке спирається на зафіксовану структуру залежностей для визначення повторно виконуваних підобчислень у разі зміни вхідних даних.

3.7 Формальна модель інкрементного прогону

Інкрементний прогін спрямований на повторне використання результатів попереднього виконання для прискорення обчислень при зміні використовуваних вхідних даних.

Нехай $L = \{L(t) \mid t \in T\}$ - множина всіх потоків, які містять послідовності підобчислень, зібрані під час початкового запуску, а M⊆A - брудна множина адрес пам’яті, що була модифікована з часу попереднього виконання.

Метою інкрементного прогону є повторне обчислення лише тих підобчислень, які прямо чи опосередковано залежать від M, а також оновлення графа залежностей GG, що відображає актуальний стан виконання.

Кожен потік tt перебирає свою послідовність L(t)=⟨L(t)[0],L(t)[1],… ⟩, і для кожного підобчислення L(t)[α] перевіряється умова повторного використання:

$$\exists a \in M \text{ та } a \in R(L(t)[\alpha]) \cup W(L(t)[\alpha]) \Rightarrow L(t)[\alpha] \text{ позначається як } dirty$$

Підобчислення, позначені як dirty, переобчислюються, а всі підобчислення, які мають залежності від них за графом G, рекурсивно анулюються.

Проте існує низка практичних аспектів, які потребують додаткових уточнень у моделі.

3.8 Врахування пропущених записів

Під час переобчислення підобчислення L(t)[α], може статись, що потік більше не записує в частину адрес, які були частиною $W_{old}(L(t)[\alpha])$, через зміну потоку керування (наприклад, умовне розгалуження). Тому ми визначаємо пропущену множину записів як:

$$\Delta W(L(t)[\alpha]) := W_{old}(L(t)[\alpha]) \setminus W_{new}(L(t)[\alpha]),$$

де W_{old} та W_{new} — відповідно старий та оновлений набори запису для підобчислення. Усі адреси з ΔW додаються до брудної множини M, щоб забезпечити повторне обчислення підобчислень, які залежать від них.

3.9 Внутрішньопотокові залежності та стекові ефекти

Оскільки точне трасування доступів до стеку є складним через його природу push/pop, та неефективність засобів ОС у відстеженні змін локальних змінних, у нашому підході стекові залежності моделюються консервативно.

Якщо підобчислення L(t)[α] позначено як dirty, тоді всі наступні підобчислення у потоці також анулюються:

$$\forall \beta > \alpha,\ L(t)[\beta] \Rightarrow \text{invalidate}.$$

Це гарантує коректне розповсюдження потенційних ефектів через локальні змінні без накладних витрат на їх явне трасування.

Під час інкрементного прогону може відбуватись дивергенція потоку керування, що призводить до появи нових потоків або зникнення існуючих.

Щоб підтримувати консистентність, ми застосовуємо стратегію повторного використання префікса: кожен потік t проходить свою стару трасу L(t)L(t), порівнюючи поточне виконання з попереднім у відповідності до детермінованого автомата станів.

Формально, для потоку t підтримується автомат:

$$A_t = (Q, \delta, q_0),$$

де Q - множина станів, δ - функція переходу, яка приймає рішення про:

- повторне використання підобчислення;
- переобчислення підобчислення;
- анулювання залишку траси потоку.

Це дозволяє відновити послідовність підобчислень з можливістю адаптації до змін у керуванні та даних.

Інкрементний прогін забезпечує ефективне переобчислення залежних фрагментів виконання програми, зберігаючи частковий порядок підобчислень

через механізм векторних годинників, і використовуючи стратегії фіксації пропущених змін, збереження консистентності стеку, а також гнучке пристосування до змін у структурі потоків.

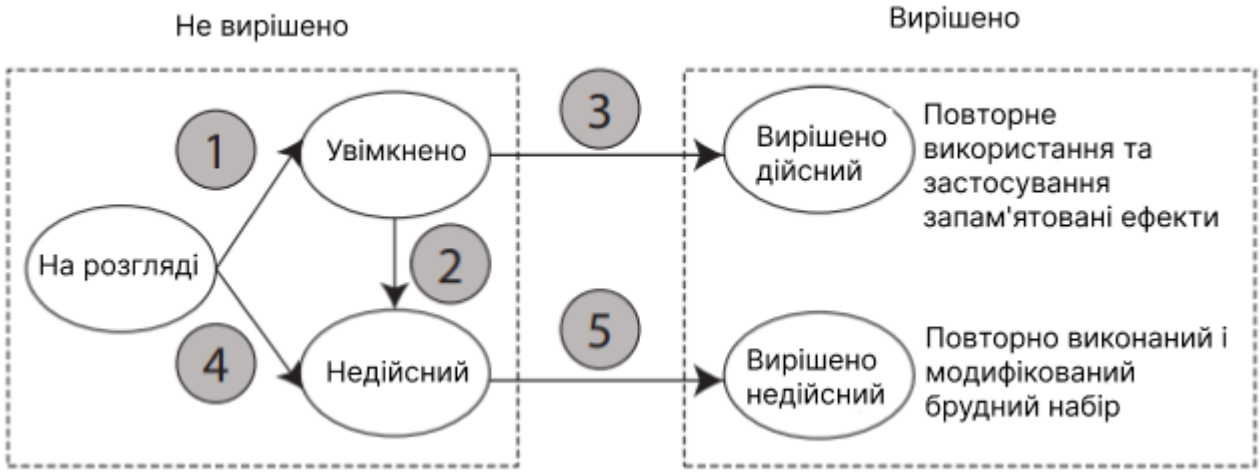


Рисунок 3.1 – Перехід стану для тунців під час інкрементного прогону

Ось як опис алгоритму інкрементного запуску можна представити у вигляді наукової частини з математичними позначеннями та формальними умовами:

Алгоритм інкрементного запуску дозволяє всім потокам працювати паралельно, зберігаючи взаємозв’язок між станами підобчислень кожного потоку.

Кожен потік може продовжувати виконання тільки після того, як всі попередні підобчислення, які відбулись перед ним, будуть вирішені (тобто повторно використані або переобчислені).

Це забезпечує коректне і послідовне оновлення графа залежностей G, що представляє систему.

Кожне підобчислення L(t)[α] знаходиться в одному з трьох основних станів: очікує, увімкнений або недійсний.

Початково стан усіх підобчислень потоку t встановлюється в очікує, за винятком підобчисленняа L(t)[0], який вже вважається увімкненим (enabled).

Під час виконання алгоритму підобчислення переходять між цими станами на основі умов, визначених графом залежностей та векторними годинниками.

Для того щоб підобчислення L(t)[α] став увімкненим, усі потоки повинні пройти через точку, вказану в годиннику підобчислень.

В роботі було використано векторні годинники C(t) для кожного потоку t, щоб визначити причинно-наслідкові зв'язки.

Якщо для будь-якого підобчислення $L(t_1)[\alpha_1]$ і $\setminus L(t_2)[\alpha_2]$ виконуються умови:

$$C(t_1)[\alpha_1] < C(t_2)[\alpha_2], \text{ то } L(t_1)[\alpha_1] \rightarrow L(t_2)[\alpha_2],$$

то $L(t_2)[\alpha_2]$ може бути увімкненим лише після того, як всі попередні підобчислення $L(t_1)[\alpha_1]$, що знаходяться в причинному порядку, будуть виконані.

Підобчислення переходить до стану resolved-valid, якщо:

$$\Delta W(L(t)[\alpha]) \cap M = \emptyset,$$

де $\Delta W(L(t)[\alpha])$ - множина пропущених записів для підобчисленняа $L(t)[\alpha]$,

M - брудна множина даних.

Якщо ж перетин не порожній, підобчислення переходить до стану invalid.

В випадку, коли підобчислення можна позначити як resolved-valid, ми не виконуватимемо підобчислення повторно, а безпосередньо застосуємо запам'ятовану множину запису до адресного простору, включаючи всі операції синхронізації:

$$resolveValid(L(t)[\alpha]) \text{де } L(t)[\alpha].W \rightarrow Memory.$$

Якщо підобчислення $L(t)[\alpha]$ був позначений як invalid, то він переходить до стану resolved-invalid, коли потік повторно виконує підобчислення, додаючи нові записи до брудної множини M, включаючи пропущені записи.

Підобчислення буде помічений як resolved-invalid, і повторне виконання триває для всіх підобчислень цього потоку, поки не буде досягнута кінцева точка:

$$\forall L(t)[\,\alpha], L(t)[\,\alpha]\ is\ resolved - invalid \Rightarrow \mathrm{re} - executeL(t)[\,\alpha]$$

Цей процес виконання підобчислень повторюється до завершення потоку \(t \), коли всі підобчислення будуть відзначені як resolved-invalid.

Підобчислення, що чекає на виконання, може перейти в стан invalid, якщо один з попередніх підобчислень цього ж потоку має стан invalid або resolved-invalid:

$$L(t)[\,\alpha]\ is\ invalid\ if\ \exists L(t)[\,\beta]\ such\ that\ \beta < \alpha\ and\ L(t)[\,\beta] \in \{invalid, resolved - invalid\}$$

Цей перехід дозволяє гарантувати, що жодні не підтверджені зміни не будуть використовуватись до завершення потоку.

Після виконання всіх підобчислень алгоритм оновлює граф залежностей \(G \), додаючи нові підобчислення, а також зберігає поточні стани всіх підобчислень для наступного інкрементного прогону:

$$\mathcal{L} := \ L(t)[\,\alpha] \mid t \in T, \alpha \in A\ \}.$$

Метод гарантує, що кожне підобчислення буде повторно використаний або повторно виконаний у відповідності до змін, що виникають в брудній множині M.

Цей підхід забезпечує ефективне оновлення системи з мінімальними витратами, дозволяючи виконувати лише необхідні перерахунки та зберігати консистентність виконання для всіх потоків.

3.8 Багатопотокове інкрементальне виконання у мобільних пристроях із використанням хмарних технологій

Нехай мобільний пристрій представлений як система M, що складається з множини локальних обчислювальних ресурсів R_M та набору процесів P_M які виконуються на пристрої.

Хмарне середовище позначимо як C, що включає обчислювальні ресурси R_c та набір віртуальних машин (або контейнерів) V_C.

Функціональний розподіл виконання між мобільним пристроєм і хмарою визначається динамічним розподілом навантаження, що мінімізує загальні витрати на виконання:

$$F: P_M \to \{R_M, R_C\}$$

де функція F визначає, які процеси повинні виконуватись локально на пристрої, а які слід передавати у хмару.

Нехай програма, що виконується, представлена у вигляді множини потоків:

$$T = \{t_1, t_2, \dots, t_n\}$$

кожен потік t_i складається з послідовності підобчислень:

$$L(t_i) = \langle L(t_i)[0], L(t_i)[1], \dots, L(t_i)[k] \rangle$$

Для кожного потоку ми визначаємо часові обмеження $D(t_i)$, які характеризують максимальний допустимий час виконання потоку t_i.

Для визначення того, чи слід виконувати підобчислення локально або у хмарі, ми використовуємо адаптивну функцію розподілу навантаження:

$$H(L(t_i)[\alpha]) = \begin{cases} R_M, \text{якщо } C_M(L(t_i)[\alpha]) + T_M < D(t_i) \\ R_C, \text{якщо } C_C(L(t_i)[\alpha]) + T_C < D(t_i) \text{ і } C_C(L(t_i)[\alpha]) < C_M(L(t_i)[\alpha]) \end{cases}$$

де:

- $C_M(L(t_i)[\alpha])$ — локальна обчислювальна вартість підобчислення,

- $C_C(L(t_i)[\alpha])$ — обчислювальна вартість у хмарі,

- T_M, T_C — затримки на передачу даних між мобільним пристроєм і хмарою.

Таким чином, якщо виконання у хмарі є швидшим і не перевищує допустимі часові обмеження, підобчислення передається на обчислення у хмару.

Під час початкового виконання будується хмарний граф залежностей G_C , який зберігається у хмарному середовищі та використовується для інкрементного оновлення:

$$G_C = (V, E)$$

де:

- $V = \{L(t_i)[\alpha]\}$— множина підобчислень,

- $E = E_C \cup E_D$, де E_C - ребра контролю,

E_D — ребра залежностей від даних.

Під час повторного запуску програми на мобільному пристрої відбувається перевірка брудної множини змінених даних M, яка визначається як:

$$M = \{a \in A \mid a \text{ змінено з попереднього запуску}\}.$$

Далі у хмарі виконується розповсюдження змін, оновлюючи лише підобчислення, що залежать від змінених даних:

$$\forall L(t_i)[\alpha] \in V, \text{якщо } W(L(t_i)[\alpha]) \cap M \neq \emptyset, \text{ тоді } L(t_i)[\alpha] \text{ перевиконується.}$$

Оновлені результати передаються назад на мобільний пристрій лише для змінених підобчислень, що мінімізує обсяг переданих даних та підвищує ефективність виконання.

Для узгодження між потоками при розподіленому виконанні використовуються векторні годинники C_t, що зберігають частковий порядок виконання:

$$C_t = \{C_t[1], C_t[2], \dots, C_t[T]\}$$

Кожне підобчислення у хмарі має власний векторний годинник:

$$C(L(t_i)[\alpha]) = max(C_t, C_C)$$

При передачі виконання між мобільним пристроєм та хмарою, потік успадковує найбільші значення векторних годинників, що забезпечує коректне впорядкування:

$$C_M = max(C_M, C_C)$$

Таким чином, мобільний пристрій завжди отримує консистентну та оновлену інформацію без потреби в глобальному бар'єрі синхронізації.

Запропонована модель дозволяє ефективно розподіляти виконання між мобільним пристроєм та хмарою, забезпечуючи мінімальні затримки та оптимальне використання обчислювальних ресурсів.

Інкрементне виконання значно зменшує обчислювальні витрати як на мобільному пристрої, так і у хмарі, а застосування векторних годинників забезпечує узгоджене оновлення залежностей між потоками.

3.10 Висновки до третього розділу

У розділі подано зроблений метод здійснення високопродуктивних обчислень для мобільних пристроїв із використанням хмарних технологій, що базується на методах інкрементального багатопотокового виконання. Основою

системи слугує конкурентний динамічний граф залежностей (КДГЗ), який дозволяє ефективно відстежувати та повторно використовувати результати попередніх обчислень у разі незначних змін вхідних даних.

Запропонований метод дозволяє:

- зменшити обсяг повторного обчислення шляхом локалізації змін у графі залежностей;

- забезпечити узгоджене виконання потоків через механізм векторних годинників;

- ефективно розподіляти обчислювальне навантаження між мобільними пристроями та хмарною інфраструктурою;

- знизити затрати на комунікацію та обчислення при збереженні семантичної коректності виконання.

В дослідженні використано моделі пам’яті Release Consistency, що дозволяє уникати зайвих обчислювальних накладних витрат при збереженні міжпотокової узгодженості.

Система також підтримує адаптивне планування задач з урахуванням енергетичних та часових обмежень, що критично важливо для мобільних обчислювальних платформ.

4 РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЕНЬ НА МОБІЛЬНИХ ПРИСТРОЯХ ІЗ ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

4.1 Системне програмне забезпечення системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Було реалізовано системне програмне забезпечення (СПЗ) системи для високопродуктивних обчислень як 32-розрядну динамічно компоновану бібліотеку для ОС Android на ядрі GNU/Linux (рис. 4.1), що повторно використовує два механізми реалізації: підсистему пам'яті та аллокатор пам'яті.

Програмна реалізація розпаралелення виконання потоків була реалізована засобами конкурентного динамічного графа залежностей для оптимізації рзопаралелених обчислень мовою C++ (через Android NDK). Доступ до низькорівневої синхронізації та паралельних алгоритмів було здійснено заслбами std::thread, std::mutex, std::atomic або OpenMP/TBB) (Додаток В).

Реалізація включає в себе також і запам'ятовувач.

Розглянемо системне програмне забезпечення системи для високопродуктивних обчислень на мобільних пристроях.

Рисунок 4.1 - Архітектура і системного програмного забезпечення системи для високопродуктивних обчислень

Реалізація застосування хмарної інфраструктури для збереження результатів роботи високопродуктивних обчислень на мобільних пристроях Android передбачала інтеграцію технологічних компонентів і організацію взаємодії між пристроєм користувача, обчислювальними процесами та хмарним середовищем.

На етапі початкової ініціалізації мобільний додаток на Android налаштовував захищене з'єднання з хмарною платформою за допомогою відповідних API, таких як REST або gRPC. При цьому використовувалися механізми автентифікації, OAuth 2.0. Після встановлення з'єднання мобільний пристрій міг надсилати дані до хмарної інфраструктури. Після завершення обчислень результати зберігалися шляхом серіалізації у компактний формат, наприклад JSON або Protocol Buffers, що мінімізує об'єм передаваних даних та підвищує швидкість обміну.

Збереження результатів у хмарі зазвичай було реалізовано через сервіс Google Cloud на основі OpenStack Swift. Мобільний додаток ініціював запит на створення об'єкта в обраному сховищі, прикріплюючи до нього необхідні метадані: час створення, ідентифікатор сесії, тип обчислень та інші параметри для подальшої ідентифікації або пошуку. Було передбачено механізми повторної передачі у випадку помилок мережі або тимчасової втрати з'єднання, що реалізуються через локальні черги повідомлень або кешування результатів у внутрішній базі даних пристрою, наприклад, через Room або SQLite. Після успішного завантаження даних у хмару локальні копії можуть видалятися або зберігатися відповідно до політики управління пам'яттю застосунку.

Зворотний процес отримання даних із хмари також оптимізується за рахунок використання часткових запитів, що дозволяє завантажувати лише необхідні частини великих результатів. Для цього мобільний додаток може використовувати спеціальні запити із вказанням діапазонів байтів або фільтрацією за допомогою серверних функцій, які підтримує обране хмарне рішення.

4.1.1 Реалізація підсистеми пам'яті

Підсистема пам'яті системне програмне забезпечення системи для високопродуктивних обчислень реалізує RC-модель пам'яті та виводить набори для читання/запису на кожен потік.

Для реалізації RC-моделі пам'яті СПЗ перетворює потоки в окремі процеси за допомогою раніше запропонованого механізму.

Цей підхід «потік-як-процес» надає кожному потоку власний приватний адресний простір, і таким чином дозволяє системному програмному забезпеченню обмежити міжпотокову комунікацію.

На практиці, СПЗ розгалужує новий процес на pthread_create() і включає в себе механізм фіксації спільної пам'яті, який забезпечує зв'язок між процесами у точках синхронізації, як того вимагає модель пам'яті RC (рисунок 4.2).

На високому рівні, протягом виконання програми, системне програмне забезпечення системи для високопродуктивних обчислень підтримує копію вмісту адресного простору в (спільному) буфері посилань, і саме через цей буфер, з інструментарієм, що надається СПЗ в точках синхронізації, процеси прозоро взаємодіють.

Комунікація між процесами реалізується шляхом визначення набору запису.

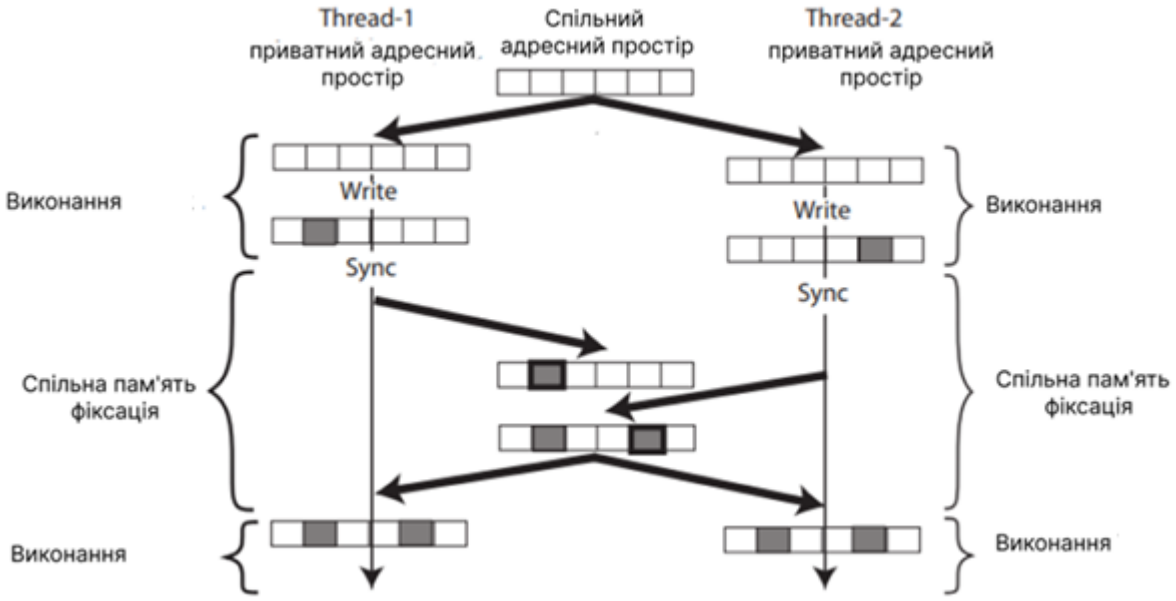


Рисунок 4.2 – Виконання задач в записувачеві

У випадку накладання записів до однієї і тієї ж ділянки пам'яті, зроблених різними процесами, це вирішує конфлікт, використовуючи політику перемоги останнього запису.

Крім того, з міркувань ефективності, реалізація механізму зв'язку покладається на приватні файли зіставлення пам'яті - це дозволяє різним процесам спільно використовувати фізичні сторінки до того моменту, як вони почнуть писати на них, і зберігає низькі накладні витрати на продуктивність завдяки механізму копіювання при записі в операційній системі.

Розглянемо набір для читання і запису. Окрім того, що він слугує основою для моделі пам'яті RC, прийнятий механізм «потік як процес» також є важливим для відстеження звернень до пам'яті: розбиваючи початковий багатопотоковий процес на декілька однопотокових процесів, СПЗ може легко отримувати набори для читання та запису для кожного потоку.

Оскільки кожен потік реалізовано як окремий процес, СПЗ використовує механізм захисту пам'яті ОС для відстеження наборів для читання та запису. Зокрема, СПЗ робить адресний простір недоступним, викликаючи функцію mprotect (PROT_NONE) на початку кожного потоку, що гарантує, що при першому читанні або записі сторінки потоком буде згенеровано сигнал.

Таким чином, у відповідному обробнику сигналу СПЗ може записувати місця доступу до пам'яті на рівні сторінок.

Одразу після запису доступу до пам'яті бібліотека СПЗ скидає біти захисту сторінок, що дозволяє підобчислення відновити операцію читання/запису, як тільки обробник сигналу повернеться.

Крім того, скидання дозволів на доступ до пам'яті також гарантує, що наступні звернення відбуватимуться без подальших помилок сторінки.

Таким чином, у СПЗ виникає щонайбільше дві помилки сторінки (одна для читання і одна для запису) для кожної сторінки, до якої здійснюється доступ.

4.1.2 Реалізація підсистеми запису і відтворення

Бібліотека реалізованого СПЗ виконує додаток у режимі запису або відтворення. Далі ми опишемо два підкомпоненти, записувач та відтворювач, які реалізують ці режими виконання.

Оскільки СПЗ повторно використовує підсистему пам'яті, яка послідовно виконує операції фіксації пам'яті з різних потоків, реалізація алгоритму запису значно спрощується. Завдяки неявній серіалізації меж потоків, що виникає у результаті границь каналів, тактові частоти потоків, каналів та вектора синхронізації ефективно зводяться до скалярних послідовних чисел, що дозволяє записувачу просто кодувати розклад потоків за допомогою послідовних чисел каналів.

Реєстратор також відповідає за запам'ятовування стану процесу в кінці кожного підобчисленняа. Для цього, використовуючи процедуру асемблера, СПЗ зберігає значення регістрів у стеку, робить знімок брудних сторінок в адресному просторі і зберігає знімок у пам'яті.

Крім того, реєстратор також зберігає КДГЗ, що складається з ідентифікаторів потоків (номер потоку та номер послідовності потоків) та відповідних наборів читання/запису, у зовнішній файл.

Подібно до записуючого пристрою, відтворювач покладається на номери послідовностей тактів для забезпечення дотримання записаного порядку розкладу виконання задач.

Спочатку відтворювач зчитує файл із вхідними змінами та КДГЗ для ініціалізації алгоритму відтворення.

Під час послідовного запуску, коли запам'ятовані фрагменти можуть бути використані повторно, програвач витягує відповідний стан із запам'ятовуючого пристрою, виправляє адресний простір та відновлює стан регістрів.

4.1.3 Реалізація підсистеми підтримки операційної системи

Оскільки практичні додатки залежать від сервісів ОС, є два важливих аспекти, пов'язаних з ОС, на які СПЗ має звернути увагу.

По-перше, системні виклики використовуються програмою для зв'язку з рештою системи, тому вплив системних викликів (на решту системи і на саму програму) потрібно враховувати; зокрема, потрібно обробляти зміни вхідних даних, зроблені користувачем.

По-друге, існують механізми операційної системи, які можуть без потреби змінювати структуру пам'яті програми під час запуску, запобігаючи повторному використанню запам'ятованих фрагментів.

Розглянемо системні виклики та зміни вводу. Оскільки СПЗ є бібліотекою користувацького простору, яка працює поверх незміненого ядра Linux, вона не має доступу до структур даних ядра.

Таким чином, наслідки системних викликів не можуть бути запам'ятовані або відтворені. Для підтримки системних викликів системних викликів, СПЗ розглядає системні виклики як роздільники потоків. Отже, безпосередньо перед системним викликом СПЗ запам'ятовує стан потоку, а одразу після повернення системного виклику СПЗ визначає, чи може він повторно використовувати наступні потоки відповідно до алгоритму відтворення.

Щоб гарантувати, що системні виклики набувають чинності (зовнішні та внутрішні), СПЗ викликає системні виклики у всіх виконаннях, навіть під час повторного запуску.

Щоб гарантувати, що вплив системних викликів на програму (тобто, значення, що повертаються, і записи, зроблені в адресний простір процесу) враховуються правилами валідації thunk, СПЗ індексує набір записів системних викликів і перевіряє, чи збігаються значення, що повертаються, з попередніми виконаннями, використовуючи знання їх семантики (наприклад, деякі параметри системних викликів параметри системних викликів є вказівниками, куди записуються дані).

Важливо, що для визначення множини запису системних викликів, які можуть зчитувати великі обсяги даних (наприклад, mmap), СПЗ дозволяє користувачеві вказувати зміни вхідних даних (які потенційно змінюють множину

запису цих системних викликів) явно за допомогою зовнішнього файлу, який містить діапазони зсувів байт.

На практиці реалізація перехоплює системні виклики через обгортки на рівні викликів бібліотеки glibc.

Щоб уникнути непотрібних залежностей даних між потоками, СПЗ повторно використовує кастомний розподільник пам'яті. Цей підхід ізолює запити на виділення та вивільнення пам'яті розподілу та перерозподілу пам'яті для кожного потоку, розбиваючи купу додатків на фіксовану кількість підтекстових куп. Це гарантує, що послідовність розподілів в одному потоці не впливає на розташування розподілів в іншому потоці, що в іншому випадку могло б викликати непотрібні повторні обчислення.

Крім того, СПЗ гарантує, що випадковий розподіл адресного простору, функція ОС, яка навмисно випадковим чином розподіляє пам'ять програми, буде вимкнена.

4.1.4 Реалізація підсистеми запам'ятовувача

Запам'ятовувач відповідає за збереження кінцевого стану кожного потоку, щоб його ефекти могли бути відтворені у наступних інкрементних запусках. Запам'ятовувач реалізовано у вигляді окремої програми, яка зберігає запам'ятований стан у сегменті пам'яті, що використовується як підкладка для реалізації сховища ключів-значень, доступного для записувача/відтворення. Запам'ятований стан зберігається в пам'яті для швидкого доступу та асинхронно реплікується.

4.2 Експерименти

Для виконання експериментів було використано таке обладнання: восьмиядерний процесор Snapdragon 8 Gen 3 (одне продуктивне ядро (Cortex-X4 з

тактовою частотою 3,2 ГГц), п'ять допоміжних (Cortex A720 на 3 ГГц) і всього два енергоефективних (Cortex-A520 на 2 ГГц).), 12 ГБ оперативної пам'яті.

Було використано тестові набори PARSEC - набір бенчмарків, розроблений для дослідження паралельних обчислень на багатоядерних процесорах.

PARSEC включає реалістичні сучасні додатки та алгоритми, які показують навантаження на процесори в практичних сценаріях (обробка зображень, фінансове моделювання, кодування відео тощо).

Для всіх експериментів кожен додаток було виконано 12 разів. Було виключено найнижчі та найвищі результати, а подано середнє значення за 10 запусків, що залишилися.

Було досліджено такі метрики як робота і час. Робота відноситься до загального обсягу обчислень, виконаних усіма потоками, і вимірюється як сума загального часу виконання всіх потоків. Час відноситься до наскрізного часу виконання, необхідний для завершення паралельних обчислень. Економія часу відображає зменшену затримку для кінцевого користувача, тоді як економія роботи відображає покращене загальне використання ресурсів.

Розглянемо результати приросту продуктивності. Для цього представимо порівняння інкрементного запуску СПЗ з pthreads, як показано на рисунку 4.3.

У цьому експерименті було змінено одну випадково вибрану сторінку вхідного файлу перед інкрементним запуском.

Потім обчислено роботу і час, необхідний для інкрементного запуску СПЗ, а також pthreads, які переобчислюють все з нуля.

Результати приросту продуктивності СПЗ по відношенню при інкрементному виконанні подано на рисунку 4.3.

Було показано прискорення роботи та часу (тобто продуктивність СПЗ, нормалізовану до продуктивності pthreads) для різної кількості потоків - від 12 до 64 потоків. При порівнянні продуктивності ми використовували однакову кількість потоків в СПЗ та pthreads. Експеримент показав, що переваги використання СПЗ значно відрізняються в різних додатках. У більш ніж половині з оцінених тестів (7 з 11) СПЗ змогла досягти принаймні 2-кратного прискорення часу. Загалом,

результати показують, що СПЗ є ефективною у широкому діапазоні тестових завдань, але також і те, що бібліотека не є універсальним рішенням.

Як і очікувалося, було виявлено, що збільшення кількості потоків, як правило, призводить до збільшення прискорення. Це пояснюється тим, що при фіксованому розмірі вхідних даних більша кількість потоків призводить до меншого обсягу роботи на один потік. В результаті, СПЗ змушений переобчислювати меншу кількість даних при зміні однієї вхідної сторінки.

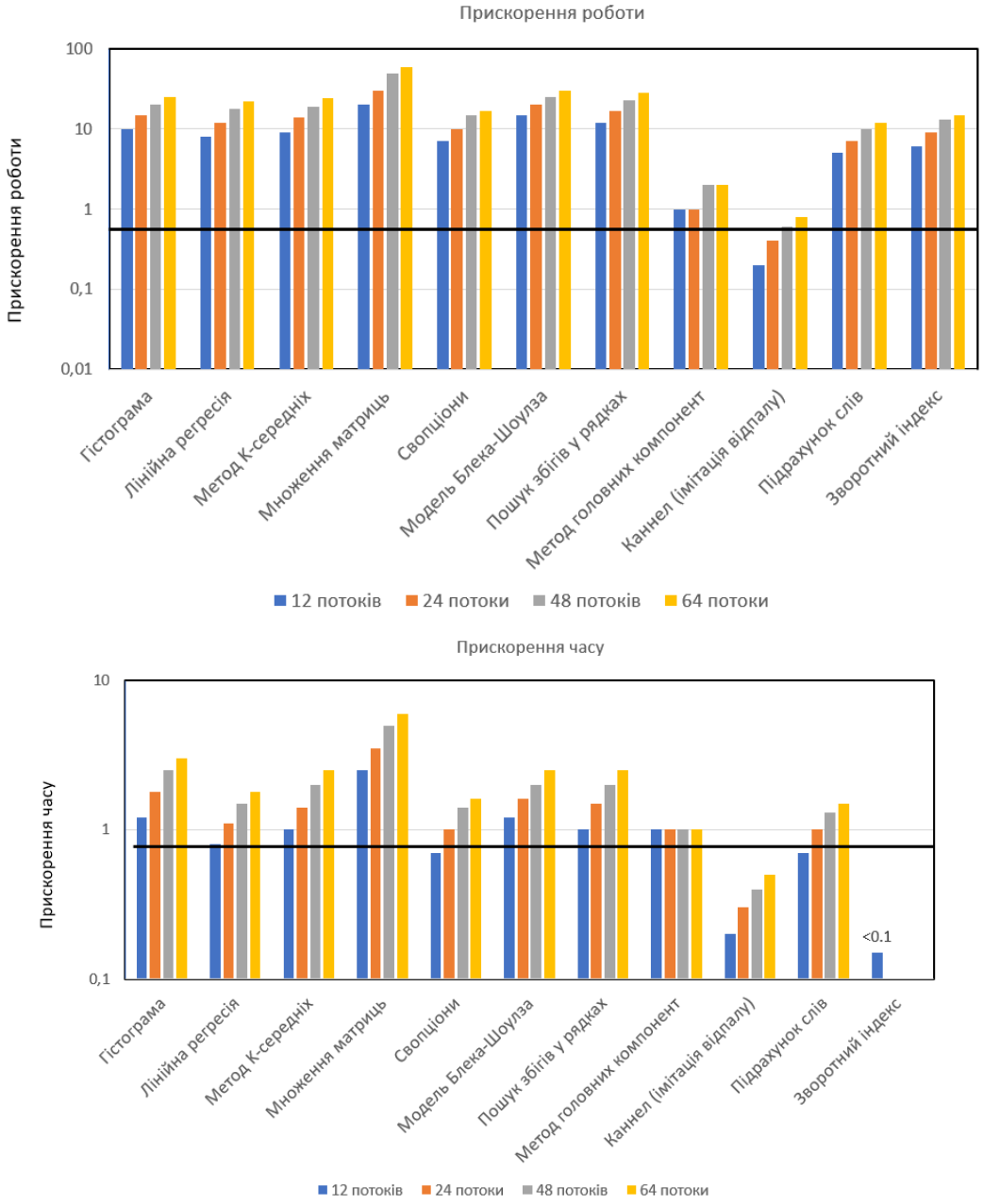


Рисунок 4.3 – Приріст продуктивності СПЗ по відношенню при інкрементному виконанні

Прискорення роботи не призводить безпосередньо до прискорення часу. Це пов'язано з тим, що навіть якщо зміни у вхідних даних впливають лише на один потік, час виконання наскрізного потоку все одно залежить від часу виконання (найповільнішого) недійсного потоку. Масштабованість за допомогою даних (прискорення роботи та часу) подано на рисункку 4.4

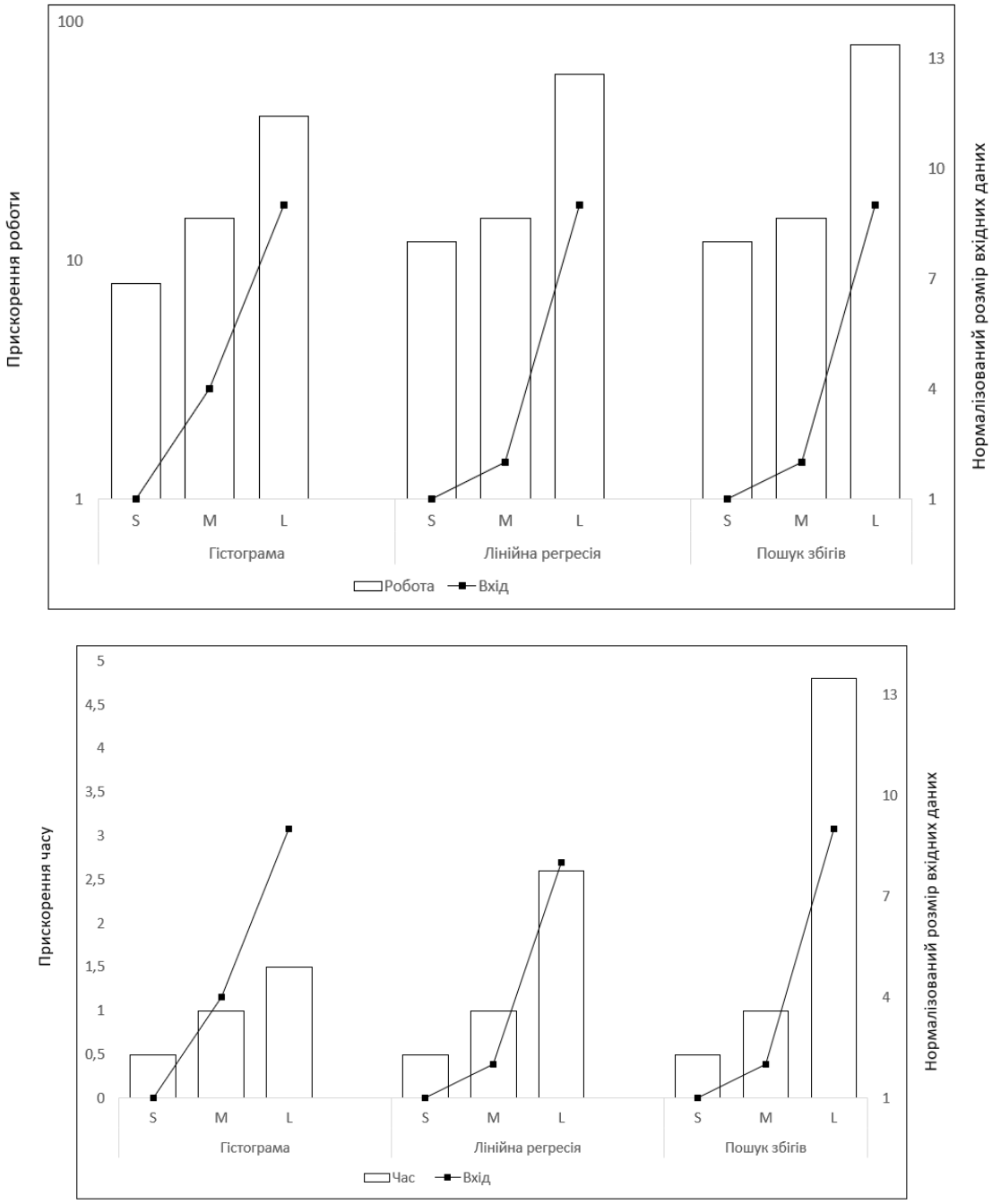


Рисунок 4.4. - Масштабованість за допомогою даних (прискорення роботи та часу)

4.3 Висновки

У розділі було здійснено реалізацію системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій. Розроблене системне програмне забезпечення (СПЗ) представлено у вигляді 32-розрядної динамічно компонованої бібліотеки для ОС Android на ядрі GNU/Linux. Архітектура СПЗ передбачає реалізацію підсистем пам’яті, запису й відтворення, підтримки операційної системи та запам’ятовувача.

Було реалізовано підсистему пам’яті на основі моделі пам'яті Release Consistency (RC), де потоки перетворюються в окремі процеси за допомогою механізму «потік-як-процес». Це дозволяє обмежити міжпотокову комунікацію, відстежувати доступ до пам'яті через обробку сторінкових помилок і забезпечувати прозору взаємодію процесів через буфер посилань. Реалізація також оптимізована для мінімізації накладних витрат завдяки використанню приватних файлів зіставлення пам'яті.

У підсистемі запису і відтворення було реалізовано механізм фіксації стану потоків із можливістю запам’ятовування і відновлення регістрів та сторінок пам’яті. Для збереження історії виконання використовується конкурентний динамічний граф залежностей (КДГЗ), що дозволяє ефективно управляти інкрементальним виконанням задач із мінімальним переобчисленням.

Підсистема підтримки операційної системи забезпечує правильне поводження із системними викликами та змінами у вхідних даних, перехоплюючи їх на рівні glibc. Реалізовано ізоляцію пам’яті потоків за допомогою кастомного аллокатора та відключення випадкового розподілу пам'яті для уникнення непередбачуваних залежностей.

Запам’ятовувач реалізовано як окрему програму зі збереженням стану в пам’яті та асинхронною реплікацією даних, що забезпечує швидкий доступ до збережених фрагментів виконання.

У рамках експериментальної оцінки було проведено випробування на наборі тестів PARSEC із використанням обладнання з процесором Intel Xeon і 32

ГБ оперативної пам’яті. Оцінка базувалася на метриках "робота" (загальний обсяг обчислень) та "час" (наскрізний час виконання). Результати експериментів продемонстрували значне прискорення виконання задач при використанні СПЗ порівняно з традиційним підходом pthreads, особливо при збільшенні кількості потоків. У більшості тестових випадків було досягнуто принаймні дворазове прискорення часу виконання.

Таким чином, запропоноване СПЗ показало свою ефективність для широкого класу практичних задач, проте результати також вказують на залежність ефективності від характеру застосованих додатків і необхідність подальшої оптимізації для специфічних сценаріїв.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено систему для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій.

У першому розділі проаналізовано високопродуктивні обчислення (HPC) та їх застосування на мобільних пристроях у поєднанні з хмарними технологіями. Встановлено, що HPC традиційно використовується у суперкомп'ютерах, однак сучасні тенденції спрямовані на мобільні пристрої з багатоядерними CPU, GPU та нейронними модулями. Використання хмарних сервісів дозволяє розширити обчислювальні можливості, знизити енергоспоживання та масштабувати ресурси без значних витрат на апаратне забезпечення. Виявлено основні недоліки існуючих рішень: обмежена обчислювальна потужність мобільних пристроїв, високе енергоспоживання та ризики перегріву, затримки через нестабільний інтернет, складність адаптації HPC-алгоритмів, ризики витоку даних та висока вартість хмарних сервісів. Це обґрунтовує необхідність подальших досліджень у цій сфері.

У другому розділі розроблено формалізовану модель системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій. Модель охоплює мобільні пристрої, хмарну інфраструктуру, адаптивне планування задач, безпеку, сховища даних та користувацькі інтерфейси. Формалізація через множини об’єктів і функцій забезпечує опис поведінки системи за змінних ресурсів і мережевих характеристик. Враховано вплив затримок і динамічного розподілу задач, що робить модель основою для подальшої розробки системи.

У третьому запропоновано метод високопродуктивних обчислень для мобільних пристроїв із використанням хмарних технологій, заснований на інкрементальному багатопотоковому виконанні та конкурентному динамічному графі залежностей (КДГЗ). Метод дозволяє мінімізувати повторні обчислення, узгоджено виконувати потоки через векторні годинники, ефективно розподіляти навантаження та знижувати комунікаційні витрати. Використання моделі пам’яті

Release Consistency і адаптивного планування задач забезпечує енергоефективність та часову оптимізацію для мобільних платформ.

У четвертому реалізовано систему для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій. Розроблене системне програмне забезпечення у вигляді бібліотеки для Android включає підсистеми пам’яті, запису і відтворення, підтримки операційної системи та запам’ятовувач. Застосовано модель пам’яті Release Consistency і механізм «потік-як-процес» для підвищення ефективності обробки пам'яті та мінімізації накладних витрат.

У підсистемах реалізовано фіксацію стану потоків через конкурентний динамічний граф залежностей, правильну обробку системних викликів і ізоляцію пам’яті потоків. Запам’ятовувач забезпечує швидкий доступ до збережених станів. Експериментальні дослідження на наборі тестів PARSEC підтвердили значне прискорення виконання задач порівняно з pthreads, особливо зі зростанням кількості потоків.

Таким чином, запропоноване СПЗ продемонструвало ефективність, проте потребує подальшої оптимізації для окремих типів застосувань.

За темою кваліфікаційної роботи опубліковано одну публікацію [93] у Збірнику наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2024». (Хмельницький 2024. С. 303-305).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Kang P. Programming for High-Performance Computing on Edge Accelerators. *Mathematics*. 2023. 11(4). 1055.

2. Guidi G., Ellis M., Buluç A., Yelick K., Culler D. 10 Years Later: Cloud Computing is Closing the Performance Gap. *In Proceedings of the Companion of the ACM/SPEC International Conference on Performance Engineering*. 2021. pp. 41–48.

3. Reed D., Gannon D., Dongarra J. Reinventing High Performance Computing: Challenges and Opportunities. *arXiv*. 2022. arXiv:2203.02544.

4. Shi W., Cao J., Zhang Q., Li Y., Xu L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* 2016. 3. 637–646.

5. Varghese B., Wang N., Barbhuiya S., Kilpatrick P., Nikolopoulos D.S. Challenges and Opportunities in Edge Computing. *In Proceedings of the 2016 IEEE International Conference on Smart Cloud (SmartCloud)*. 2016. pp. 20–26.

6. Cao K., Liu Y., Meng G., Sun Q. An Overview on Edge Computing Research. *IEEE Access*. 2020. 8. 85714–85728.

7. Saraf P.D., Bartere M.M., Lokulwar P.P. A Review on Evolution of Architectures, Services, and Applications in Computing towards Edge Computing. *In International Conference on Innovative Computing and Communications*. 2022. pp. 733–744.

8. Karumbunathan L. Solving Entry-Level Edge AI Challenges with NVIDIA Jetson Orin Nano. 2022. Available online: https://developer.nvidia.com/blog/solving-entry-level-edge-ai-challenges-with-nvidia-jetson-orin-nano (дата доступу 24.12.2024).

9. Li E., Zeng L., Zhou Z., Chen X. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Trans. Wirel. Commun.* 2020. 19. 447–457.

10. Chen J., Ran X. Deep Learning With Edge Computing: A Review. *Proc. IEEE*. 2019. 107. 1655–1674.

11. Riha L., Le Moigne J., El-Ghazawi T. Optimization of Selected Remote Sensing Algorithms for Many-Core Architectures. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* 2016. 9. 5576–5587.

12. Tu W., Pop F., Jia W., Wu J., Iacono M. High-Performance Computing in Edge Computing Networks. *J. Parallel Distrib. Comput.* 2019. 123. 230.

13. Cecilia J.M., Cano J.C., Morales-Garcia J., Llanes A., Imbernon B. Evaluation of Clustering Algorithms on GPU-Based Edge Computing Platforms. *Sensors*. 2020. 20. 6335.

14. Poulos A., McKee S.A., Calhoun J.C. Posits and the State of Numerical Representations in the Age of Exascale and Edge Computing. *Softw. Pract. Exp.* 2022. 52. 619–635.

15. Zamora-Izquierdo M.A., Santa J., Martínez J.A., Martínez V., Skarmeta A.F. Smart Farming IoT Platform based on Edge and Cloud Computing. *Biosyst. Eng.* 2019. 177. 4–17.

16. Kalyani Y., Collier R. A Systematic Survey on the Role of Cloud, Fog, and Edge Computing Combination in Smart Agriculture. *Sensors*. 2021. 21. 5922.

17. Liu S., Liu L., Tang J., Yu B., Wang Y., Shi W. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE*. 2019. 107. 1697–1716.

18. Facing the Edge Data Challenge with HPC + AI. 2022. Available online: https://developer.nvidia.com/blog/facing-the-edge-data-challenge-with-hpc-ai (дата доступу 25 лютого 2025).

19. Stone J.E., Gohara D., Shi G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Comput. Sci. Eng.* 2010. 12. 66–73.

20. Nickolls J., Buck I., Garland M., Skadron K. Scalable Parallel Programming with CUDA. *Queue*. 2008. 6. 40–53.

21. Li B. Dong W. EdgeProg: Edge-centric Programming for IoT Applications. *Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. 2020. C. 212–222.

22. IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*. 2008. C. 1–70.

23. Li W. Jin G. Cui X. See S. An Evaluation of Unified Memory Technology on NVIDIA GPUs. *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2015. С. 1092–1098.

24. Jarząbek Z. Czarnul P. Performance Evaluation of Unified Memory and Dynamic Parallelism for Selected Parallel CUDA Applications. *Journal of Supercomputing*. 2017. № 73. С. 5378–5401.

25. Choi J. You H. Kim C. Young Yeom H. Kim Y. Comparing Unified, Pinned, and Host/Device Memory Allocations for Memory-intensive Workloads on Tegra SoC. *Concurrency and Computation: Practice and Experience*. 2021. № 33. E6018.

26. Allen T. Ge R. In-Depth Analyses of Unified Virtual Memory System for GPU Accelerated Computing. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021.

27. NVidia. Accelerated Linux Graphics Driver README and Installation Guide Chapter 44. Open Linux Kernel Modules. 2022. Доступно онлайн: http://download.nvidia.com/XFree86/Linux-x86_64/515.43.04/README/kernel_open.html (дата доступу 25 лютого 2025).

28. NVidia. NVIDIA Linux Open GPU Kernel Module Source. 2022. Доступно онлайн: https://github.com/NVIDIA/open-gpu-kernel-modules (дата доступу 25 лютого 2025).

29. Khronos OpenCL Working Group. The OpenCL Specification Version v3.0.12. 2022. Доступно онлайн: https://registry.khronos.org/OpenCL/specs/3.0-unified/pdf/OpenCL_API.pdf (дата доступу 25 лютого 2025).

30. Cavicchioli R. Capodieci N. Bertogna M. Memory Interference Characterization between CPU Cores and Integrated GPUs in Mixed-Criticality Platforms. *Proceedings of the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017. С. 1–10.

31. Portable Computing Language (PoCL). Доступно онлайн: http://portablecl.org (дата доступу 25 лютого 2025).

32. Khronos OpenCL Working Group. SYCL 1.2.1 Specification. 2022. Доступно онлайн: https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf (дата доступу 25 лютого 2025).

33. Burns R. Davidson C. Dodds A. Enabling OpenCL and SYCL for RISC-V Processors. *Proceedings of the International Workshop on OpenCL*. 2021.

34. Asanović K. Patterson D.A. Instruction Sets Should Be Free: The Case For RISC-V. *Technical Report UCB/EECS-2014-146*. 2014.

35. Reddy Kuncham G.K. Vaidya R. Barve M. Performance Study of GPU applications using SYCL and CUDA on Tesla V100 GPU. *Proceedings of the 2021 IEEE High Performance Extreme Computing Conference (HPEC)*. 2021. С. 1–7.

36. Intel DPC++ SYCL for CUDA User Manual. Доступно онлайн: https://github.com/intel/llvm/blob/sycl/sycl/doc/UsersManual.md (дата доступу 25 лютого 2025).

37. Dagum L. Menon R. OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*. 1998. № 5. С. 46–55.

38. Chapman B. Huang L. Biscondi E. Stotzer E. Shrivastava A. Gatherer A. Implementing OpenMP on a High Performance Embedded Multicore MPSoC. *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*. 2009. С. 1–8.

39. Liang T.Y. Li H.F. Chen Y.C. An OpenMP Programming Environment on Mobile Devices. *Mobile Information Systems*. 2016. 4513486.

40. Gayatri R. Yang C. Kurth T. Deslippe J. A Case Study for Performance Portability Using OpenMP 4.5. *Accelerator Programming Using Directives Proceedings of the 5th International Workshop WACCPD 2018*. 2019. С. 75–95.

41. OpenACC. Available online: http://www.openacc-standard.org (дата доступу 25 лютого 2025).

42. Liang L, He H, Zhao J, Liu C, Luo Q, Chu X. An Erasure-Coded Storage System for Edge Computing. *IEEE Access*. 2020. 8. С. 96271–96283.

43. Huber J, Cornelius M, Georgakoudis G, Tian S, Diaz JM, Dinel K, Chapman B, Doerfert J. Efficient Execution of OpenMP on GPUs. In *Proceedings of the 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, Seoul, Republic of Korea, 2–6 April 2022. С. 41–52.

44. Bailey D, Barszcz E, Barton J, Browning D, Carter R, Dagum L, Fatoohi R, Frederickson P, Lasinski T, Schreiber R, та ін. The NAS Parallel Benchmarks. *Int. J. High Perform. Comput. Appl.* 1991. 5. С. 63–73.

45. NAS Parallel Benchmarks. Available online: https://www.nas.nasa.gov/software/npb.html (дата доступу 25 лютого 2025).

46. NAS Parallel Benchmarks Changes. Available online: https://www.nas.nasa.gov/software/npb_changes.html (дата доступу 25 лютого 2025).

47. Varghese B, Wang N, Bermbach D, Hong CH, Lara ED, Shi W, Stewart C. A Survey on Edge Performance Benchmarking. *ACM Comput. Surv.* 2021. 54. С. 1–33.

48. Seo S, Jo G, Lee J. Performance Characterization of the NAS Parallel Benchmarks in OpenCL. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, USA, 6–8 November 2011. С. 137–148.

49. Do Y, Kim H, Oh P, Park D, Lee J. SNU-NPB 2019: Parallelizing and Optimizing NPB in OpenCL and CUDA for Modern GPUs. In *Proceedings of the 2019 IEEE International Symposium on Workload Characterization (IISWC)*, Orlando, FL, USA, 3–5 November 2019. С. 93–105.

50. Araujo G, Griebler D, Rockenbach DA, Danelutto M, Fernandes LG. NAS Parallel Benchmarks with CUDA and beyond. *Softw. Pract. Exp.* 2021. 53. С. 53–80.

51. Kang P, Lim S. A Taste of Scientific Computing on the GPU-Accelerated Edge Device. *IEEE Access*. 2020. 8. С. 208337–208347.

52. Ionica MH, Gregg D. The Movidius Myriad Architecture's Potential for Scientific Computing. *IEEE Micro*. 2015. 35. С. 6–14.

53. Amazon AWS for the Edge. Available online: https://aws.amazon.com/edge (дата доступу 25 лютого 2025).

54. Microsoft Azure Edge Zone. Available online: https://docs.microsoft.com/azure/networking/edge-zones-overview (дата доступу 24 грудня 2024).

55. Sunarjo MS, Gan HS, Setiadi DRI. High-performance convolutional neural network model to identify COVID-19 in medical images. *Journal of Computing Theories and Applications*. 2023. 1(1). С. 19–30.

56. Michalakes J. HPC for Weather Forecasting. In *Parallel Algorithms in Computational Science and Engineering*; Grama A, Sameh AH, ред.; Birkhäuser Cham: Basel, Switzerland, 2020. С. 297–323.

57. Richardson LF. Weather Prediction by Numerical Process. 2nd ed. Cambridge University Press: London, UK, 1922. С. 236.

58. Eadline D. The evolution of HPC. In *The insideHPC Guide to Co-Design Architectures Designing Machines Around Problems: The Co-Design Push to Exascale*, insideHPC; LLC: Moscow, Russian, 2016. С. 3–7. Available online: https://insidehpc.com/2016/08/the-evolution-of-hpc/ (дата доступу 25 лютого 2025).

59. Li T, Narayana VK, El-Ghazawi T. Exploring Graphics Processing Unit (GPU) Resource Sharing Efficiency for High Performance Computing. *Computers*. 2013. 2. С. 176–214.

60. Dabrowski JJ, Zhang Y, Rahman A. ForecastNet: A time-variant deep feed-forward neural network architecture for multi-step-ahead time-series forecasting. In *Proceedings of the International Conference on Neural Information Processing*, Bangkok, Thailand, 18–22 November 2020; Springer: Cham, Switzerland, 2020. С. 579–591.

61. Espeholt L, Agrawal S, Sønderby C, Kumar M, Heek J, Bromberg C, Gazen C, Hickey J, Bell A, Kalchbrenner N. Skillful twelve hour precipitation forecasts using large context neural networks. *arXiv*. 2021. arXiv:2111.07470.

62. Pathak J, Subramanian S, Harrington P, Raja S, Chattopadhyay A, Mardani M, Kurth T, Hall D, Li Z, Azizzadenesheli K, та ін. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators. *arXiv*. 2022. arXiv:2202.11214.

63. Müller M, Aoki T. Hybrid Fortran: High Productivity GPU Porting Framework Applied to Japanese Weather Prediction Model. In *International Workshop on Accelerator Programming Using Directives*; Springer: Cham, Switzerland, 2017. C. 20–41.

64. Hersbach H, Bell B, Berrisford P, Hirahara S, Horanyi A, Muñoz-Sabater J, Nicolas J, Peubey C, Radu R, Schepers D, та ін. The ERA5 global reanalysis. *Q. J. R. Meteorol. Soc.* 2020. 146. C. 1999–2049.

65. Nakaegawa T, Pinzon R, Fabrega J, Cuevas JA, De Lima HA, Cordoba E, Nakayama K, Lao JIB, Melo AL, Gonzalez DA, та ін. Seasonal changes of the diurnal variation of precipitation in the upper Río Chagres basin, Panamá. *PLoS ONE*. 2019. 14. e0224662.

66. ECWMF 2021a: Fact Sheet: Supercomputing at ECMWF. Available online: https://www.ecmwf.int/en/about/media-centre/focus/2021/fact-sheet-supercomputing-ecmwf (дата доступу 25 лютого 2025).

67. Müller A, Deconinck W, Kühnlein C, Mengaldo G, Lange M, Wedi N, Bauer P, Smolarkiewicz PK, Diamantakis M, Lock S-J, та ін. The ESCAPE project: Energy-efficient Scalable Algorithms for Weather Prediction at Exascale. *Geosci. Model Dev.* 2019. 12. C. 4425–4441.

68. Jungclaus JH, Lorenz SJ, Schmidt H, Brovkin V, Brüggemann N, Chegini F, Crüger T, De-Vrese P, Gayler V, Giorgetta MA, та ін. The ICON Earth System Model Version 1.0. *J. Adv. Modeling Earth Syst.* 2022. 14. e2021MS002813.

69. Courant R, Friedrichs K, Lewy H. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math. Ann.* 1928. 100. C. 32–74. (In German)

70. Schättler U. Operational NWP at DWD Life before Exascale. In *Proceedings of the 19th Workshop on HPC in Meteorology*, Reading, UK, 20–24 September 2021. Available online: https://events.ecmwf.int/event/169/contributions/2770/attachments/1416/2542/HPC-WS_Schaettler.pdf (дата доступу 25 лютого 2025).

71. Fuhrer O., Osuna C., Lapillonne X., Gysi T., Cumming B., Bianco M., Arteaga A., Schulthess T.C. Towards a performance portable, architecture agnostic implementation strategy for weather and climate models. *Supercomput. Front. Innov.* 2014. 1, 45–62. Available online: http://superfri.org/superfri/article/view/17 (accessed on 17 March 2018).

72. Fuhrer O., Chadha T., Hoefler T., Kwasniewski G., Lapillonne X., Leutwyler D., Lüthi D., Osuna C., Schär C., Schulthess T.C., et al. Near-global climate simulation at 1 km resolution: Establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geosci. Model Dev.* 2018. 11, 1665–1681.

73. CSCS. Fact Sheet: "Piz Daint", One of the Most Powerful Supercomputers in the World. 2017. p. 2. Available online: https://www.cscs.ch/fileadmin/user_upload/contents_publications/factsheets/piz_daint/FSPizDaint_Final_2018_EN.pdf (accessed on 25 February 2025).

74. Hosansky D. New NCAR-Wyoming Supercomputer to Accelerate Scientific Discovery; NCAR & UCAR News: Boulder, CO, USA, 27 January 2021. Available online: https://news.ucar.edu/132774/new-ncar-wyoming-supercomputer-accelerate-scientific-discovery (accessed on 25 February 2025).

75. R-CCS. RIKEN Center for Computational Science Pamphlet. 2021. p. 14. Available online: https://www.r-ccs.riken.jp/en/wp-content/uploads/sites/2/2021/09/r-ccs_pamphlet_en.pdf (accessed on 25 February 2025).

76. Duc L., Kawabata T., Saito K., Oizumi T. Forecasts of the July 2020 Kyushu Heavy Rain Using a 1000-Member Ensemble Kalman Filter. *SOLA* 2021. 17, 41–47.

77. R-CCS. About Fugaku 2022. Available online: https://www.r-ccs.riken.jp/en/fugaku/about/ (accessed on 25 February 2025).

78. JAMSTEC 2013. JAMSTEC Vision. Available online: https://www.jamstec.go.jp/e/about/vision/ (accessed on 25 February 2025).

79. Mizuta R., Oouchi K., Yoshimura H., Noda A., Katayama K., Yukimoto S., Hosaka M., Kusunoki S., Kawai H., Nakagawa M. 20-km-mesh global climate simulations using JMA-GSM model—mean climate states—. *J. Meteorol. Soc. Jpn. Ser. II* 2006. 84, 165–185.

80. Haarsma R.J., Roberts M.J., Vidale P.L., Senior C.A., Bellucci A., Bao Q., Chang P., Corti S., Fučkar N.S., Guemas V., et al. High Resolution Model Intercomparison Project (HighResMIP v1.0) for CMIP6. *Geosci. Model Dev.* 2016. 9, 4185–4208.

81. Kawase H., Imada Y., Sasaki H., Nakaegawa T., Murata A., Nosaka M., Takayabu I. Contribution of Historical Global Warming to Local-Scale Heavy Precipitation in Western Japan Estimated by Large Ensemble High-Resolution Simulations. *J. Geophys. Res. Atmos.* 2019. 124, 6093–6103.

82. Hatfield S., Düben P., Chantry M., Kondo K., Miyoshi T., Palmer T. Choosing the Optimal Numerical Precision for Data Assimilation in the Presence of Model Error. *J. Adv. Model. Earth Syst.* 2018. 10, 2177–2191.

83. Nakano M., Yashiro H., Kodama C., Tomita H. Single Precision in the Dynamical Core of a Nonhydrostatic Global Atmospheric Model: Evaluation Using a Baroclinic Wave Test Case. *Mon. Weather Rev.* 2018. 146, 409–416.

84. NVIDIA 2024a. Training, HPC up to 20x. Available online: https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/ (accessed on 25 February 2025).

85. NVIDIA 2022a. NVIDIA A100 Tensor Core GPU. pp. 3. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf (accessed on 25 February 2025).

86. Suda R. Fast Spherical Harmonic Transform Routine FLTSS Applied to the Shallow Water Test Set. *Mon. Weather Rev.* 2005. 133, 634–648.

87. Nguyen H.T., Krishnan P., Krishnaswamy D., Usman M., Buyya R. Quantum Cloud Computing: A Review, Open Problems, and Future Directions. 2024.

88. Reed D., Gannon D., Dongarra J. Reinventing High Performance Computing: Challenges and Opportunities. 2022.

89. Ramesh R.S. Scalable Systems and Software Architectures for High-Performance Computing on Cloud Platforms. 2024.

90. Netto M.A.S., Calheiros R.N., Rodrigues E.R., Cunha R.L.F., Buyya R. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. 2017.

91. Bailey, B. What Is An XPU? Semiconductor Engineering 11 November 2021. Available online: https://semiengineering.com/what-is-an-xpu/ (дата доступу 25 лютого 2025).

92. Reinders, J.; Ashbaugh, B.; Brodman, J.; Kinsner, M.; Pennycook, J.; Tian, X. Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems Using C++ and SYCL; Springer Nature: Berlin, Germany, 2021; p. 548.

93. Лисенко С.М., Бондар О.М. Збірник наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «*Актуальні проблеми комп'ютерних наук АПКН-2024*». (Хмельницький 2024. С. 303-305).

ДОДАТОК А
(обов'язковий)

ПУБЛІКАЦІЯ

УДК 004.7.056.5

Бондар О.М., Лисенко С.М.

Хмельницький національний університет

ДОСЛІДЖЕННЯ СИСТЕМ ДЛЯ ВИСОКОПРОДУКТИВНИХ ОБЧИСЛЕНЬ НА МОБІЛЬНИХ ПРИСТРОЯХ ІЗ ВИКОРИСТАННЯМ ХМАРНИХ ТЕХНОЛОГІЙ

Досліджено хмарні технології IaaS і PaaS, що дозволяють віртуалізацію обчислювальних ресурсів, і інструменти для обробки великих даних, такі як Hadoop та Dryad. Ці технології ефективно обробляють великі обсяги даних, але їх ефективність знижується при збільшенні обсягу оброблюваних даних або в додатках, що потребують швидкого обміну даними між вузлами. Хоча системи, як-от Hadoop, Dryad, CGL-MapReduce та MPI, забезпечують паралельне обчислення, обмеження в комунікації вимагають нових методів для зниження затримок і підвищення гнучкості в управлінні даними.

IaaS and PaaS cloud technologies that enable virtualization of computing resources and big data processing tools such as Hadoop and Dryad are explored. These technologies efficiently process large volumes of data, but their efficiency decreases when the volume of processed data increases or in applications that require fast data exchange between nodes. Although systems such as Hadoop, Dryad, CGL-MapReduce, and MPI provide parallel computing, communication limitations require new methods to reduce latency and increase flexibility in data management.

Хмара та хмарні технології є основними напрямками хмарних обчислень. "Хмара" включає різні інфраструктурні сервіси, такі як IaaS (інфраструктура як послуга) і PaaS (платформа як послуга), які надають організації завдяки віртуалізації. "Хмарні технології" охоплюють інструменти для хмарних середовищ виконання, такі як Hadoop і MapReduce, а також системи зберігання даних і комунікації, наприклад, Hadoop Distributed File System (HDFS) та Amazon [1].

У традиційних підходах, після розробки паралельних додатків, вони виконуються на обчислювальні кластери, суперкомп'ютери або GRID-інфраструктури, де фокус на розподілі ресурсів сильно залежить від наявності обчислювальних потужностей. Для того, щоб програма і дані були виконані, їх потрібно перемістити на доступні обчислювальні ресурси. Ці інфраструктури є високоефективними у виконанні паралельних додатків з інтенсивними обчисленнями. Однак, коли обсяги даних, до яких звертається додаток, збільшуються, загальна ефективність знижується через неминуче переміщення даних [2, 3].

Hadoop. Apache Hadoop має подібну архітектуру до середовища виконання MapReduce від Google, де доступ до даних здійснюється через HDFS, яка відображає всі локальні диски обчислювальних вузлів в єдину ієрархію файлової

системи, що дозволяє розподіляти дані між усіма вузлами зберігання даних/обчислень. HDFS також реплікує дані на декілька вузлів, тому вихід з ладу будь-якого вузла, що містить частину даних, не вплине на обчислення, які використовують ці дані. Hadoop планує обчислювальні завдання MapReduce в залежності від розташування даних, покращуючи загальну пропускну здатність вводу/виводу. Вихідні дані завдань відображення спочатку зберігаються на локальних дисках до тих пір, поки пізніше, коли до них не отримають доступ завдання редукції їх (pull) через HTTP-з'єднання. Хоча такий підхід спрощує механізм обробки збоїв в Hadoop, він додає значні комунікаційні витрати на проміжні передачі даних, особливо для додатків, які часто генерують невеликі проміжні результати [4].

Dryad та DryadLINQ. Dryad - це розподілений виконавчий рушій для паралельних додатків з великими масивами даних. Він поєднує стиль програмування MapReduce з графами потоків даних для вирішення обчислювальних задач. Dryad розглядає обчислювальні задачі як орієнтовані ациклічні графи (DAG), де вершини представляють обчислювальні задачі, а ребра виступають в якості каналів зв'язку, по яких дані передаються від однієї вершини до іншої. Дані зберігаються на локальних дисках (або розбиваються на розділи) за допомогою спільних каталогів Windows і файлів метаданих, а Dryad планує виконання вершин в залежності від розташування даних. (Примітка: Академічна версія Dryad містить лише API DryadLINQ (Y.Yu, Isard et al. 2008) для програмістів. Тому всі наші реалізації написані з використанням DryadLINQ, хоча він використовує Dryad як основне середовище виконання). Dryad також зберігає результати роботи вершин на локальних дисках, а інші вершини, які залежать від цих результатів, отримують доступ до них через спільні каталоги. Це дозволяє Dryad повторно виконувати невдалі вершини, що покращує відмовостійкість моделі програмування [5, 6].

CGL-MapReduce. CGL-MapReduce - це полегшене середовище виконання MapReduce, яке включає декілька покращень моделі програмування MapReduce, таких як (i) швидша передача проміжних даних через мережу pub/sub broker; (ii) підтримка довготривалих завдань map/reduce; та (iii) ефективна підтримка ітераційних обчислень MapReduce. Архітектура CGL-MapReduce показана на рисунку 1 (ліворуч). Використання потокової передачі даних дозволяє CGL-MapReduce надсилати проміжні результати безпосередньо від виробників до споживачів і усуває накладні витрати на файлові механізми передачі даних, прийняті в Hadoop і Dryad. Підтримка довготривалих завдань map/reduce дозволяє конфігурувати та повторно використовувати завдання map/reduce у випадку ітеративних обчислень MapReduce, а також усуває необхідність повторного конфігурування або повторного завантаження статичних даних у кожній ітерації. Ця особливість пов'язана з розрізненням "статичних даних" і "динамічних даних", які ми підтримуємо в CGL-MapReduce. Крім того, CGL-MapReduce підтримує розподіл менших наборів даних змінних між усіма завданнями мапи безпосередньо, функція, подібна до MPI_Bcast(), яка часто виявляється корисною у багато додатків для

аналізу даних. Hadoop надає подібну функцію за допомогою розподіленого кешу, в якому файл або дані копіюються на всі обчислювальні вузли [7-9].

Таблиця 1 – Порівняння можливостей, що підтримуються різними середовищами виконання паралельного обчислення даних

Особливість	**Hadoop**	**Dryad**	**CGL-MapReduce**	**MPI**
Модель програмування	MapReduce	Потоки виконання на основі DAG	MapReduce	Різноманітність топологій, побудованих з використанням багатого набору паралельних конструкцій
Обробка даних	HDFS	Спільні каталоги/ Локальні диски	Спільна файлова система / Локальні диски	Спільний файл системи
Проміжна передача даних	HDFS/ Точка-точка через HTTP	Файли/TCP-труби/ Спільна пам'ять FIFO	Мережа розповсюдження контенту	Канали зв'язку з низькою затримкою
Планування	Місцезнаходження даних / знання про стійку	Графік часу виконання на основі локалізації даних/топології мережі оптимізації	Місцезнаходження даних	Доступні можливості обробки
Обробка збоїв	Стійкість через HDFS Повторне виконання завдань мапування та скорочення	Повторне виконання вершин	Наразі не реалізовано (Повторне виконання завдань карти, скорочення надлишкових завдань)	Програмний рівень Перевірка точки доступу OpenMPI, FT MPI
Моніторинг	Моніторингова підтримка HDFS, Моніторинг Обчислення MapReduce	Підтримка моніторингу графіків виконання	Програмний інтерфейс для контролю за ходом виконання завдань	Мінімальна підтримка моніторингу на рівні завдань
Мовна підтримка	Реалізовано з використанням Java. Інші мови підтримуються через потокове передавання Hadoop	Програмування на C# DryadLINQ забезпечує програмування на LINQ API для Dryad	Реалізовано за допомогою Java Інші мови підтримуються за допомогою Java обгортки	C, C++, Fortran, Java, C#

MPI. MPI є незалежним від мови протоколом зв'язку, який використовує парадигму передачі повідомлень для обміну даними та станом між набором кооперативних процесів, що працюють у розподіленій системі пам'яті. Специфікація MPI (Forum, MPI) визначає набір процедур для підтримки різних

моделей паралельного програмування, таких як комунікація "точка-точка", колективна комунікація, похідні типи даних і паралельні операції вводу/виводу. Більшість середовищ виконання MPI розгортаються в обчислювальних кластерах, де набір обчислювальних вузлів з'єднано за допомогою високошвидкісного мережевого з'єднання, що забезпечує дуже низькі затримки зв'язку (як правило, мікросекунди). [10-12].

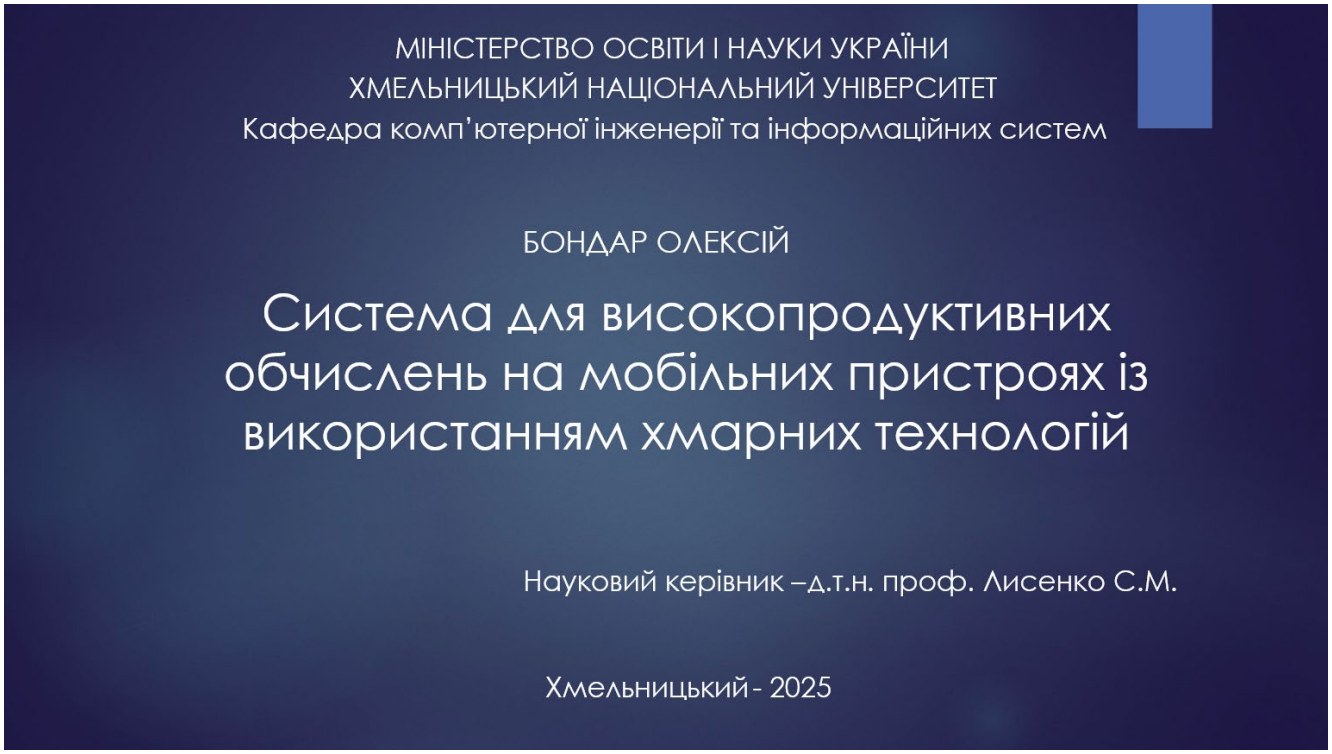
Таким чином, загальним недоліком технологій Hadoop, Dryad, CGL-MapReduce і MPI є їхня обмежена ефективність у підтримці паралельних обчислень, що потребують низької затримки та високої швидкості обміну даними між вузлами. Ці системи орієнтовані на обробку великих обсягів даних, але не завжди можуть забезпечити необхідну швидкість і гнучкість для більш інтенсивних комунікаційних моделей. Розробка нового методу, який би забезпечував менші затримки та ефективніше управління комунікаціями між вузлами, могла б покращити ефективність таких систем для широкого кола паралельних додатків.

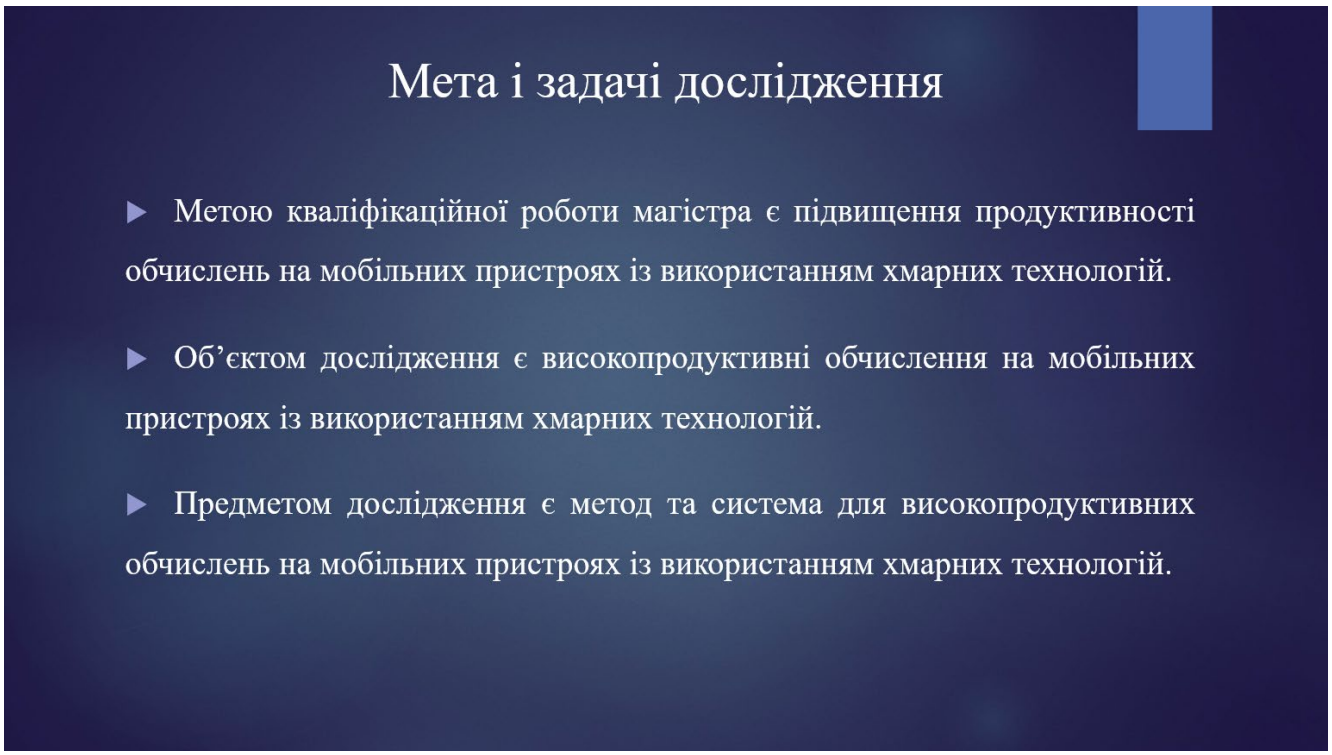
Перелік посилань

1. YANAMALA, Anil Kumar Yadav. Optimizing data storage in cloud computing: techniques and best practices. International Journal of Advanced Engineering Technologies and Innovations, 2024, 1.3: 476-513.
2. MISTRY, Hirenkumar Kamleshbhai, et al. The Impact Of Cloud Computing And Ai On Industry Dynamics And Competition. Educational Administration: Theory and Practice, 2024, 30.7: 797-804.
3. LIM, Sooyoung; PARK, Dongchul. Improving Hadoop MapReduce performance on heterogeneous single board computer clusters. Future Generation Computer Systems, 2024.
4. GUAN, Shaopeng, et al. Hadoop-based secure storage solution for big data in cloud computing environment. Digital Communications and Networks, 2024, 10.1: 227-236.
5. KONG, Xiaohui, et al. Association between Hematocrit in the First Two Hours of Life and Retinopathy during Prematurity: A Retrospective Study from DRYAD. 2024.
6. HABERMANN, Ted. Sustainable Connectivity in a Community Repository. Data Intelligence, 2024, 1-36.
7. HUANG, Min, et al. Digital Protection and Innovative Development Path of Red Culture Resources Based on Distributed Machine Learning Supported by Intelligent Information. Journal of Combinatorial Mathematics and Combinatorial Computing, 2024, 120: 381-391.
8. XING, Chengcai. Harnessing Traditional Business Culture Resources: Scalable Parameter Server Architecture with Distributed Machine Learning. Journal of Electrical Systems, 2024, 20.7s: 217-226.
9. ZHOU, Hui, et al. MPI Progress For All. arXiv preprint arXiv:2405.13807, 2024.
10. ADAM, Julien, et al. To Share or Not to Share: a case for MPI in Shared-Memory. In: European MPI Users' Group Meeting. Cham: Springer Nature Switzerland, 2024. p. 89-102.
11. SCHEFFLER, Konrad; BOBERG, Marija; KNOPP, Tobias. Solving the MPI reconstruction problem with automatically tuned regularization parameters. Physics in Medicine & Biology, 2024, 69.4: 045024.
12. WU, Zhenning, et al. A Parallel Sequential SBAS Processing Framework Based on Hadoop Distributed Computing. Remote Sensing, 2024, 16.3: 466.

ДОДАТОК Б
(обов'язковий)

ПРЕЗЕНТАЦІЯ





Мета і задачі дослідження

Поставлена мета досягається розв'язанням таких основних завдань:

- проаналізувати відомі методи продуктивності обчислень на мобільних пристроях із використанням хмарних технологій;

- розробити моделі здійснення високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

- розробити метод здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій;

- здійснити дослідження методу здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій.

Наукова новизна та практична цінність отриманих результатів

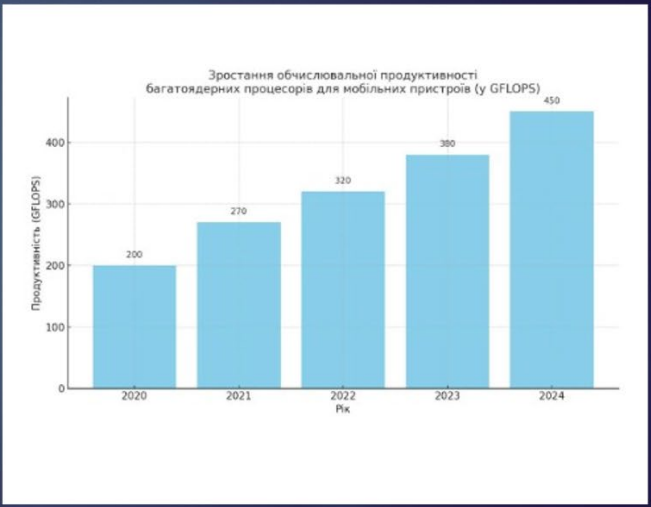
Наукова новизна отриманих результатів:

- розроблено новий метод здійснення високопродуктивних обчислень мобільних пристроях із використанням хмарних технологій, уможливлює зменшення обсяг повторного обчислення шляхом локалізації змін у графі залежностей, забезпечує узгоджене виконання потоків, розподіляє обчислювальне навантаження між мобільними пристроями з використанням хмарної інфраструктури, що знижує затрати на комунікацію та обчислення при збереженні даних.

- удосконалено систему для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Актуальність дослідження

- Розпаралелення дозволяє ефективно використовувати всі ядра мобільного процесора для обробки ресурсоємних задач у реальному часі.
- Оптимальний розподіл навантаження між ядрами і хмарною інфраструктурою зменшує локальне навантаження на пристрій.
- Поєднання локальних обчислень і хмарних сервісів забезпечує масштабовану обробку даних, що надходять від IoT-пристроїв



Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Крок 1	Крок 2	Крок 3
Декомпозиція на підобчислення	Побудова графа конкурентних динамічних залежностей	• Поширення змін і поступове повторне виконання.

Метод здійснення високопродуктивних обчислень

Характеристика методу

Оптимізація обчислень - зменшення повторних обчислень шляхом локалізації змін у графі залежностей.

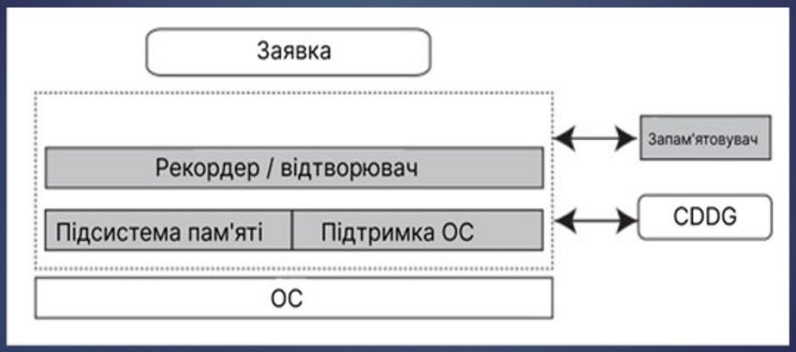
Узгоджене виконання потоків- забезпечення синхронізації і ефективної взаємодії між обчислювальними задачами.

Розподіл навантаження - динамічне розподілення обчислень між пристроєм та використання результатів обчислення в хмарі для максимізації продуктивності.

Масштабованість - можливість адаптації системи до зростання обчислювальних потреб

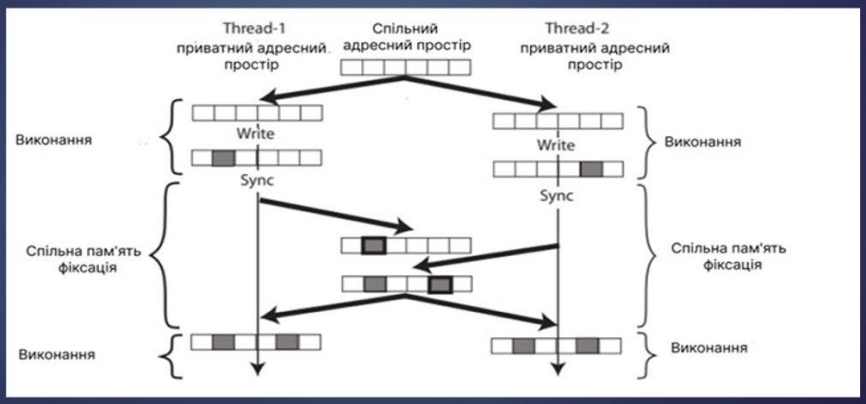
Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Архітектура системи для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

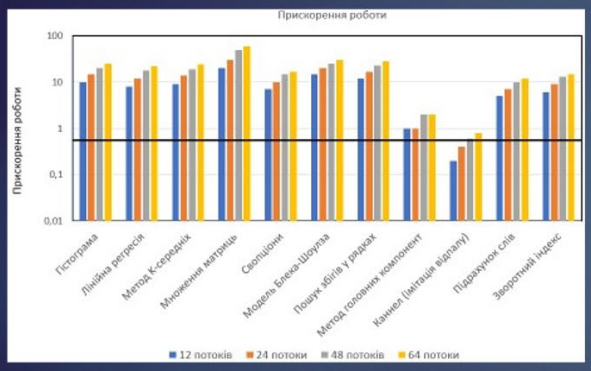


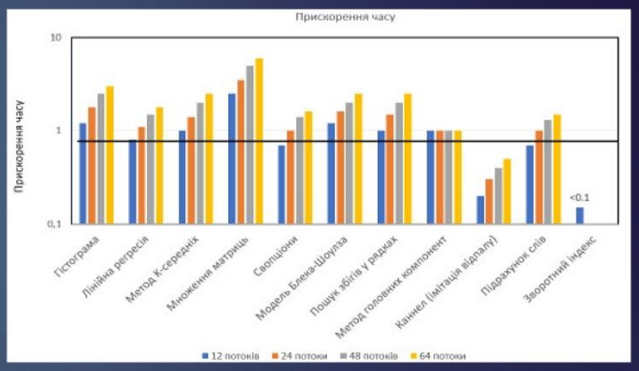
Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Виконання задач



Результати приросту продуктивності СПЗ





Приріст продуктивності СПЗ по відношенню приінкрементному виконанні

Публікації

- За темою кваліфікаційної роботи магістра опубліковані тези у матеріалах конференції XXIV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів «Актуальні проблеми комп'ютерних наук АПКН-2024» Хмельницький 2024. С. 303-305

Висновки

- У першому розділі проаналізовано можливості застосування високопродуктивних обчислень (HPC) на мобільних пристроях з використанням хмарних технологій. Встановлено переваги такого підходу, зокрема масштабованість і енергоефективність, а також визначено основні проблеми: обмежені ресурси пристроїв, затримки зв'язку, ризики безпеки та висока вартість хмарних сервісів.
- Другий розділ присвячено формалізації моделі системи HPC, яка охоплює мобільні пристрої, хмарну інфраструктуру, планування задач, безпеку та інтерфейси. Враховано вплив змінних ресурсів і мережевих характеристик, що дозволяє гнучко адаптувати систему до умов роботи.
- У третьому розділі запропоновано метод інкрементального багатопотокового виконання з використанням конкурентного динамічного графа залежностей. Метод забезпечує мінімізацію повторних обчислень, оптимізацію часу й енерговитрат, а також ефективний розподіл задач.
- Четвертий розділ описує реалізацію системи у вигляді бібліотеки для Android. Застосовано модель пам'яті Release Consistency і механізм «потік-як-процес», що забезпечують ефективну обробку пам'яті та зниження накладних витрат.

ДОДАТОК В
(обов'язковий)

ЛІСТИНГИ СИСТЕМНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

// Фрагмент коду мовою C++ для Android NDK

```
// TaskGraph.cpp
#include <jni.h>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <vector>
#include <atomic>
#include <unordered_map>
#include <queue>
#include <functional>

class Task {
public:
    using TaskFunc = std::function<void()>;

    Task(int id, TaskFunc func) : id(id), func(func), dependencies(0) {}

    void addDependency() {
        dependencies++;
    }

    void execute() {
        if (func) func();
    }

    int id;
    TaskFunc func;
    std::atomic<int> dependencies;
    std::vector<int> dependents;
};

class TaskGraph {
public:
    void addTask(int id, Task::TaskFunc func) {
        std::unique_lock<std::mutex> lock(mutex_);
        tasks_[id] = std::make_shared<Task>(id, func);
    }

    void addDependency(int from, int to) {
        std::unique_lock<std::mutex> lock(mutex_);
        auto fromTask = tasks_[from];
        auto toTask = tasks_[to];
        toTask->addDependency();
        fromTask->dependents.push_back(to);
```

```
    }

    void run() {
        std::queue<std::shared_ptr<Task>> readyQueue;

        for (auto& [id, task] : tasks_) {
            if (task->dependencies.load() == 0) {
                readyQueue.push(task);
            }
        }

        std::vector<std::thread> workers;
        std::mutex queueMutex;

        for (int i = 0; i < std::thread::hardware_concurrency(); ++i) {
            workers.emplace_back([&]() {
                while (true) {
                    std::shared_ptr<Task> task = nullptr;

                    {
                        std::unique_lock<std::mutex> lock(queueMutex);
                        if (readyQueue.empty()) break;
                        task = readyQueue.front();
                        readyQueue.pop();
                    }

                    task->execute();

                    for (int dependentId : task->dependents) {
                        auto dependent = tasks_[dependentId];
                        if (--(dependent->dependencies) == 0) {
                            std::unique_lock<std::mutex> lock(queueMutex);
                            readyQueue.push(dependent);
                        }
                    }
                }
            });
        }

        for (auto& t : workers) {
            if (t.joinable()) t.join();
        }
    }

private:
    std::unordered_map<int, std::shared_ptr<Task>> tasks_;
    std::mutex mutex_;
};
```

JNI Обгортка (TaskGraphJNI.cpp)

```
#include <jni.h>
#include "TaskGraph.cpp"  // або використай #include "TaskGraph.h"

extern "C"
JNIEXPORT void JNICALL
Java_com_example_myapp_TaskGraphRunner_runTaskGraph(JNIEnv* env, jobject /*
this */) {
    TaskGraph graph;

    // Додаємо задачі
    graph.addTask(1, []() { /* Тяжкі обчислення */ });
    graph.addTask(2, []() { /* Тяжкі обчислення */ });
    graph.addTask(3, []() { /* Залежить від 1 і 2 */ });

    // Залежності
    graph.addDependency(1, 3);
    graph.addDependency(2, 3);

    graph.run();
}
```

Java Код для виклику

```
public class TaskGraphRunner {
    static {
        System.loadLibrary("native-lib");
    }

    public native void runTaskGraph();
}
```

2. Android UI (XML)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="24dp">

    <Button
        android:id="@+id/runTasksButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Запустити обчислення" />

    <TextView
        android:id="@+id/resultText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Результат з'явиться тут"
        android:layout_marginTop="20dp"/>
</LinearLayout>
```

3. Java Код

```
package com.example.parallelgraph;

import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    static {
        System.loadLibrary("native-lib");
    }

    private native String runTaskGraph();  // JNI-метод

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button runButton = findViewById(R.id.runTasksButton);
        TextView resultText = findViewById(R.id.resultText);

        runButton.setOnClickListener(v -> {
            String result = runTaskGraph();
            resultText.setText(result);
        });
    }
}
```

4. C++ Код (NDK)

```
#include <jni.h>
#include <string>
#include <thread>
#include <vector>
#include <unordered_map>
#include <queue>
#include <functional>
#include <mutex>
#include <atomic>

class Task {
public:
    using TaskFunc = std::function<void()>;

    Task(int id, TaskFunc func) : id(id), func(func), dependencies(0) {}

    void addDependency() {
        dependencies++;
    }

    void execute() {
        if (func) func();
    }

    int id;
    TaskFunc func;
    std::atomic<int> dependencies;
    std::vector<int> dependents;
};

class TaskGraph {
public:
    void addTask(int id, Task::TaskFunc func) {
        std::unique_lock<std::mutex> lock(mutex_);
        tasks_[id] = std::make_shared<Task>(id, func);
    }

    void addDependency(int from, int to) {
        std::unique_lock<std::mutex> lock(mutex_);
        auto fromTask = tasks_[from];
        auto toTask = tasks_[to];
        toTask->addDependency();
        fromTask->dependents.push_back(to);
    }

    std::string run() {
        std::queue<std::shared_ptr<Task>> readyQueue;
        for (auto& [id, task] : tasks_) {
            if (task->dependencies.load() == 0) {
```

```
                readyQueue.push(task);
            }
        }

        std::mutex queueMutex;
        std::vector<std::thread> workers;
        std::atomic<int> resultSum = 0;

        for (int i = 0; i < std::thread::hardware_concurrency(); ++i) {
            workers.emplace_back([&]() {
                while (true) {
                    std::shared_ptr<Task> task = nullptr;

                    {
                        std::unique_lock<std::mutex> lock(queueMutex);
                        if (readyQueue.empty()) break;
                        task = readyQueue.front();
                        readyQueue.pop();
                    }

                    task->execute();

                    for (int depId : task->dependents) {
                        auto dep = tasks_[depId];
                        if (--(dep->dependencies) == 0) {
                            std::unique_lock<std::mutex> lock(queueMutex);
                            readyQueue.push(dep);
                        }
                    }
                }
            });
        }

        for (auto& t : workers) {
            if (t.joinable()) t.join();
        }

        return "Обчислення завершено!";
    }

private:
    std::unordered_map<int, std::shared_ptr<Task>> tasks_;
    std::mutex mutex_;
};

// JNI інтерфейс
extern "C"
JNIEXPORT jstring JNICALL
Java_com_example_parallelgraph_MainActivity_runTaskGraph(JNIEnv* env,
jobject /* this */) {
    TaskGraph graph;
```

```
    // Прикладні задачі з обчисленням
    graph.addTask(1, []() {
        std::this_thread::sleep_for(std::chrono::milliseconds(300));
    });

    graph.addTask(2, []() {
        std::this_thread::sleep_for(std::chrono::milliseconds(400));
    });

    graph.addTask(3, []() {
        std::this_thread::sleep_for(std::chrono::milliseconds(200));
    });

    // Залежності: 1 та 2 → 3
    graph.addDependency(1, 3);
    graph.addDependency(2, 3);

    std::string result = graph.run();
    return env->NewStringUTF(result.c_str());
}
```

5. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18.1)
project("parallelgraph")

add_library(
        native-lib
        SHARED
        native-lib.cpp)

find_library(
        log-lib
        log)

target_link_libraries(
        native-lib
        ${log-lib})
```

5. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18.1)
project("parallelgraph")

add_library(
        native-lib
        SHARED
        native-lib.cpp)

find_library(
        log-lib
        log)

target_link_libraries(
        native-lib
        ${log-lib})
```

6. AndroidManifest.xml

```
<application ... >
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олексій БОНДАР

Співавтор:

Назва: БОНДАР_Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:7.6%

Коефіцієнт подібності 2:3%

Мікропробіли: 4

Заміна букв: 14

Інтервали: 0

Білі знаки: 1

Дата створення звіту: 2025-04-28 19:23:18.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-04-28 Доцент Андрій Нічепорук

Дата експерт

Mon Apr 28 20:16:53 EEST 2025, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.274 Educational

Максимальне співпадіння з одним документом 28.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 12%**

ID: 240563 Назва: МКР Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій Додано в БД: 2025-04-28 Автора: Олексій БОНДАР Керівники: Сергій ЛИСЕНКО Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	123102	970	36199 (29%)	242 (25%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
193097	Назва: Звіт з ПДП Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій Додано в БД: 2025-03-21 Автора: Бондаря О. М. Керівники: Лисенко С.М. Консультанти: Опоненти:	34153 (28.0%)	213 (22.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Бондар Олексій Миколайович

Тема: Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень – ; **кількість сторінок записки** 86

1. Короткий зміст роботи та прийнятих рішень: У роботі запропоновано систему, що дозволяє виконувати ресурсоємні обчислення на мобільних пристроях за допомогою хмарних технологій.

2. Висновок про відповідність роботи дипломному завданню: Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проаналізовано актуальні методи високопродуктивних обчислень, включаючи хмарні технології, кластерні системи та мобільне хмарне обчислення (MCC), що активно досліджуються у світовій практиці. У другому розділі побудовано модель системи з поділом обчислювальних задач між мобільним пристроєм і хмарою з урахуванням таких сучасних підходів, як динамічний offloading та енергозбереження. У третьому розділі розроблено архітектуру системи, що поєднує мобільний додаток, хмарну платформу та REST API, із використанням передових засобів програмного забезпечення й концепцій безпечної передачі даних. У четвертому розділі реалізовано прототип системи із застосуванням Google Cloud Platform, Android SDK та Flask, проведено експериментальне тестування, що продемонструвало переваги використання хмари для обробки ресурсоємних задач на мобільних пристроях.

4. Позитивні сторони роботи: Система підвищує продуктивність мобільних пристроїв, знижує енергоспоживання, забезпечує масштабованість, швидкодію та

легку інтеграцію з іншими сервісами завдяки використанню хмарних технологій і REST API.

5. Негативні сторони роботи: Серед недоліків системи є залежність від стабільного інтернет-з'єднання, можливі затримки при передачі даних, ризики безпеки під час обміну інформацією з хмарою, а також додаткові витрати на хмарні ресурси.

6. Оцінка графічного оформлення та пояснювальної записки роботи: відсутній.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: Відсутні.

9. Оцінка дипломної роботи: Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «відмінно» (4,35/B).

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) д.т.н., професор, Мартинюк В.В., завідувач кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"29" квітня 2025 р. _____ (підпис)

Завідувачу кафедри КІІС
доктору філософії, доцент
Ользі ПАВЛОВІЙ

Бондар Олексій Миколайович
ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-23-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (StrikePlagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

19.04.2025
дата

підпис

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система для високопродуктивних обчислень на мобільних пристроях із використанням хмарних технологій

Автор: Бондар Олексій Миколайович

Спеціальність: 123 – Комп`ютерна інженерія

Освітня програма: освітньо-наукова

Науковий керівник: Лисенко Сергій Миколайович, д.т.н., професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) системи перевірки виявили збіги з іншими документами в частині стандартних формулювань, структури змісту та назв розділів, що є типовими для кваліфікаційних робіт.
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи лише на частину речення;
4) в якості запозичень в окремих місцях системою зафіксовано послідовності програмного коду, які є вхідними даними до вирішення задач і не можуть розглядатися як об`єкт авторських прав і, відповідно, їх порушення;
5) усі ознаки модифікації тексту, зафіксовані системою, стосуються поєднання латинських символів з україномовними скороченнями індексів у формулах, що не може вважатися зміною самого тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 7.6 % і адресується до 57 першоджерела; та системою Anti-Plagiarism складає 28%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Сергій ЛИСЕНКО

Гарант ОП _____ Олег САВЕНКО

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА