Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Система оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах»

08-54.БКР.003.00.000 ПЗ

Виконала: студентка 4 курсу, групи КІ-23мсз
спеціальність 123 – Комп'ютерна інженерія
освітня програма – Комп'ютерна інженерія
_____ Кривоносова З.В.

Керівник: к.т.н., доцент каф. ОТ
_____ Колесник І.С.

Рецензент: к.т.н., доц., зав. каф. МБІС
_____ Карпінець В.В.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д. _____
«13» 06 2025 р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ проф.,
д.т.н. Азаров О.Д.
"21" 03 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Кривоносовій Зінаїді Володимирівні

1. Тема роботи: Система оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах, керівник роботи Колесник Ірина Сергіївна, к.т.н., доцент кафедри ОТ, затверджено наказом вищого навчального закладу від " 20 " березня 2025 року №97.

2. Строк подання студентом роботи 13.05.2025

3. Зміст текстової частини (перелік питань, які потрібно розробити): вступ; аналіз предметної області організації паралельних обчислень; розробка системи балансування навантаження в розподілених гетерогенних комп'ютерних середовищах; реалізація оптимізованих паралельних обчислень в розподілених гетерогенних комп'ютерних середовищах; висновки; перелік джерел посилань.

4. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): схема процесу обробки завдань в розподіленій комп'ютерній системі; модифікована схема процесу обробки задач в розподілених гетерогенних комп'ютерних системах; карта розміщення вузлів тестових систем з відображенням умовних логістичних зв'язків між їх вузлами.

5. Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к.т.н., доц. каф. ОТ Колесник І.С.	21.03.2025	13.05.2025
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.2025	30.05.2025

6. Дата видачі завдання 21.03.2025

7. Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№ етапу	Назва етапу	Термін виконання		Примітка
		початок	кінець	
1	Аналіз завдання. Вступ	21.03.2025	25.03.2025	Виконано
2	Аналіз предметної області організації паралельних обчислень	26.03.2025	01.04.2025	Виконано
3	Розробка системи балансування навантаження в розподілених КС	02.04.2025	25.04.2025	Виконано
4	Реалізація оптимізованих паралельних обчислень в розподілених КС	26.04.2025	02.05.2025	Виконано
5	Оформлення пояснювальної записки	03.05.2025	13.05.2025	Виконано

Студент _____ Зінаїда КРИВОНОСОВА

Керівник роботи _____ к.т.н., доц. каф. ОТ Ірина КОЛЕСНИК

АНОТАЦІЯ

УДК 519.87

Кривоносова З.В. Система оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця, ВНТУ, 2025. 81с.

На укр. мові. Бібліогр.: 16 назв; рис.: 14; табл.: 3.

В даній бакалаврській кваліфікаційній роботі досліджено процес розробки системи оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах. Проведено аналіз існуючих рішень організації паралельних обчислень в гетерогенних комп'ютерних середовищах. Проаналізовано процес обробки задач в розподіленій комп'ютерній системі. Описано організацію структури та режиму роботи з довільними обчисленнями розподіленої гетерогенної комп'ютерної системи. Описано процес розробки програмного забезпечення для оптимізації паралельних обчислень в розподілених гетерогенних комп'ютерних системах шляхом балансування навантаження на їх вузли. Розроблено пакет тестових програмних комплексів для організації роботи розподілених гетерогенних комп'ютерних систем із акселераторами, які було розгорнуто в двох реальних хмарних РГКС. В рамках кожного із програмних комплексів було організовано розв'язання ряду поширених математичних та прикладних задач

Ключові слова: паралельні обчислення, гетерогенні комп'ютерні середовища, балансування навантаження, графічні акселератори.

ABSTRACT

Krivonosova Z. The system of optimization of parallel calculations in distributed heterogeneous computer environments. Bachelor's qualification work in specialty 123 "Computer Engineering", educational program "Computer Engineering". Vinnitsa, VNTU, 2025. 81p.

On the Ukrainian. language. Bibliogr: 16 titles; Fig.: 14; Table: 3.

This bachelor's qualification has investigated the process of developing a system of optimization of parallel computing in distributed heterogeneous computer environments. The analysis of existing decisions of the organization of parallel computing in heterogeneous computer environments is carried out. The process of processing problems in a distributed computer system is analyzed. The organization and mode of operation with arbitrary calculations of the distributed heterogeneous computer system are described. The process of software development for optimization of parallel computing in distributed heterogeneous computer systems is described by balancing the load on their units. A package of test software complexes was developed to organize the work of distributed heterogeneous computer systems with accelerators, which was deployed in two real cloud RGCS. As part of each of the software complexes, a number of common mathematical and applied tasks was organized

Keywords: parallel calculations, heterogeneous computer environments, load balancing, graphic accelerators.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОРГАНІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ 6
1.1 Організація обчислень в розподілених гетерогенних комп'ютерних середовищах 6
1.2 Процес обробки задач в розподілених комп'ютерних системах 11
1.3 Аналіз сучасних підходів до паралельного програмування 15
1.4 Аналіз сучасних технологій програмування для паралельних та розподілених систем 23
1.5 Висновки до розділу 1 25
2 РОЗРОБКА СИСТЕМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В РОЗПОДІЛЕНИХ ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СЕРЕДОВИЩАХ 27
2.1 Аналіз апаратної та програмної складової розподілених гетерогенних комп'ютерних середовищ 27
2.2 Оцінювання вхідної задачі 31
2.3 Опис алгоритму ініціалізації системи 33
2.4 Опис алгоритму оптимізації паралельних обчислень на основі балансування навантаження 39
2.5 Висновки до розділу 2 48
3 РЕАЛІЗАЦІЯ ОПТИМІЗОВАНИХ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ В РОЗПОДІЛЕНИХ ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СИСТЕМАХ 50
3.1 Обґрунтування вибору засобів реалізації прототипів 50
3.2 Створення контрольних прототипів 54
3.3 Опис тестової системи 55
3.4 Опис процесу тестування 58
3.5 Висновки до розділу 3 67
ВИСНОВКИ 69
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 70

ДОДАТОК А 72
ДОДАТОК Б 75
ДОДАТОК В 76

ВСТУП

Актуальність теми обґрунтовується сучасною тенденцією до створення численних комп'ютерних систем із гетерогенною будовою, що базуються на використанні центрального процесора та супутніх акселераторів з архітектурою, відмінною від основної. Водночас, для забезпечення належної продуктивності комплексних програмних рішень застосування виключно паралельних систем стає недостатнім. Загальноприйнятою практикою для обчислень з високим навантаженням є їх впровадження в розподілених комп'ютерних системах, де паралельні системи виступають лише структурними компонентами. У будь-якій розподіленій системі одним із визначальних чинників ефективності функціонування виступає раціональний розподіл навантаження на всіх рівнях цієї системи. Розв'язання цієї NP-повної проблеми наразі становить окремий напрям комп'ютерних наук, однак вона залишається нагальним питанням, оскільки віднайти абсолютно досконале рішення за прийнятний час поки неможливо. У випадках, коли кінцеві вузли розподіленої системи представлені гетерогенними комп'ютерними системами, завдання суттєво ускладнюється, адже необхідно забезпечити рівномірний розподіл обчислень між різнотипними елементами, характеристики продуктивності та комунікаційні можливості яких часто істотно відрізняються.

На сьогодні існує широкий спектр стандартних методологій балансування навантаження в розподілених системах, проте більшість із них орієнтовані на специфічні типи завдань, і лише незначна кількість має універсальний характер. Сучасні інструменти організації паралельних обчислень у розподілених комп'ютерних системах з акселераторами характеризуються низьким рівнем абстракції, що породжує численні труднощі та обмеження. Вбудовані механізми балансування навантаження у таких системах переважно розраховані на гомогенні та здебільшого повнозв'язні конфігурації.

Мета бакалаврської кваліфікаційної роботи полягає у створенні системи оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних

середовищах.

Для реалізації поставленої мети необхідно вирішити такі завдання:

— провести аналіз сучасних підходів до організації паралельних обчислень у гетерогенних комп'ютерних середовищах;
— дослідити наявні методи оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах;
— розглянути ключові проблеми, що виникають при формуванні розподілених гетерогенних комп'ютерних середовищ та організації паралельних обчислень у них, вивчити запропоновані шляхи їх вирішення;
— здійснити експериментальні випробування щодо оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах;
— оцінити та проаналізувати результати проведеного дослідження.

Об'єктом дослідження виступає процес оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах.

Предметом дослідження є програмні засоби оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах.

Методологія дослідження бакалаврської кваліфікаційної роботи ґрунтується на застосуванні технологій C, C# для програмної реалізації системи оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах.

Практична цінність отриманих результатів – розроблено систему оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ОРГАНІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

1.1 Організація обчислень в розподілених гетерогенних комп’ютерних середовищах

Протягом всієї історії становлення та еволюції сучасних обчислювальних технологій ключовими параметрами для оцінювання результативності роботи процесорних елементів та побудованих на їх базі обчислювальних платформ традиційно виступали тимчасові витрати на опрацювання конкретних завдань та розмір оперативної пам'яті, що резервується для здійснення відповідних операцій. Втім, в останні роки спостерігається істотне зростання доступних обсягів оперативної пам'яті в комп'ютерних архітектурах, внаслідок чого домінуючим критерієм для визначення продуктивності подібних систем залишається переважно темпоральний аспект. Відтак, сучасні алгоритмічні концепції та підходи до оптимізації обчислювальних операцій орієнтуються першочерговим чином на скорочення цього параметра як на рівні апаратних компонентів, так і на рівні програмних рішень.

Розглядаючи апаратну складову, початковим етапом у напрямку прискорення функціонування процесорного компонента стало збільшення його робочої частоти. Проте подібне збільшення обмежується фізичними законами, і на сьогоднішній день частотні показники найсучасніших мікросхем наближаються до граничних можливостей кремнієвих технологічних рішень. Наступною стратегією стала інтеграція в рамках єдиного процесорного елемента множини незалежних ядер, які здатні реалізовувати обчислювальні операції одночасно. Обчислювальні платформи, що базуються на таких процесорах, одержали найменування паралельних комп'ютерних архітектур. Водночас, окрім значного ускладнення вимог до програмного забезпечення, розробленого для подібних систем, з'явилася також необхідність істотного підвищення складності інших апаратних елементів для гарантування ефективної роботи кожного додаткового ядра.

Таким чином, надалі збільшення кількості ядер задовольняло потреби

звичайних споживачів, але не відповідало динаміці зростання вимог наукових та комерційних сфер діяльності. Одночасно з цим розвиток мережевих технологій прогресував, забезпечуючи високошвидкісну передачу значних обсягів інформаційних потоків. Тому подальшим вирішенням для підвищення швидкості обчислювальних операцій у комп'ютерних архітектурах стала ідея розподілу обчислювальних завдань між множиною автономних однотипних паралельних комп'ютерних платформ через локальну мережеву інфраструктуру. Новий тип обчислювальних архітектур отримав назву кластерних комп'ютерних систем.

Дане рішення представляло собою певний баланс переваг та недоліків. З одного боку, фінансові витрати на розширення потужностей у кластерних комп'ютерних архітектурах були значно меншими у порівнянні з витратами на збільшення кількості ядер або робочої частоти, що дозволяло створення великомасштабних систем цього типу, які отримали окреме найменування -- суперкомп'ютерні комплекси. Але в свою чергу, високопродуктивне системне програмне забезпечення, а також прикладне програмне забезпечення, що використовують такі типи архітектур, виявлялось набагато складнішим, у порівнянні з програмами, які використовують стандартні паралельні платформи.

Крім того, створення кластерних комп'ютерних мереж, які працюють на основі сучасних паралельних високопотужних архітектур теж зіштовхнулося з чисельними проблемами. Подібні системи займали дедалі більше фізичного простору, вимагали потужних систем термостабілізації та електричного живлення, найпотужніші суперкомп'ютерні комплекси зазвичай розміщувалися безпосередньо поряд з електростанціями. Відповідно, розробка та експлуатація таких архітектур були доступні виключно великим науково-дослідним установам та корпораціям.

Паралельно з цими процесами розвиток комп'ютерних мереж забезпечив прийнятну швидкість передачі інформації через глобальну мережу, що разом із появою на ринку широкого спектру доступних процесорних компонентів дало можливість виділити понадкластерні системи більш широкої категорії -- розподілені комп'ютерні архітектури. Компоненти таких систем можуть

підключатися в необмеженій кількості, мати різноманітні технічні характеристики, навіть представляти собою кластерні системи, і при цьому локалізуватися як в одному приміщенні, так і в різних регіонах світу.

Додатковим підходом до оптимізації тривалості виконання програм у комп'ютерних системах стало розділення обчислювальних операцій всередині самої архітектури. Оскільки процесор є центральним компонентом системи, навіть під час інтенсивного опрацювання визначеної користувачем задачі він реалізує численні інші, не пов'язані з поточним завданням операції.

Особливість функціонування супервізора операційної системи полягає в тому, що всі ці завдання, особливо з підвищеним пріоритетом, розміщуються на початку черги завдань. Тобто, користувацьке обчислення, в будь-якому випадку, незалежно від наших намірів, періодично призупинятиметься, а системні ресурси спрямовуватимуться на опрацювання більш пріоритетних завдань. Тому розвантаження процесора шляхом звільнення його від реалізації допоміжних і системних завдань через їхнє делегування окремим спеціалізованим обчислювачам, що отримали назву співпроцесорів або прискорювачів, також відіграє важливу роль у підвищенні швидкості обчислень на центральному процесорі.

Графічний процесор (GPU - англ. graphics processing unit) є одним із найбільш значущих спеціалізованих прискорювачів. Його створення дозволило розвантажити CPU від однієї з найресурсоємніших операцій -- відображення користувацького інтерфейсу. У спрощеному вигляді архітектуру GPU можна представити як комплекс великої кількості однотипних простих процесорних компонентів. Кожен із цих компонентів є значно економічнішим порівняно з аналогами в CPU. Це сприяло інтенсивному розвитку GPU щодо збільшення кількості ядер, і сьогодні типова кількість ядер у середньостатистичному GPU досягає п'ятисот. Проте, як показує практичний досвід, якщо гетерогенна комп'ютерна система, що включає і CPU, і GPU, не використовується для обробки складної графіки, то наявна в сучасних GPU обчислювальна потужність виявляється надмірною, обчислення виконуються настільки швидко, що більшість

часу ядра GPU перебувають у стані очікування наступного завдання від CPU.

Застосування цього незадіяного обчислювального потенціалу, значного та економічно доцільного, є логічним вирішенням питання підвищення продуктивності обчислень. З розробкою та інтеграцією в GPU програмованих шейдерних блоків з'явилася можливість програмування стандартних користувацьких обчислень для GPU. Ця технологія отримала найменування GPGPU (англ. General-purpose computing on graphics processing units, неспеціалізовані обчислення на графічних процесорах) і стала визначальним прориввом у розвитку сучасної комп'ютерної техніки.

Перші графічні процесори компанії Nvidia, які підтримували технологію CUDA (від англ. Compute Unified Device Architecture, уніфікована обчислювальна архітектура пристрою - реалізація техніки GPGPU від цієї компанії) характеризувалися високою вартістю та обмеженою доступністю. Проте незабаром з'явилися альтернативні реалізації GPGPU, серед яких варто відмітити фреймворк OpenCL. Розроблений компанією Apple для мови C++, він поширюється під відкритою ліцензією та забезпечує можливість програмування загальних обчислень для GPU не лише від компанії Nvidia, а, наприклад, і компанії AMD. Також, на додаток до того, що OpenCL підтримує ширший спектр GPU порівняно з CUDA, такі GPU зазвичай є доступнішими та простішими, включаючи навіть звичайні відеокарти.

Отже, впровадження техніки GPGPU та таких технологій як CUDA й OpenCL, окрім істотного прискорення обчислень, також надало можливість малому та середньому бізнесу, а також більшості звичайних інститутів та університетів створити власні потужні обчислювальні центри, оскільки розподілені комп'ютерні системи, засновані на гетерогенних системах із CPU та GPU, забезпечують достатню швидкість обчислень і водночас є значно економічнішими порівняно з обчислювальними системами, побудованими виключно на великій кількості CPU або GPU. Також, як демонструє практичний досвід, техніка GPGPU стала однією з основ динамічного розвитку систем штучного інтелекту.

Отже, можна зробити висновок, що на сучасному етапі розвитку

комп’ютерних технологій основна увага у контексті пришвидшення розв’язання великомасштабних обчислювальних задач зосереджується на розподілених обчислювальних системах, зокрема гетерогенних. Ефективне функціонування таких систем значною мірою залежить від правильного, з урахуванням продуктивності компонентів, розподілу навантаження між усіма елементами паралельної структури системи, включно з кожним процесором або апаратним прискорювачем (за їх наявності).

У разі, коли навантаження розподілене нерівномірно, потенційне прискорення роботи системи суттєво зменшується. Навіть якщо всі вузли мають однакову продуктивність, то при розподілі обчислень на P процесорів, протягом часу TM ефективно будуть задіяні не більше ніж PM процесорів (де PM < P). Відповідно до закону Амдала [3], граничне значення коефіцієнта прискорення KS, можливе у таких умовах, визначається за формулою (1.1):

$$K_S = \frac{T_1}{T_P} = \frac{T_1}{\frac{T_1 K_P}{P} + \frac{T_1(1-K_P)}{P_M}} = \frac{1}{\frac{K_P}{P} + \frac{1-K_P}{P_M}} \quad , \qquad (1.1)$$

де K_P – коефіцієнт, який показує яка частина обчислень виконується на всіх P ядрах;

T_1 – час виконання програми на одному ядрі;

T_P – час виконання програми на всіх P ядрах.

За умов значного збільшення кількості процесорів P, граничне значення виразу дорівнює максимально досяжному коефіцієнту прискорення SMAX (1.2) [3]:

$$S_{MAX} = \frac{P_M}{1-P} \quad , \qquad (1.2)$$

де S_{MAX} – максимально можливий в системі із P ядер коефіцієнт прискорення.

Отже, верхня межа можливого прискорення системи обмежується саме ступенем нерівномірного завантаження. Відповідно, зменшення цієї

нерівномірності є ключовим напрямом оптимізації продуктивності в розподілених обчислювальних середовищах.

1.2 Процес обробки задач в розподілених комп’ютерних системах

З метою визначення ключових елементів розподілених обчислювальних платформ, конструктивні та функціональні вади яких спричиняють асиметричне використання наявних обчислювальних можливостей, першочергово слід проаналізувати повний цикл процесів опрацювання завдань у подібних архітектурах.

До моменту призначення для виконання конкретним обчислювальним компонентом архітектури, кожне завдання, що поступає до системи, проходить через множину рівнів оптимізації, які реалізуються різноманітними планувальниками. Кожен такий планувальник аналізує розподілену комп'ютерну систему (РКС) з позиції власного рівня абстракції. Серед таких рівнів доцільно виділити наступні категорії:

- рівень системного планувальника РКС - на цьому етапі архітектура інтерпретується як комплекс різнорідних компонентів, які абстрагуються до рівня підсистем (кластерного або паралельного типу). Базуючись на архітектурній схемі системи та визначених критеріях ефективності складових елементів РКС і каналів зв'язку між ними, планувальник створює загальний граф паралельних операцій та реалізує оптимальний розподіл завдань чи підзавдань, що виділені в графі, направляючи їх разом з необхідними даними до відповідних підсистем. Безпосередньої участі в обчислювальних процесах не приймає, однак може ініціювати повторне виконання завдання у випадку його неопрацювання в межах встановленого часового ліміту;
- рівень планувальника кластерної підсистеми - характеризується призначенням спеціалізованого планувальника для кожної підсистеми.

Такий планувальник інтерпретує архітектуру як абсолютно гомогенну структуру та гарантує рівномірний розподіл отриманих завдань між складовими компонентами. Прямої участі в обчислювальних операціях не приймає;

— рівень планувальника операційної системи - функціонує в прихованому від користувача режимі та координує безпосереднє виконання завдань на специфічних обчислювальних ядрах, а також керує направленням окремих завдань до спеціалізованих прискорювачів. Реалізує міграційні процеси або прив'язку завдань до конкретних ядер, оптимізує їх виконання. Водночас не володіє можливістю забезпечення міграції завдань за межі фізичної машини;
— рівень внутрішньоядерного паралелізму - для певних категорій сучасних процесорів дозволяє одночасне виконання множини потоків на єдиному фізичному ядрі. Дана технологічна концепція отримала найменування Hyperthreading. Переключення потоків у межах ядра реалізується на апаратному рівні, характеризується підвищеною прозорістю для операційної системи, виконується швидше за програмну міграцію потоків між ядрами, та може ініціюватися апаратними подіями (зокрема, промахами кешу);
— рівень інструкційного паралелізму - більшість сучасних процесорних елементів, що застосовуються в РКС, мають суперскалярну архітектуру. Кожне ядро такого процесора володіє здатністю одночасного виконання кількох інструкцій протягом одного системного такту.

Відповідно до наукових досліджень професора Стіренко, загальний процес опрацювання завдань в РГКС може бути представлений схематично, як показано на рисунку 1.1 [3].

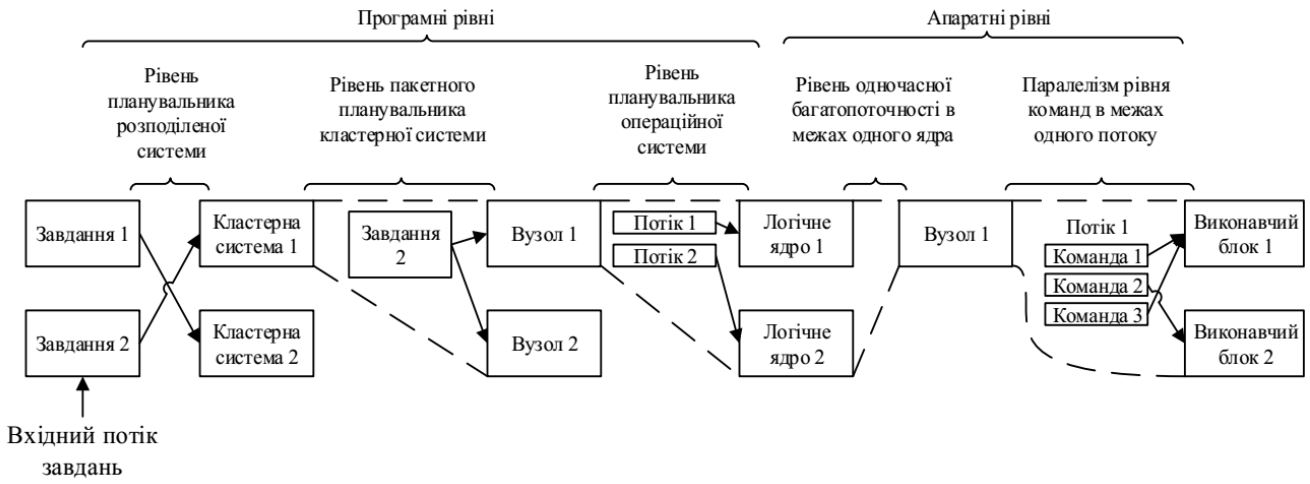

Рисунок 1.1 — Схема рівнів обробки задач в розподіленій комп’ютерній системі

Базуючись на проведеному аналізі, можна констатувати, що в рамках описаної моделі опрацювання завдань у РКС системні ресурси, необхідні для виконання операцій, резервуються одномоментно на початку процесу та вивільняються синхронно після завершення виконання. Завдання позбавлене можливості вивільнення тих ядер, які стали надлишковими внаслідок завершення їх функціонування, або формування запиту на додаткове резервування обчислювальних ресурсів.

Додатково слід зазначити, що не враховується сценарій, коли розробником програмного забезпечення експліцитно передбачено декомпозицію певних завдань на субзавдання безпосередньо в процесі виконання (наприклад, у Fork-Join архітектурі). Це також генерує дисбаланс навантаження, який проявляється виключно під час виконання, залишаючись непомітним для планувальників розподілених, кластерних та паралельних систем. Саме цей фактор обумовлює виникнення проблематики балансування навантаження в РКС.

Концептуально балансування навантаження можна класифікувати на статичне та динамічне. Статичне балансування реалізується до безпосереднього ініціювання виконання завдань та базується на архітектурі завдань, а також алгоритмічних підходах до оцінювання, планування та прогнозування навантаження і продуктивності. Динамічне балансування, на противагу, здійснюється безпосередньо в процесі функціонування завдання та не залежить від

прогностичних оцінок тривалості виконання.

Динамічний підхід характеризується більш широкою сферою застосування завдяки можливості роботи з завданнями, що мають нерегулярний паралелізм. В цілому, таке балансування може реалізовуватися автоматично на рівні системного планувальника операційної системи та набуває додаткової ефективності при переході в програмному забезпеченні від концепції потоків до концепції завдань, що одночасно зменшує гранулярність паралелізму. Завдання, які є менш ресурсоємними порівняно з повноцінними потоками, можуть планувартися та переміщуватися операційною системою з підвищеною швидкістю порівняно з потоками.

Водночас, не для всіх типів завдань доцільним є перехід від потокової моделі паралелізму до моделі, заснованої на завданнях. Крім того, управління завданнями може здійснюватися виключно на рівні планувальника операційної системи, що виключає можливість перерозподілу виконання на вищих рівнях архітектури в процесі функціонування.

Таким чином, виникає необхідність забезпечення можливості автоматичного оптимального завантаження компонентів та автоматичного розподілу завдань на вищих рівнях системної ієрархії, не лише на етапі ініціалізації, а й безпосередньо в процесі виконання. Жоден з ідентифікованих на рисунку 1.1 рівнів опрацювання завдань у РКС не інтегрує механізми динамічного балансування навантаження. Відтак, першочергово необхідно ідентифікувати рівні, де можливе та доцільне застосування такого балансування, а згодом конкретизувати, що становитиме базову одиницю балансування навантаження в архітектурі.

Як статичне, так і динамічне балансування навантаження неможливі без попереднього аналітичного дослідження завдання та оцінювання різноманітних темпоральних характеристик у системі. При цьому на результати таких аналізів та характеристик суттєво впливає модель паралельного програмування, що застосовується та підтримується в конкретній РКС.

1.3 Аналіз сучасних підходів до паралельного програмування

Ключовою тенденцією в сучасній розробці програмного забезпечення, як прикладного, так і системного характеру, є намагання застосовувати найвищі можливі рівні абстракції. Цьому процесу значною мірою сприяє активний розвиток та зростаюча популярність мов програмування високого рівня, що надають можливість розробникам зосереджуватися на створенні програмної логіки, використовуючи синтаксичні та семантичні конструкції, що тісно пов'язані з предметною сферою, а не з апаратними чи системними компонентами.

Основна слабкість даного підходу полягає в обмеженні можливостей програміста щодо взаємодії з системою на найглибших рівнях, включаючи апаратний, що унеможливлює проведення детального налаштування оптимізації програми. Втім, позитивні аспекти переважають цей недолік. До них належать: прискорення процесу створення програмного забезпечення, розширення потенціалу для повторного застосування коду, покращення портативності та сумісності, створення передумов для автоматизованого тестування, відлагодження та оптимізації, оскільки програмний код більшою мірою описує цілі, а не способи їх досягнення.

В області паралельного програмного забезпечення дана тенденція також знаходить своє втілення. У сучасних мовах програмування та бібліотеках для паралельних обчислень з'являється дедалі більше високорівневих абстракцій. Ці абстракції дозволяють програмісту відмовитися від концепції ресурсоємних повноцінних потоків на користь більш легких завдань, або навіть просто визначити потенційно придатні для паралельної обробки сегменти послідовної програми (зазвичай це ітерації циклічних конструкцій), після чого компілятор самостійно організовує необхідний паралелізм.

Крім того, незважаючи на надзвичайну складність автоматичного паралелізму на сьогоднішній день, сучасні оптимізуючі компілятори включають певні його елементи. Навіть без спеціальних вказівок від розробника вони здатні забезпечувати автоматичну паралельну обробку деяких програмних сегментів, що

підходять для розпаралелювання.

Однак такі методології, попри їх постійний розвиток, залишаються можливими виключно в паралельних системах із загальною пам'яттю. Як зазначалося раніше, такі системи мають суттєві обмеження щодо розширення кількості процесорних елементів, тому в даний час використовуються лише як термінальні вузли розподілених паралельних систем із приватною пам'яттю. Незважаючи на це, подібні підходи демонструють виняткову ефективність, що актуалізує завдання їх адаптації для використання в системах з приватною пам'яттю.

Оскільки подібні системи зазвичай є розподіленими, незалежно від найвищих рівнів абстракції, з програмної перспективи вони являють собою сукупність окремих процесів. Таким чином, постає питання організації ефективної взаємодії між цими процесами під час обробки завдань.

Існує два фундаментальні типи взаємодії паралельних процесів, реалізація яких необхідна для повноцінного функціонування будь-якої паралельної системи:

— передача даних;

— синхронізація паралельних процесів (потоків, завдань, тощо).

На програмному рівні можна виокремити дві базові моделі взаємодії паралельних процесів: модель, заснована на спільних змінних, та модель, що базується на обміні повідомленнями. Кожна з них забезпечує реалізацію обох основних операцій та гарантує повноцінне функціонування паралельної системи. На практиці ці моделі служать фундаментом для реалізації кінцевої моделі програмування, прийнятої в системі. Розглянемо їх більш детально.

Модель, заснована на статичному створенні потоків, передусім розроблена для застосування в середовищах паралельних обчислень із загальнодоступною пам’яттю. В її основі лежить необхідність вручну задавати у програмі набір потоків, а також використання стандартного інтерфейсу для контролю за їх поведінкою. До основного функціоналу такого інтерфейсу належать:

— ініціалізація нового потоку;

— запуск потоку на виконання;

— призупинення потоку на певний час;

— поновлення виконання потоку;

— завершення потоку примусовим способом;

— отримання інформації про поточний стан потоку;

— очікування завершення виконання потоку.

На основі деяких із цих операцій надаються також більш абстрактні програмні інструменти для організації взаємодії потоків, зокрема для розв'язання завдань взаємного виключення та синхронізації. Основними серед них є семафори, м'ютекси, синхронні методи, монітори, замки, бар'єри, події та атомарні операції й змінні. У паралельній програмі можуть виникати такі небажані ситуації, як блокування, гонки та відсутність гарантії завершення.

Ситуації, коли один або кілька потоків нескінченно довго очікують на певний сигнал від інших потоків і, відповідно, ніколи не можуть завершити свою роботу, називаються блокуваннями. При ручному формуванні потоків та відповідному ручному розміщенні в програмному коді елементів їх взаємодії людський фактор є основною причиною виникнення подібних ситуацій. Випадки, коли розробник не повністю передбачив усі можливі варіанти виконання потоків або неправильно розмістив операції захоплення та звільнення потоками елементів синхронізації, призводять до виникнення блокувань у роботі паралельних програм.

Під терміном «блокування» розуміється ситуація, коли один або більше потоків чекають на певну умову (наприклад, сигнал або ресурс), яка ніколи не настає, що призводить до нескінченного очікування. Основним фактором ризику тут виступає людський чинник — зокрема, ручне розміщення коду, що відповідає за синхронізацію та взаємодію потоків. Помилки в логіці, наприклад, неправильне розташування операцій захоплення/звільнення ресурсів, здатні спровокувати подібні зависання в програмі.

У таких випадках потоки припиняють активну роботу, утримуючи ресурси системи, й можуть бути відновлені лише втручанням операційної системи. Більш небезпечним варіантом є динамічне блокування — особлива форма, коли потоки не припиняються повністю, а безкінечно змінюють свої стани, не досягаючи

реального прогресу. Подібна ситуація часто виникає через часові залежності або типові програмістські помилки — нескінченні цикли, некоректні переходи, відсутність точок виходу.

Ще одна характерна проблема — явище, відоме як «голодування» потоків. У цьому випадку певні потоки залишаються у стані очікування доступу до спільного ресурсу, який згідно з логікою повинен бути їм доступним, однак через хибну організацію доступу він залишається недоступним на тривалий час або взагалі.

Зазначена проблема виникає як наслідок двох попередньо описаних ситуацій. Гарантія виконання (або прогресу) визначається як здатність паралельної програми досягти свого завершення незалежно від способу планування потоків та їхньої синхронізації. Відповідно, при самостійному створенні потоків та впровадженні блокуючих елементів синхронізації ця гарантія фактично відсутня.

Варто зазначити, що модель із ручним створенням потоків є історично першою та найпоширенішою. Через це постійно ведуться дослідження, спрямовані на прогнозування, запобігання та усунення вищеописаних проблем у паралельних програмах, розроблених за цією моделлю. Незважаючи на ці зусилля, ймовірність виникнення помилок у таких програмах залишається достатньо високою.

Істотним недоліком даної моделі є також необхідність частого використання блокуючих механізмів синхронізації, що призводить до формування в паралельній програмі послідовних ділянок виконання. Це суперечить основній ідеї розпаралелювання та, згідно з розглянутим раніше законом Амдала, встановлює принциповий верхній ліміт максимально можливого коефіцієнта прискорення таких програм.

Ключовою перевагою цієї моделі є надання програмісту можливості працювати з низькорівневими примітивами, що дозволяє максимально адаптуватися до особливостей цільової системи та забезпечує гнучкість у реалізації та модифікації будь-якої з описаних далі моделей.

Модель з динамічним створенням потоків можна охарактеризувати як одну з найскладніших і водночас найпотужніших моделей розпаралелювання. Це насамперед пов'язано з відсутністю обмежень щодо способів організації

паралелізму, структури обчислень та загальної архітектури програми.

Розробнику надається обмежений набір варіантів створення потоків, визначення їх станів та засобів взаємодії, які охоплюють найпоширеніші потреби. Зазвичай потрібно лише описати шаблон одного робочого потоку та логіку створення й завершення таких потоків. За такої моделі гарантія прогресу зазвичай вже присутня, що сприяє уникненню значної кількості потенційних блокуючих ситуацій.

Однією з найпоширеніших моделей у сфері паралельного програмування вважається рекурсивна модель Fork-Join. В її основі — дві ключові операції: Fork, що означає поділ одного (батьківського) потоку на декілька (зазвичай два, хоча це не є обов’язковим) дочірніх потоків з подальшим виконанням у кожному з них певних завдань (включаючи можливість нових викликів Fork-Join), та Join, яка відповідає за об'єднання результатів виконання дочірніх потоків у початковому потоці.

Цей підхід знайшов реалізацію в численних сучасних мовах програмування та бібліотеках. Наприклад: у Java — через пакет concurrent, у C/C++ — завдяки бібліотекам OpenMP, Futures і мові-розширенню Cilk, у Go — через goroutines, а в C# — за допомогою елементів бібліотеки TPL, включаючи задачі (Tasks), паралельні цикли та ітератори.

Однак дана модель має певні обмеження. Одним із недоліків є обмежена свобода дій для програміста, що ускладнює реалізацію деяких класів задач. Наприклад, у версії OpenMP 2.0 підтримка нерегулярного паралелізму залишається слабкою: директива pragma omp for чудово підходить для регулярного паралелізму, однак при нестандартних схемах доступу залишається лише pragma omp section, яка не дозволяє змінювати кількість потоків або розмір задач динамічно.

Для розв'язання подібних проблем у стандарт C++ у 2011 році була додана бібліотека Futures, яка дозволяє потокам передавати роботу іншим. Проте і тут відсутні механізми динамічної адаптації рівня паралелізму під час виконання програми — їх реалізація повністю покладається на розробника й повинна враховувати специфіку поставленого завдання.

Аналогічну функціональність демонструють засоби бібліотеки Futures, OpenMP 3.0 (Tasks) та Cilk, які мають як переваги, так і схожі недоліки.

Незважаючи на зазначені обмеження, концепція обмеженого управління потоками знаходить подальший розвиток. Основна ідея — повне усунення необхідності ручного створення потоків, як у статичному, так і в динамічному режимі. Замість цього вся відповідальність за організацію паралельності перекладається на бібліотеки часу виконання та компілятори, що забезпечують автоматичну оптимізацію й розпаралелювання.

Сучасні компілятори здатні самостійно проводити аналіз програмного коду з метою виявлення ділянок, які можуть бути виконані паралельно, та організовують їх запуск відповідним чином. Але такий підхід має свої вади. Насамперед, він не застосовний до інтерпретованих мов, оскільки ті виконуються послідовно, рядок за рядком, і не мають доступу до всієї структури програми наперед. У випадку навіть з оптимізатором це суперечить перевазі інтерпретаторів — відсутності необхідності в компіляції.

Крім того, завдання аналізу синтаксичного дерева за даними входить до класу NP-повних задач, отже, знаходження точного рішення є практично неможливим. Тому компілятори часто створюють лише приблизне, а іноді й надмірно складне паралельне виконання, що призводить до зниження ефективності — як через зайві взаємодії між потоками, так і через створення надлишкових критичних секцій.

Щоб подолати ці труднощі, було запропоновано підхід, що передбачає попереднє ручне маркування ділянок коду як потенційно паралельних. У цьому випадку компілятор або оптимізатор зосереджується на аналізі саме цих частин, що знижує загальні витрати на обробку й підвищує шанси на досягнення ефективного паралельного виконання. Яскравим прикладом подібного підходу є команда Parallel.Invoke у бібліотеці TPL (мова C#), яка дозволяє безпосередньо вказати задачі, що можуть виконуватись одночасно.

В обох випадках у разі виявлення можливостей до розпаралелювання програма сама визначає яку кількість потоків створити та як організувати взаємодію між ними.

Очевидні переваги цього підходу полягають у наступному:

— значне спрощення процесу створення паралельних програм, що дозволяє працювати навіть спеціалістам без глибоких знань у галузі паралелізму;
— можливість автоматичного регулювання розміру одиниці паралельної обробки (гранули);
— спрощення умов для динамічної адаптації цього розміру.

Проте основна вада цього підходу полягає в складності маніпулювання даними. У подібній моделі фактично не існує чіткого уявлення про дані, що призводить до проблем у забезпеченні коректного доступу до них або навіть унеможливлює автоматичне розпаралелювання в разі складних структур доступу.

На основі попереднього аналізу моделей можна стверджувати, що забезпечення правильної взаємодії з даними є ключовим чинником успішного впровадження паралельного виконання. Проблема не зводиться лише до уникнення конфліктів доступу, а й охоплює ризик отримання некоректних результатів при автоматичному управлінні потоками.

Одним із перспективних підходів для вирішення цих питань є використання транзакційної пам’яті, яка є розширенням концепції спільного доступу до змінних. Оскільки операції зчитування не порушують цілісності, потоки можуть вільно читати дані та виконувати над ними певні обчислення (розпочати транзакцію). Результати транзакції фіксуються лише за умови, що жоден інший потік паралельно не завершив свою транзакцію з цими ж даними.

До переваг цієї моделі належать:

— відсутність необхідності блокування для реалізації взаємного виключення;
— спрощення синхронізації завдяки усуненню потреби у ручному виділенні критичних ділянок;
— виключення можливості появи класичних ситуацій блокування.

Водночас варто враховувати і обмеження. За умов активного використання одних і тих самих даних великою кількістю потоків можливе постійне відхилення

транзакцій, що впливає на продуктивність. Крім того, хоча реалізація транзакційної пам’яті можлива і на програмному рівні, найбільша ефективність досягається лише за наявності апаратної підтримки цієї технології.

Конфлікти доступу до пам’яті виникають переважно через намагання кількох потоків одночасно змінити значення, розташоване за однією адресою. Це пояснюється тим, що хоча кожен потік має власний стек, йому не виділяється окремий простір пам’яті в адресному сенсі.

Історично це обумовлювалось високою вартістю пам’яті, однак із її здешевленням з'явилася можливість призначати кожному потоку окремий адресний простір. Такий підхід дозволяє уникнути потреби в критичних секціях, оскільки потоки працюють із власними даними. Однак досягнення теоретично максимального прискорення (згідно з законом Амдала) усе ще обмежується необхідністю передачі даних між потоками, що вимагає копіювання або пересилання.

Операції передачі інформації між потоками зазвичай називають обміном повідомленнями. Для реалізації цієї моделі достатньо двох базових операцій: надсилання та прийому повідомлення.

Безсумнівною перевагою цієї концепції в системах з локальною пам’яттю є її низькорівневий характер, який дає програмісту повний контроль над комунікаціями і дозволяє будувати високорівневі абстракції на її основі.

Проте це водночас і основна її слабкість: програмна реалізація надмірно відображає апаратну структуру обчислювальної системи, що ускладнює зосередження на логіці конкретної задачі. До того ж код, заснований на використанні низькорівневих інструкцій, зазвичай є громіздким і менш зручним для підтримки.

Наявність блокувань потоків при обробці критичних секцій у моделях, заснованих на спільних змінних, спричиняє низку недоліків:

— обмеження масштабованості системи;
— створення нових можливостей для виникнення блокувань;

— проблема вибору оптимального співвідношення між обсягом захищених даних та кількістю механізмів синхронізації.

Альтернативним підходом є повна відмова від блокувань у програмі, чого можна досягти шляхом використання алгоритмів вирішення задач синхронізації та взаємного виключення, які не потребують використання блокування.

Відповідно до рівня гарантій прогресу та наявності очікування потоків виділяють три основні групи неблокуючих паралельних алгоритмів:

— алгоритми без перешкод (англ. obstruction-free) надають найслабші гарантії;
— алгоритми без блокування (англ. lock-free) забезпечують гарантії прогресу обчислень у системі в цілому;
— алгоритми без очікування (англ. wait-free) забезпечують найсильніші гарантії прогресу.

1.4 Аналіз сучасних технологій програмування для паралельних та розподілених систем

Інтенсивний розвиток паралельних та розподілених систем вимагає як створення інноваційних, так і дослідження існуючих моделей і технологій їх програмування. На основі проведеного вище аналізу можна зробити висновок, що ключовим питанням є визначення рівня абстракції у вибраній моделі.

З одного боку, використання низькорівневих абстракцій теоретично забезпечує найбільш ефективне функціонування системи, оскільки розробник працює насамперед з поняттями, близькими до системного рівня. При цьому необхідно адаптувати поставлену задачу до абстракцій системного рівня.

З іншого боку, застосування високорівневих абстракцій дозволяє оперувати поняттями, наближеними до прикладного рівня та/або бізнес-логіки, звільняючи розробника від необхідності організації системної взаємодії.

Сучасні підходи до організації паралельних обчислень дедалі частіше ґрунтуються на використанні високорівневих абстракцій, що дозволяють розробникам працювати із математичними структурами та елементами бізнес-

логіки, не заглиблюючись у деталі реалізації паралелізму. У таких підходах паралельність реалізується на рівні загальної архітектури без необхідності ручного управління потоками.

Деякі з новітніх математичних бібліотек забезпечують можливість виконання функціональних викликів як у послідовному, так і в паралельному режимі без жодних змін у зовнішньому інтерфейсі [19]. Існують методики, що дозволяють описувати структуру паралелізму на рівні логіки задач та зв’язків між ними, не деталізуючи конкретну реалізацію [4]. Водночас, усі зазначені інструменти зазвичай орієнтовані на роботу в середовищах із загальнодоступною пам’яттю [3].

Як було розглянуто раніше, найбільш ефективними з точки зору масштабованості є архітектури, побудовані за принципами розподілених обчислювальних систем. Навіть обчислювальні системи з апаратними прискорювачами по суті функціонують як розподілені, адже взаємодіють із пам’яттю та обчислювальними ресурсами асинхронно.

Застосування абстрактних моделей у таких середовищах ставить перед програмним забезпеченням низку викликів, серед яких — автоматичне розбиття та передача даних, призначення обчислювальних блоків окремим вузлам, а також об’єднання результатів після виконання.

Ключову роль у цьому контексті відіграє проблема забезпечення доступності даних. У системах зі спільною пам’яттю ця задача значно спрощується, оскільки дані доступні всім обчислювальним потокам за замовчуванням. Натомість у середовищах з локальною пам’яттю розробник має вручну забезпечити передавання необхідних даних до вузла обробки через механізми копіювання або передачі по мережі.

Паралельні обчислення найчастіше реалізуються шляхом розбиття задачі на серію підзадач, які обробляються одночасно на кількох ядрах або потоках у межах одного обчислювального пристрою. У розподілених обчислювальних системах завдання виконуються на різних фізичних машинах, що координуються між собою через мережеві канали зв’язку.

Для зручності узагальнення особливостей сучасних паралельних технологій

доцільно навести порівняльну характеристику у вигляді таблиці (табл. 1.1).

Таблиця 1.1 — Порівняльна таблиця особливостей сучасних технологій паралельного програмування

Технологія	**Цільова ПКС**	**Режим доступу до даних**
Директиви препроцесору (OpenMP)	ПКС з СП	Копіювання автоматичне відповідно до вказівок; додаткові вказівки для взаємного виключення; лише спільна пам’ять.
Низькорівневий інтерфейс передачі повідомлень (MPI)	ПКС з ЛП	Обов’язкове явне копіювання даних між вузлами; можливість програмованої буферизації під час передачі.
Інтерфейс програмування графічних акселераторів (Nvidia CUDA, OpenCL)	ПКС з СП та акселераторами	Обов’язкове явне копіювання даних в пам’ять акселератора. Багаторівнева ієрархія пам’яті.
Низькорівневий інтерфейс передачі повідомлень з підтримкою графічних акселераторів (CUDA-MPI)	ПКС з ЛП та акселераторами	Те ж саме, що попереднє, та додатково можливість явного копіювання у пам’ять акселератора в іншому вузлі.
Технологія Intel Threading Building Blocks	ПКС з СП та застосуванням різних акселераторів	Автоматичне копіювання на акселератор та взаємне виключення в екземплярах спеціально структурованих типів; лише спільна пам’ять.
Стандартна бібліотека Futures мови C++ , стандартні засоби мов C#, Java, Node.js	ПКС з СП	Можливість відкладеного обчислення та доступу (за безпосереднім запитом), зокрема паралельного. Лише спільна пам’ять.

1.5 Висновки до розділу 1

У даному розділі було проведено комплексний аналіз існуючих рішень організації паралельних обчислень у гетерогенних комп'ютерних середовищах.

Детально розглянуто процес обробки задач у розподіленій комп'ютерній системі. Було досліджено сучасні методології паралельного програмування та здійснено порівняльний аналіз моделей паралельного програмування. Визначено проблему відсутності ефективної системи балансування навантаження в розподілених комп'ютерних системах, функціонування якої одночасно враховувало б такі важливі фактори:

— гетерогенність кінцевих вузлів системи;

— наявність неоднорідних акселераторів в цих вузлах;

— потенційну неповнозв'язність системи;

— гетерогенність комунікативного середовища системи.

Відповідно, існує необхідність розробки такої системи, яка враховувала б усі зазначені аспекти.

2 РОЗРОБКА СИСТЕМИ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ В РОЗПОДІЛЕНИХ ГЕТЕРОГЕННИХ КОМП’ЮТЕРНИХ СЕРЕДОВИЩАХ

2.1 Аналіз апаратної та програмної складової розподілених гетерогенних комп’ютерних середовищ

Для детального викладу алгоритму розподілу навантаження необхідно спочатку всебічно проаналізувати цільову систему, розглянути запропоновані в ній підходи та технологічні рішення, а також дослідити варіанти її роботи.

Досліджувана розподілена система включає комплекс обчислювальних вузлів та комунікаційні з'єднання між ними. Кожний обчислювальний вузол являє собою паралельну комп'ютерну архітектуру, що може характеризуватися розподіленою або централізованою пам'яттю. Характеристики пам'яті визначаються через присутність прискорювачів у складі вузла. У випадку наявності прискорювача, вузол автоматично відноситься до систем з розподіленою пам'яттю, де компонентами виступають прискорювач разом з основним процесорним блоком, який у поєднанні з оперативною пам'яттю вузла створює архітектуру з централізованою пам'яттю. За аналогічним принципом класифікується вузол, що не містить прискорювачів. Отже, на базовому рівні архітектури кожний вузол обов'язково включає паралельну комп'ютерну систему з централізованою пам'яттю.

Враховуючи присутність у системі вузлів з прискорювачами, комунікаційні з'єднання можуть реалізовуватися у двох варіантах:

- на рівні всієї розподіленої системи — різноманітні доступні канали передачі даних у рамках локальних та/або глобальних мережевих інфраструктур;
- на рівні окремого вузла — протокол взаємодії прискорювача з основною платформою комп'ютерної системи.

Оскільки головною метою системи є забезпечення паралельного розв'язання обчислювальних завдань, необхідно базуватися на методах представлення завдань у системі. Найбільш оптимальним є формування максимального рівня абстракції,

при якому потрібно лише представлення математичного апарату, тоді як система автономно здійснює виокремлення паралельних сегментів та їх реалізацію. Проте для численних завдань неможливо реалізувати математичний алгоритм паралельно без застосування певних ручних коригувань. Тому доцільно використовувати поширений підхід високого рівня абстракції, який потребує представлення математичного розв'язання завдання з мінімальним ручним виокремленням сегментів для паралельного виконання. Доречно застосувати схему програмного інтерфейсу, запропоновану професором Стіренко [17], з певними модифікаціями.

Модифікація передбачає розширення набору операцій, а саме:

— загальний розподіл даних;

— загальне збирання даних;

— загальний розподіл завдання;

— локальний розподіл даних;

— локальне збирання даних;

— локальний розподіл завдання.

При цьому перші три операції є експліцитними (доступними для користувача), а останні три — службовими, виконання яких, незважаючи на важливість цього етапу програми, залишається прихованим від користувача.

Операції з даними доцільно модифікувати за аналогією з підходами у технологіях OpenMP та MPI на рівнях паралельної системи з централізованою пам'яттю та розподіленою пам'яттю відповідно. Завдяки специфічним характеристикам функціонування алгоритму аналізу завдання (описаним далі), з'являється можливість автоматизованої обробки доступності даних, а не лише в ручному режимі.

Такий інтерфейс надає широкі можливості для ефективної організації паралельного виконання значної кількості прикладних математичних задач. Як зазначено в дослідженнях [18], більшість сучасних паралельних алгоритмів зводяться до певного набору типових схем (рис. 2.1). Це підтверджується й тим, що до 80% обчислювального навантаження в типових програмних продуктах складають циклічні або ітеративні процеси [18], які й підпадають під такі

структури.

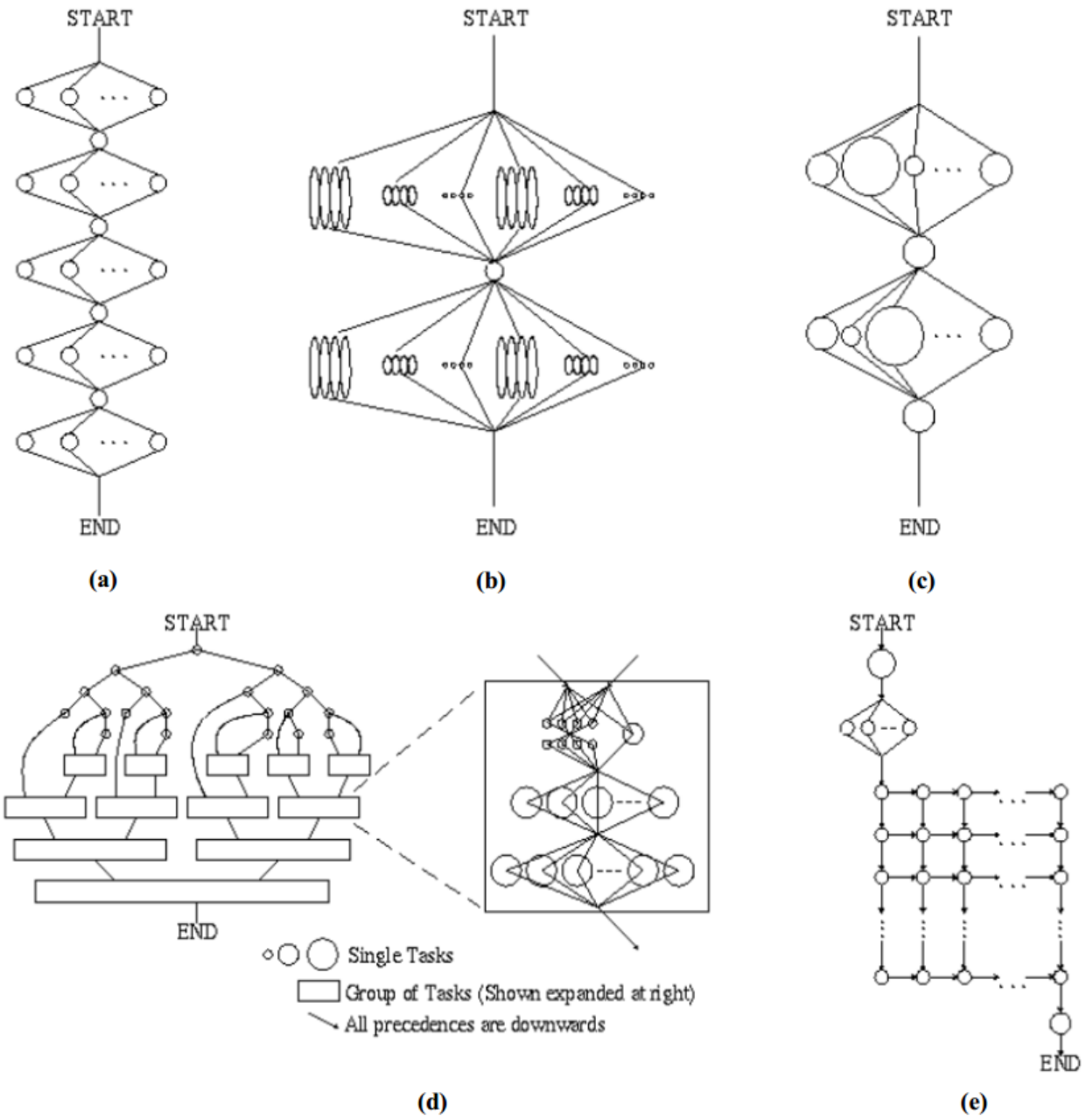

Рисунок 2.1 — Приклади найрозповсюдженіших схем роботи (графів паралельних задач) паралельних програм

Водночас, навіть на цьому рівні абстрагування інтерфейс не охоплює всі можливі типи задач. Наприклад, функціонування розподіленої системи в ролі сервера змінює логіку реалізації, відтак не підлягає опису наведеними схемами. Саме тому зберігається потреба у використанні низькорівневих механізмів

управління, які забезпечують гнучкість в реалізації нетипових сценаріїв.

У сучасних методиках побудови паралельних обчислень вхідні задачі зазвичай описуються через один із двох узагальнених шаблонів паралелізму:

— Data-driven parallelism: кожен обчислювальний елемент виконує однакову послідовність дій, але над своєю окремою частиною загальних вхідних даних. Після завершення обробки результати об’єднуються у спільний вихід;

— Task-driven parallelism: усім обчислювачам передається однаковий набір даних, але кожен з них виконує над цими даними власний, унікальний набір операцій. Кінцевий результат у такому випадку визначається виходом останньої оброблювальної задачі.

Більшість графів, зображених на рисунку 2.1, належать до концепції Data-driven parallelism. У рамках цієї схеми розглядається також поділ на регулярний та нерегулярний паралелізм (рис. 2.1, d).

Регулярний паралелізм передбачає, що кожен потік виконує ідентичний набір інструкцій, незалежно від зовнішніх факторів. Якщо ж спостерігається відмінність в операціях між потоками (через умови, вхідні параметри, ідентифікатори потоків тощо), такий паралелізм визначається як нерегулярний.

У цьому контексті "Task-driven parallelism" завжди відноситься до нерегулярного типу, оскільки виконання задачі варіюється між потоками. Натомість "Data-driven parallelism" може бути як регулярним, так і наближено нерегулярним — у випадках, коли відмінності в операціях мають закономірний характер або виникають як винятки.

Зокрема, схема (рис. 2.1, e) ілюструє приклад нерегулярного Data-driven parallelism.

Беручи до уваги той факт, що більшість прикладних математичних задач, які реалізуються у паралельному середовищі, відповідають саме шаблону "Data-driven parallelism", основна увага в розглянутій системі приділяється його реалізації. При цьому особливо детально аналізуються регулярний режим та умовно-нерегулярний режим (у якому відмінності в поведінці потоків виникають систематично, а не випадково), як це зображено на рисунку 2.1, варіанти b та c.

2.2 Оцінювання вхідної задачі

Виходячи з дослідження попередніх розділів, аналіз вхідного завдання необхідний як для точнішого оцінювання продуктивності вузлів, характерної саме для цього завдання, так і для формування автоматизованої обробки спільних для процесів даних.

В обох випадках насамперед необхідно засобами мови реалізації створити абстрактне синтаксичне дерево вхідного завдання за правилами відповідної мови.

Абстрактне синтаксичне дерево завдання представляє собою скінченну множину вершин та з'єднань між ними, які формують позначену орієнтовану деревоподібну структуру. У цій структурі внутрішні вершини відповідають операторам мови програмування, а листя — операндам. Такі структури використовуються на етапі парсингу програмного коду і необхідні для проміжного представлення програми між конкретним синтаксичним деревом і структурою даних, яка потім застосовується як внутрішнє представлення компілятора або інтерпретатора програми для оптимізації і генерації коду.

Розглянемо для прикладу наступний псевдокод обчислення алгоритму Евкліда для визначення найбільшого спільного дільника двох цілих чисел:

```
while b ≠ 0
if a > b
a := a − b
else
b := b − a
return a
```

Абстрактне синтаксичне дерево цього коду матиме загальний вигляд, приведений на рисунку 2.2.

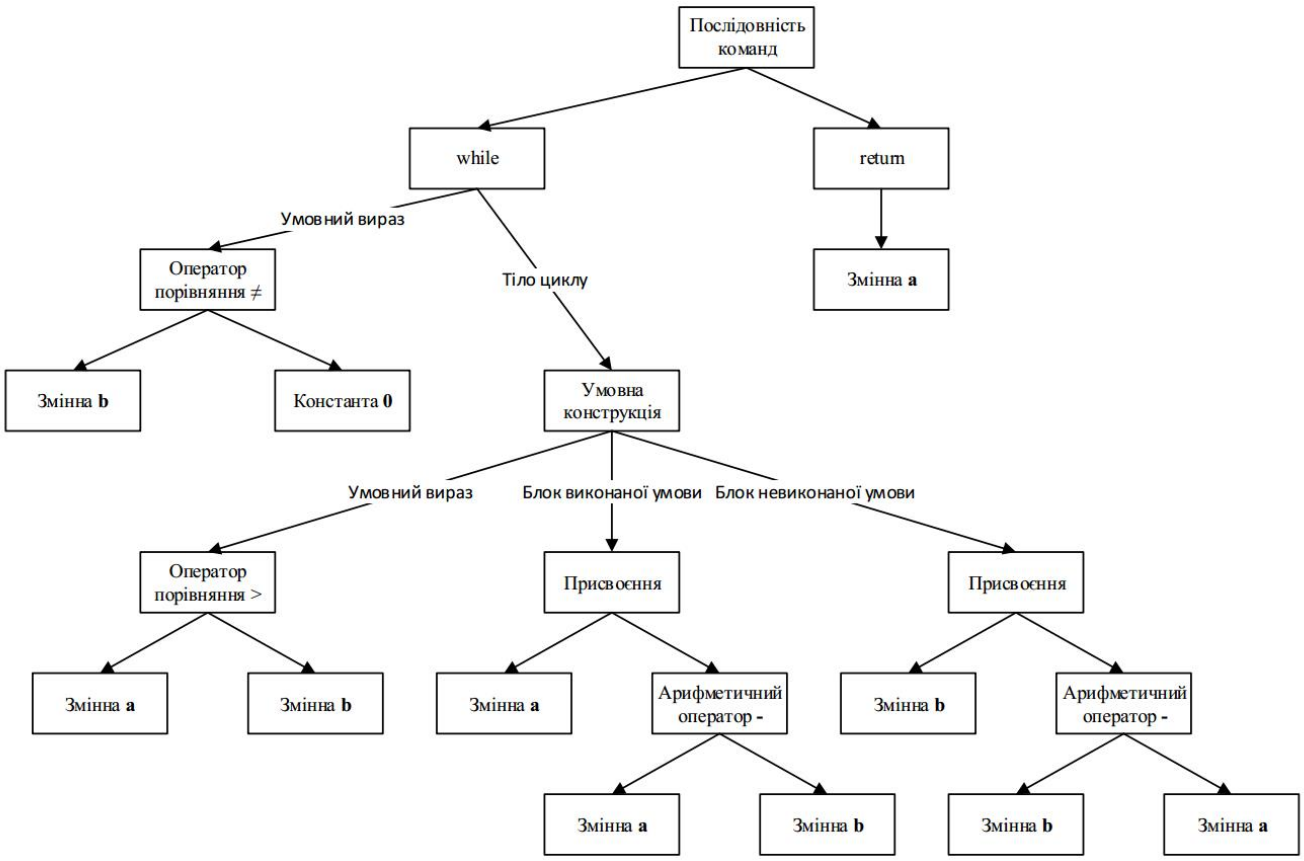

Рисунок 2.2 — Узагальнене абстрактне синтаксичне дерево для наведеного вище алгоритму Евкліда

Враховуючи відповідність завдання переважно шаблону "Data-driven parallelism", можна стверджувати, що кожний кінцевий потік виконуватиме в ідеальному випадку однакові дії. Тоді, з урахуванням того, що в будь-якій сучасній комп'ютерній системі виконання алгоритму зводиться до послідовності елементарних арифметичних операцій [8], можна здійснити приблизну оцінку обсягу таких операцій у завданні. Для цього:

— в графі завдань методом обходу в глибину спочатку визначається наявність циклічних конструкцій та обчислюється їх потужність (максимально можлива кількість задекларованих повних ітерацій, без урахування переривань). Якщо межі циклічної конструкції залежать від обсягу певного набору даних, використовується показник повного обсягу цих даних, оскільки на цьому етапі програми він уже відомий (опис цього наведено далі). За результатами цього кроку отримуємо набір значень $I_1..I_n$, де I —

потужність конкретного циклу, N — порядковий номер циклу;

— далі проводиться повторний пошук в глибину, але лише починаючи з вершин циклічних конструкцій, тобто аналізуються цикли від 1 до N. Здійснюється підрахунок значень C_MUL, C_SUM, C_SUB, C_M, C_DIV — кількості елементарних операцій множення, додавання, віднімання, присвоєння і ділення відповідно, які зустрічаються в кожному циклі. Після завершення кожного обходу кожне значення множиться на відповідний цьому циклу показник потужності I;

— отримані значення коригуються з урахуванням наявності позациклових елементарних операцій шляхом збільшення відповідних значень C_MUL, C_SUM, C_SUB, C_M, C_DIV на кількість виявлених операцій відповідного типу.

Отримані в результаті кінцеві значення показників C_MUL, C_SUM, C_SUB, C_M, C_DIV будуть використані в алгоритмах функціонування інших компонентів, описаних у наступних підрозділах.

За допомогою абстрактного синтаксичного дерева також можна визначити наявність спільних даних. Оскільки операції розподілу даних та розподілу завдання оголошуються окремими функціями в коді, можна стверджувати, що всі дані, які одночасно містяться в дереві в тій гілці, що відповідає за завдання, які підлягають розподілу, та в гілках, які відповідають за попередні конструкції (тобто розташовані в графі на одному рівні ліворуч), потребують копіювання. Ті ж дані, які, крім цього, знаходяться ще й у гілках, виконання яких передбачається після розподілу (тобто розташовані на тому самому рівні графа праворуч), вимагають організації потокобезпечної обробки.

2.3 Опис алгоритму ініціалізації системи

Як уже зазначалося, система являє собою набір розподілених слабозв'язаних вузлів. Система передбачає початкову рівноправність усіх вузлів, тобто будь-який вузол може виступити ініціатором обчислень. Саме з операції ініціалізації вузлом

обчислень розпочинається ініціалізація системи.

Оскільки система може динамічно змінюватись через додавання чи видалення вузлів, жоден вузол початково не володіє повною інформацією про всю систему. Перед початком обчислення вузол має лише дані про ті вузли, яким він може делегувати обчислення. Ця інформація на кожному вузлі зберігається в його власному конфігураційному файлі і представляє собою набори значень Z_i={ CPU_i, IP_i, PORT_i, ACC_i }, де i — порядковий номер під'єднаного вузла, IP_i — адреса вузла i, PORT_i — виділений програмі порт на вузлі i, CPU_i — відсоток потужностей, який може бути використаний програмою на центральному процесорному елементі цього вузла, ACC_i — відсоток потужностей, який може бути використаний програмою на прискорювачі цього вузла.

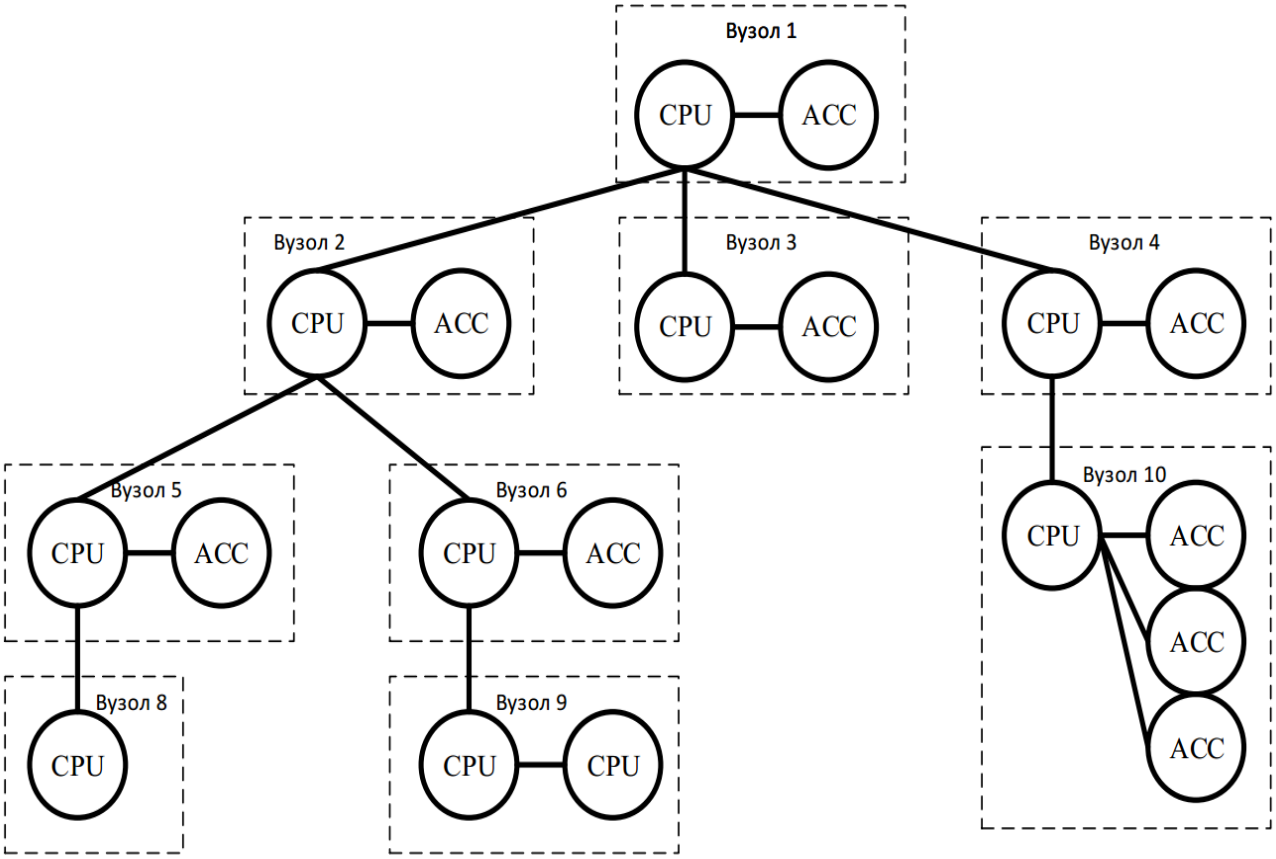

Рисунок 2.3 — Схематичний приклад графу для системи з 10 різноманітних за складом вузлів

На основі наведених даних виконується логістична реорганізація структури

системи таким чином, щоб вона мала вигляд орієнтованого ациклічного дерева. У цій ієрархії кореневою вершиною виступає ініціатор, далі розташовані вузли першого рівня, яким ініціатор делегує обчислення, потім — вузли другого рівня, які отримують завдання від вузлів першого рівня, і так далі. Прискорювальні пристрої (акселератори) розглядаються як окремі логічні одиниці, жорстко прив’язані лише до тих вузлів, у яких вони фізично встановлені.

Наступним етапом є призначення ваг кожному елементу дерева-графа на основі відносної продуктивності його компонентів.

Для зважування ребер необхідно спочатку визначити абсолютну пропускну здатність каналів зв’язку між вузлами, а також внутрішніх інтерфейсів між центральними процесорами та акселераторами (якщо такі присутні).

Щоб оцінити швидкодію інтерфейсів до акселераторів, виконується тестове надсилання та прийом фіксованих за розміром пакетів з різними типами даних, вимірюється час цих операцій. Таким чином, пропускна здатність інтерфейсу WI визначається за формулою (2.1):

$$W_I = \frac{2 * S}{\Delta T_{SEND} + \Delta T_{RECV}}, \tag{2.1}$$

де ΔT_{SEND} – різниця між часом початку та закінчення процесу відправки даних,

ΔT_{RECV} – різниця між часом початку та закінчення процесу отримання даних,

S – обсяг відправлених даних.

Для оцінки пропускної здатності зовнішніх каналів між вузлами пропонується використовувати утиліту traceroute, оскільки її результати менше піддаються впливу змін маршруту порівняно з ping. Значення пропускної здатності каналу між вузлами WC обчислюється за формулою (2.2):

$$W_C = \sum_{i=1}^{N} \left(\frac{S_i}{\sum_{j=1}^{M} Tj} \right) , \quad (2.2)$$

де N – кількість запусків утиліти traceroute,

M – кількість проміжних вузлів на шляху слідування пакету на поточному запуску,

T – час проходження пакетом від одного вузла маршруту до іншого,

S – розмір відправленого на поточному запуску пакету.

Далі ці абсолютні величини переводяться у відносні значення. У цьому випадку вагу ребер визначають не у співвідношенні до сумарного значення, а відносно найкращого каналу або інтерфейсу. Відповідно, відносна пропускна здатність каналу WC визначається за формулою (2.3):

$$W_{CR} = \frac{W_C * 100}{W_{CB}} , \quad (2.3)$$

де W_C – абсолютний показник пропускної здатності каналу,

W_{CB} – абсолютний показник продуктивності найшвидшого із каналів в системі.

Відповідно, відносний показник пропускної здатності інтерфейсу взаємодії з акселератором визначатиметься за формулою (2.4):

$$W_{IR} = \frac{W_I * 100}{W_{IB}} , \quad (2.4)$$

де W_I – абсолютний показник пропускної здатності інтерфейсу,

W_{IB} – абсолютний показник продуктивності найшвидшого із інтерфейсів в системі.

Усі розраховані відносні показники будуть використані як ваги ребер у

графовій моделі системи.

Для зважування вершин необхідно визначити абсолютну обчислювальну продуктивність кожного з елементів вузла. Це здійснюється через оцінку середнього часу виконання п’яти базових операцій: додавання (T_SUM), віднімання (T_SUB), множення (T_MUL), ділення (T_DIV) та присвоєння (T_M). Для кожного елементу (як CPU, так і акселератора) запускається серія обчислень із фіксованою кількістю операцій над випадковими даними з вимкненою оптимізацією компіляції.

Далі, абсолютний показник продуктивності вузла WEA обчислюється за формулою (2.5):

$$W_{EA} = \frac{P * \left(\frac{C_{SUM}}{T_{SUM}} + \frac{C_{SUB}}{T_{SUB}} + \frac{C_{MUL}}{T_{MUL}} + \frac{C_{DIV}}{T_{DIV}} + \frac{C_M}{T_M}\right)}{\Delta L * \Delta t * \Delta M} , \qquad (2.5)$$

де T_{SUM}, T_{SUB}, T_{MUL}, T_{DIV}, T_M – середній час виконання елементом операцій додавання, віднімання, множення, ділення і присвоєння відповідно,

C_{SUM}, C_{SUB}, C_{MUL}, C_{DIV}, C_M – кількості в задачі елементарних операцій додавання, віднімання, множення, ділення і присвоєння відповідно,

P – кількість ядер (або потоків) цього елементу,

ΔL – різниця початкового та кінцевого показників завантаженості елементу,

Δt – різниця початкового та кінцевого показників температури елементу,

ΔM – різниця початкового та кінцевого показників кількості хеш-промахів елементу (для акселераторів не враховується).

Далі необхідно перевести абсолютні показники продуктивності у відносні відсоткові значення. При цьому важливо врахувати, що загальний час обробки підзавдання на вузлі визначається не тільки безпосереднім часом виконання обчислень, але й часом простою цього вузла в очікуванні передачі даних. Тому доцільно модифікувати значення абсолютної продуктивності елементу відповідно

до значення відносної пропускної здатності каналу зв'язку, який веде до цього елементу.

Сума відносних показників повинна становити 100%. Отже, за правилом складних відсотків, визначаємо $W_{(er)}$ – відносний показник продуктивності кожного елементу за формулою (2.6):

$$W_{ER} = \frac{\sum_{i=1}^{N} \left(W_{EA_i} * W_{R_i}\right)}{W_{EA^*} W_R}, \tag{2.6}$$

де N – сумарна кількість елементів в системі,

W_{EA} – абсолютний показник продуктивності елементу,

W_R – відносний показник пропускної здатності каналу зв’язку або інтерфейсу взаємодії з елементом.

Як результат, кожна вершина графу отримує вагу, що відповідає її реальній обчислювальній ефективності з урахуванням пропускної здатності каналів, а ребра — ваги на основі пропускної здатності зв’язку.

На цьому етапі формується повна логістична карта системи, адаптована до конкретного обчислювального завдання (див. рис. 2.4 — умовна ілюстрація на основі структури з рис. 2.3).

Після завершення зважування графової моделі система переходить до стадії планування та запуску обчислень, що відбуваються під контролем балансувальників навантаження на різних ієрархічних рівнях.

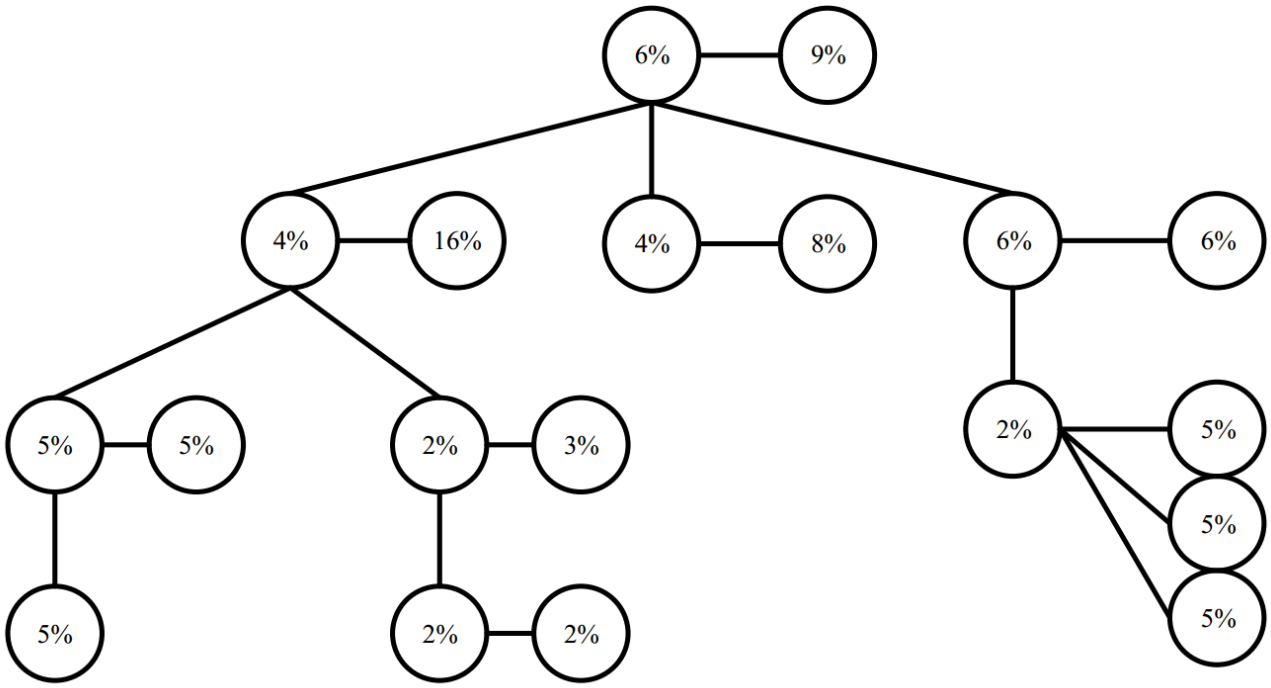

Рисунок 2.4 — Схематичний приклад зваженого графу (вага вузлів враховує вагу ребер)

2.4 Опис алгоритму оптимізації паралельних обчислень на основі балансування навантаження

В сучасній практиці проектування архітектури комп'ютерних систем виокремлюють два фундаментальні підходи до розподілу навантаження: статичний та динамічний методи балансування.

Статичний механізм розподілу навантаження виконує основну роботу на етапі підготовки до фактичного старту паралельної програми. Базуючись на інформації про характеристики системи (кількісні параметри компонентів, їх обчислювальну потужність, взаємозалежності між елементами, пропускну спроможність комунікаційних каналів) та характеристики завдання (число підзадач, зв'язки між ними, обсяги та інтенсивність передач між підзадачами, наявність розподілених ресурсів), механізм балансування згідно з заданим алгоритмом знаходить оптимальний розподіл підзадач між доступними

обчислювальними елементами відповідно до обраного критерію ефективності (як правило, це мінімізація часу виконання або максимізація показника ефективності). Надалі роль балансувальника зводиться до послідовного виконання завдань за розробленим планом без корекцій.

Завдання балансування відноситься до класу NP-повних задач, тому рішення, що його знаходить балансувальник, є лише наближенням до оптимального. При цьому, зазвичай, чим точніше таке наближення, тим більше часових ресурсів потрібно для його знаходження за допомогою складних алгоритмів, що збільшує загальний час роботи системи, що є суттєвим недоліком. Іншим недоліком статичного балансування є те, що при обробці підзадач на вузлах балансувальник не враховує поточні показники стану цих вузлів та хід обчислень. Крім того, рішення, що його формує статичний балансувальник, втратить ефективність, якщо протягом обчислень один із вузлів системи вийде з ладу, що скасовує всі попередні плани обчислень і потребуватиме повторного перебалансування невиконаних завдань.

Проте відсутність контролю системи протягом розрахунків одночасно є і перевагою статичного балансувальника, адже система не витрачає ресурси на контроль та динамічне перебалансування, що дозволяє направити всі потужності виключно на обчислення. Особливо це помітно при балансуванні на рівні розподіленої системи (рис 1.1), оскільки дані контролю стану вузлів повинні передаватися через мережеві канали, які значно повільніші порівняно з внутрішніми каналами кінцевих вузлів.

На відміну від статичного, динамічний балансувальник практично не витрачає час на підготовку попереднього оптимального рішення, а одразу розпочинає призначення підзадач обчислювальним вузлам та організацію передач. В процесі його функціонування виконується постійний контроль параметрів стану системи та черг підзадач, на основі яких приймаються рішення щодо розподілу наступних підзадач між вузлами.

Серед переваг динамічного підходу варто відзначити його значно вищу гнучкість і можливість максимального використання всіх системних ресурсів.

Зазвичай рішення, що їх формує динамічний балансувальник, виявляються якіснішими порівняно з тими, що пропонує статичний. Також при динамічному балансуванні підвищується стійкість системи до збоїв та знижуються витрати на обробку проблемних ситуацій - у випадку втрати зв'язку з одним із вузлів балансувальник просто повторно додасть до черги підзадачі, які були призначені для втраченого вузла.

Недоліком, відповідно, є необхідність виділення певної частки ресурсів (як обчислювальних, так і комунікаційних) для забезпечення процесу контролю вузлів.

Щоб ефективно розв'язати завдання розподілу з врахуванням апаратних характеристик, доцільно переробити архітектуру РКС (рис. 1.1), усунувши абстрактний рівень кластерної системи (адже заявляється, що кожен вузол може також являти собою розподілену неповнозв'язну систему, яка є ширшою концепцією у порівнянні з кластерною та збільшує можливості схеми) і ввівши новий абстрактний рівень -- рівень паралельної системи з відповідним рівнем планувальника паралельної системи. Потреба в цій переробці та її позитивні якості випливають з подальшого викладу процесу розподілу, запропонованого для системи.

Згідно з внесеними корективами в схему роботи РКС одержуємо зміну схеми з рисунку 1.1, показану на рисунку 2.5.

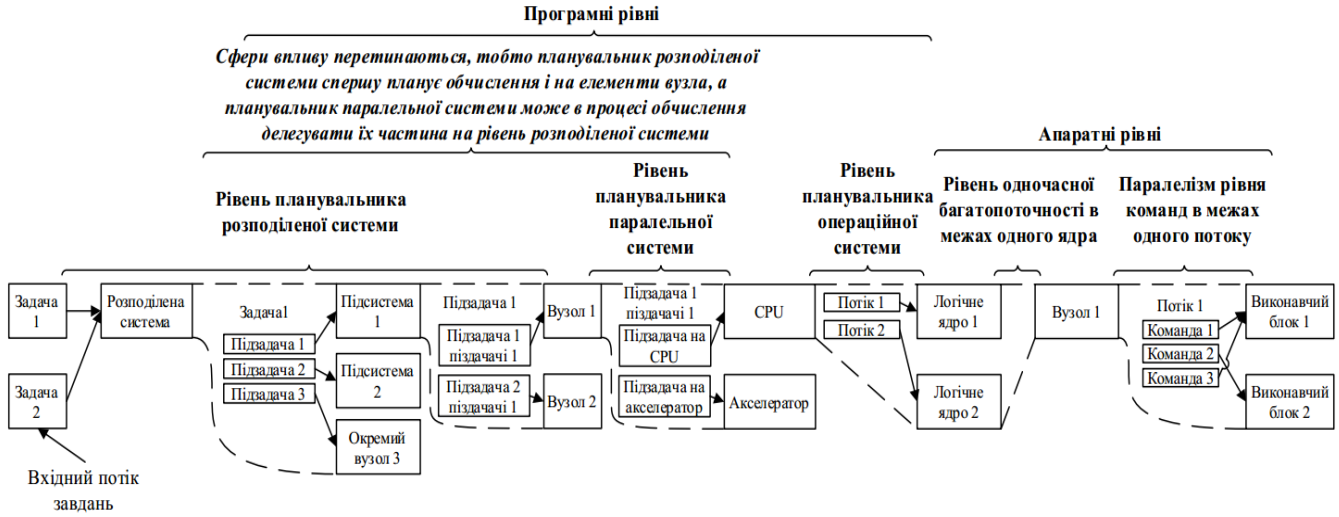

Рисунок 2.5 — Модифікована схема процесу обробки задач в розподілених гетерогенних комп’ютерних системах

В дослідженні пропонується використовувати паралельно як статичний, так і динамічний розподіл на різних стадіях опрацювання завдання на різних рівнях системи (стадії розподіленої системи та стадії паралельної системи, а також між ними) відповідно до схеми, показаної на рисунку 2.5.

Як зазначалось раніше, головними мінусами статичного розподілу є:

— неточність отриманого розв'язку;

— додаткові часові затрати на стадії підготовки;

— слабка стійкість до несправностей.

Враховуючи, що система створена для здійснення паралельних програм, де паралелізм виокремлений на базі послідовних паралельних секцій (рис. 2.1) і відповідає шаблону «Data-driven parallelism», то відповідно до проведених попередньо досліджень можна стверджувати, що коли на стадії розподіленої системи виконати початковий статичний розподіл, де рівень використання вузла буде залежати від кількості даних, опрацювання яких планується на цьому вузлі, а кількість даних буде визначатись відносною продуктивністю вузла в системі, то можна отримати досить ефективне початкове наближення для розв'язання завдання розподілу навантаження за нетривалий період часу.

Відповідно до характеристики головних функцій для розпаралелювання програми, здійснюються перші три з вказаних функцій, а саме «Загальне розділення даних», «Загальний збір даних», «Загальне розділення задачі», при цьому їх використання тісно взаємопов'язане.

На вузлі-ініціаторі обчислювальних процесів спочатку здійснюється функція загального розділення даних. Параметрами цієї функції виступають дані, які будуть розподілятись між вузлами. При цьому зрозуміло, що розділення стосується тільки позначених даних, притому таких, які характеризуються векторною структурою в пам'яті (зокрема, масиви, матриці, списки, символи рядків, рядки документів, пікселі зображення та інше); коли тип даних не можна розділити, такі дані просто дублюються.

Розподіл векторних даних відбувається на базі параметрів ваги вершин

графу-карти системи, отриманих на попередній стадії. Ваги ребер не беруться до уваги на цьому етапі, оскільки вони вже враховані на стадії створення відносних ваг вершин (8). Тому розмір частини даних SP для кожного вузла визначається згідно з наступною формулою (2.7):

$$S_P = \lceil S * W \rceil , \tag{2.7}$$

де S – розмір векторної структури даних, яка підлягає поділу,

W – вага (у відсотках) вузла на карті системи.

Паралельно в системі, відповідно до числа вузлів, формується комплект динамічних об'єктів, клас яких може додатково (хоча не обов'язково) успадковуватись від класу паралельного завдання або потоку. Ці об'єкти включають передусім поля-ідентифікатори (які відповідають ідентифікаторам вузлів в системі), а також динамічні поля, до яких включаються дані, що мають передаватись вузлам.

Кожен вхідний вектор, який має передаватись з розділенням, представляє відповідну частину розміром SP загального вектора в програмі, тобто від вхідного вектора послідовно відділяються частини, розмірність яких відповідає поточному значенню SP зі списку, створеного на попередній стадії.

Подальшою загальною функцією є «Загальне розділення задачі». Ця функція отримує функцію, яку розробник-користувач хоче виконати в паралельному режимі, функцію з кодом, який має виконуватись на акселераторі, а також ідентифікатор завдання. Все це дублюється у відповідні динамічні об'єкти з попереднього списку, при цьому кожен такий об'єкт вже містить ідентифікатор, який передається паралельній функції та акселераторній функції. Це потрібно, зокрема, для того, щоб коли розробник планує використання розподілених даних у своїй програмі, він міг единим кодом здійснювати опрацювання цих даних залежно від ідентифікатора вузла (якщо така потреба передбачається).

Заключним універсальним компонентом системи постає «Агрегація інформаційних ресурсів». Функціональність даного модуля полягає у специфікації

типів даних, що підлягають консолідації на головному обчислювальному вузлі після завершення розрахункових циклів. Крім того, модуль описує різноманітні стратегії збору інформації, включаючи алгоритми редукування даних. Присутність функції «Агрегація інформаційних ресурсів» гарантує її самочинну активацію по закінченні виконання паралельних та апаратно-прискорених процедур у межах динамічного об'єкта. Створення таких функціональних компонентів реалізується в рамках відповідних динамічних структур. Завершення цього процесу означає досягнення системою стану готовності для запуску обчислювальних процедур.

Адаптивний розподіл ресурсів характеризується наступними перевагами:

безпосередня корекція під час виконання недосконалостей у розв'язанні завдань балансування навантаження, що були визначені статичним алокатором;

кардинальне підвищення толерантності системи до збоїв, що призводить до мінімізації втрат навіть у випадку повної втрати зв'язку з усіма обчислювальними вузлами;

практичну реалізацію проаналізованої в попередньому розділі інноваційної методології відстрочених обчислень у більш елементарній формі, ніж це було б можливо реалізувати на стадії статичного проектування.

Адаптивний алокатор також забезпечує ініціалізацію та координацію обчислювальних процесів. Його функціонування локалізується на периферійних вузлах, що згідно зі схематичним представленням на рисунку 2.5 відповідає стадії паралельної архітектури. Одночасно функціонує централізаційний механізм, який у специфічних режимах роботи дозволяє часткове переміщення на вищий абстракційний рівень (відповідно до стадії розподіленої архітектури).

На початковій фазі адаптивний алокатор кожного обчислювального вузла в контексті методології відстрочених обчислень направляє головному координатору обчислювальних процесів (прямо або через проміжні елементи) запит на отримання специфічного для вузла-заявника динамічного об'єкта з колекції динамічних структур, що була охарактеризована попередньо. Такі запити акумулюються на головному вузлі у вигляді асинхронної черги зворотних викликів. Враховуючи, що інформаційні масиви, призначені для передачі вузлам, вже структуровані таким

способом, щоб врівноважити розмір інформаційного пакета для конкретного вузла з пропускною здатністю комунікаційного каналу до цього вузла, черговість надання даних не є критичною. Програмною реалізацією передбачено, що одразу після повної підготовки динамічного об'єкта зі списку для передачі на відповідний вузол, з асинхронної черги запитів буде вилучено відповідний запит і через механізм зворотного виклику буде здійснено передачу серіалізованого динамічного об'єкта на цільовий вузол (безпосередньо або через проміжні елементи, на яких реалізовано функції подальшого маршрутизування запитів).

Після отримання периферійним вузлом відповідного йому динамічного об'єкта відбувається його десеріалізація та подальша декомпозиція описаного в об'єкті завдання на дрібніші субзавдання. Алгоритм такої декомпозиції в основному відтворює принципи розподілу завдань на стадії статичного алокування (таким чином можна констатувати, що статичний розподіл також частково реалізується на стадії паралельної архітектури), проте фрагменти даних характеризуються однорідністю. Декомпозиція відбувається на кількість частин, що суттєво перевищує число ядер/потоків центрального обчислювального елемента даного вузла. Усі новостворені внаслідок декомпозиції динамічні об'єкти інтегруються в паралельну чергу виконання. Початкові P об'єктів, де P відповідає кількості ядер/потоків центрального обчислювального елемента, миттєво передаються на обробку, що полягає в активації потокової функції, специфікованої у відповідному динамічному об'єкті.

Важливо зауважити, що у випадку наявності акселератора на вузлі, від координатора надходять два динамічних об'єкти: перший призначений для обчислень на центральному обчислювальному елементі, другий -- для операцій на акселераторі. Алгоритм обробки об'єкта для центрального обчислювального елемента був описаний вище. Алгоритм обробки об'єкта для акселератора базується на аналогічній схемі декомпозиції, але обмежується трьома об'єктами.

Архітектурою системи передбачено виділення двох потоків для взаємодії з акселератором. Обґрунтування такого рішення полягає в плануванні використання переважно інтерфейсу PCI Express, що функціонує в послідовному

повнодуплексному режимі. Таким чином, два потоки необхідні для сценарію, коли один потік виконує трансмісію власних даних, тоді як інший одночасно здійснює прийом результатів. Очевидно, що вірогідність такого сценарію є обмеженою.

Крім того, на кожному вузлі функціонує спеціалізований потік-аналізатор, відповідальний за здійснення моніторингових операцій щодо стану паралельної системи та її складових. З попередньо встановленою константною періодичністю (наприклад, кожні 10 секунд) цей потік завершує блокування та проводить аналіз характеристик системних елементів (центральних процесорів та акселераторів). Аналіз включає: варіацію системного параметра навантаження елемента між двома найновішими вимірюваннями, флуктуацію тактової частоти між двома найновішими вимірюваннями (з урахуванням того, що за рішенням планувальника операційної системи елемент може активувати режим Turbo лише на певному етапі обчислень), зміну температури елемента між двома найновішими вимірюваннями, статус черг обробки потокових та акселераторних функцій. Ці дані аналізатор транслює адаптивному планувальнику, після чого відповідний потік-аналізатор переходить у стан блокування до наступного циклу вимірювання.

Динамічний планувальник, відповідно до отриманих від спостерігача даних, ухвалює ряд рішень щодо подальшого опрацювання. У разі недосягнення встановленого в конфігураційному файлі вузла параметра цільової завантаженості елемента, на виконання спрямовується, залежно від дельти між поточною і цільовою завантаженістю, один або кілька динамічних об'єктів з відповідної черги елемента, або ж, за наявності заблокованих об'єктів, що вже проходили обробку, в пріоритетному порядку відновлюється їхня обробка. У випадку вичерпання черги вузол генерує сигнал готовності до прийняття додаткових розрахункових операцій (механізм цього процесу деталізовано далі).

При перевищенні встановленого в конфігураційному файлі вузла параметра цільової завантаженості елемента, залежно від дельти між поточною і цільовою завантаженістю, призупиняється виконання одного або кількох динамічних об'єктів з тих, що перебувають в обробці. Ініціюється міжвузловий перерозподіл (алгоритм якого деталізовано далі).

За оптимальних параметрів завантаженості аналізується варіація частот. При зростанні частот реалізуються дії, ідентичні описаним у пункті 1. При зниженні частот реалізуються дії, ідентичні описаним у пункті 2.

За стабільності частот оцінюється температурна варіація. При зниженні температури щонайменше на 10 градусів і поточному значенні менше 60 градусів реалізуються дії, ідентичні описаним у пункті 1. При підвищенні температури щонайменше на 10 градусів і поточному значенні понад 60 градусів реалізуються дії, ідентичні описаним у пункті 2. Слід підкреслити, що температурні параметри носять орієнтовний характер і мають калібруватися відповідно до попередніх спостережень.

За температурної варіації, що не перевищує 10 градусів, жодних коригувань не відбувається, функціонування вузла та його компонентів продовжується у штатному режимі.

Аналізатор може активуватися також в екстреному режимі при надходженні запиту на уточнення параметрів (деталізація далі).

Адаптивний розподіл також частково реалізується (з найнижчим пріоритетом, тобто вищим пріоритетом характеризуються субзавдання, вже локалізовані на вузлі) між вузлами, відповідно до стадії розподіленої системи. Для цього застосовується вищезазначений механізм перерозподілу, що полягає в делегуванні розрахункових операцій між вузлами-виконавцями.

Проблематика трансферу процесів у межах розподіленої системи характеризується надзвичайною складністю і наразі не має комплексного розв'язання. Це обумовлюється необхідністю передачі разом з процесом його повного контексту, стану стеку, даних, зіставлення адресного простору з урахуванням потенційної гетерогенності архітектур вузлів. Це стосується як процесорних елементів, так, особливо, акселераторів. Тому пропонується механізм, що передбачає не трансфер частини процесу, що вже виконується, а делегування частини розрахункових операцій, що знаходяться в черзі очікування.

В охарактеризованому раніше алгоритмі функціонування потоку-аналізатора вузла передбачено сценарій переходу вузла в режим готовності до прийняття

додаткових розрахункових операцій. Враховуючи, що в системі реалізована часткова, а не повна зв'язність, сигнал готовності вузол транслює головному координатору, де такі сигнали консолідуються в уніфікований список, упорядкований за продуктивністю вершин (тобто ваговими коефіцієнтами вузлів системного графу), від найпотужнішого до найслабшого. Головному координатору також транслюються сигнали перевантаженості вузлів, тобто запити від вузлів на делегування частини їхніх розрахункових операцій іншим вузлам. При надходженні запиту на делегування розрахункових операцій, головний координатор аналізує стан списку вузлів, готових до прийняття розрахункових операцій, і за наявності відповідних вузлів обирає першочерговий (найпотужніший) та транслює йому верифікаційний сигнал, що екстрено активує аналізатор цього вузла для перевірки актуальних параметрів (з метою підтвердження релевантності сигналу готовності, тобто відсутності змін параметрів завантаженості вузла з моменту отримання останнього сигналу). При підтвердженні верифікацією готовності вузла до прийняття розрахункових операцій активується механізм міжвузлового делегування розрахункових операцій, що буде деталізовано далі. При негативному результаті верифікації сигнал від цього вузла елімінується зі списку сигналів і здійснюється перевірка наступного вузла за списком. Додатково за аналогією можна реалізувати верифікацію актуальності делегування і з боку вузла-заявника.

2.5 Висновки до розділу 2

В даному розділі було представлено характеристику організації структури та режиму функціонування розподіленої гетерогенної обчислювальної системи при роботі з довільними обчисленнями. Складовими вузлами такої системи виступають довільні паралельні обчислювальні системи, які також можуть містити акселератори. Представлений підхід нерозривно пов'язаний з методом розподілу навантаження на різних абстрактних стадіях системи, що є ключовим аспектом ефективного функціонування гетерогенної розподіленої системи.

Для оптимальної роботи планувальників запропоновано комплексне оцінювання як вхідного завдання, так і всіх компонентів системи, причому оцінювання елементів враховує особливості завдання, яке розв'язується в системі.

Передача даних в системі реалізована в контексті технології відкладених обчислювальних операцій, зокрема у варіанті відкладених асинхронних передач, з централізацією їх збору та механізмом перенаправлення передач між вузлами. Планування таких передач та їх організація покладені на планувальник нижчого рівня, тобто рівня операційних систем вузлів.

3 РЕАЛІЗАЦІЯ ОПТИМІЗОВАНИХ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ В РОЗПОДІЛЕНИХ ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

3.1 Обґрунтування вибору засобів реалізації прототипів

На підставі детального аналізу, представленого в попередніх частинах дослідження, можна стверджувати, що ключовими інструментами програмного забезпечення, що знаходять широке застосування при створенні ПЗ для розподілених обчислювальних архітектур, виступають технології MPI (які функціонують на рівні розподіленої архітектури, іноді у поєднанні з OpenMP для паралельних систем, або використовуються як самостійне рішення) та Intel TBB. Поза межами основного недоліку, детально розглянутого в початковому розділі, який пов'язаний із спрямованістю цих технологій на однорідні повнозв'язні архітектури та відсутністю інтегрованої підтримки акселераційних пристроїв на рівні планування та розподілу обчислювальних процесів, можна ідентифікувати додаткові обмежувальні фактори, спричинені застосуванням даних технологічних рішень.

Попри високу продуктивність програмних рішень, створених із залученням цих інструментів, та загальну зручність їх практичного використання, головною проблемою з погляду програмування залишається застаріла орієнтація їхньої концепції, яка з плином часу стає все більш відчутною. Серед найважливіших аспектів, що засвідчують цю тенденцію, варто виокремити три взаємозалежні компоненти.

OpenMP та MPI були первісно розроблені для застосування з мовами Fortran та C, тоді як Intel TBB призначався для мови C++. Fortran на сьогоднішній день практично повністю втратив свою значущість у галузі розробки прикладних програмних продуктів, а використання мови C перемістилося головним чином у сферу системного програмування та підтримки наявного програмного коду. Ці програмні мови поступово втрачають свою популярність та актуальність у створенні прикладного програмного забезпечення, що підтверджується

статистичними відомостями за останні роки, зібраними такими платформами як Google, GitHub, StackOverflow.

Особливість OpenMP та передусім MPI полягає в їхній направленості на процедурну методологію програмування. Базові принципи обох технологій формувалися в період, коли процедурна парадигма переважала в прикладному програмуванні. Сучасні стандарти передбачають застосування значно складніших концептуальних методик --- об'єктно-орієнтованого, функціонального та реактивного підходів. Незважаючи на постійну еволюцію цих технологій, потреба в забезпеченні сумісності з попередніми версіями ускладнює повноцінне впровадження сучасних методологій програмування.

Стосовно Intel TBB, слід відзначити його нерозривний зв'язок із мовою C++. Хоча C++ продовжує залишатися потужним та високопродуктивним інструментом, який постійно розвивається та впроваджує інноваційні підходи до створення практичного програмного забезпечення і зберігає статус еталону для високонавантажених обчислень, щорічно з'являються нові мови програмування, які наближаються та іноді перевершують C++ у питаннях продуктивності, що підтверджують наукові дослідження [15-16]. Водночас C++ залишається складною мовою, що потребує високої кваліфікації розробників і певною мірою ускладнює використання новітніх парадигм (це пов'язано з тим, що під час створення основ C++ впровадження таких парадигм не планувалося, а через проблеми зворотної сумісності їх реалізація не завжди характеризується елегантністю та простотою) порівняно з більш сучасними мовами. Подібна ситуація спостерігається з Intel TBB --- незважаючи на те, що це потужний набір інструментів для паралельного програмування (проте обмежений системами зі спільною пам'яттю), наступна проблема також стосується і його.

Присутність низького рівня абстракції призводить до необхідності створення значного проміжного рівня в програмному комплексі для організації взаємодії між рівнем бізнес-логіки та комунікаційним рівнем, адаптації інтерфейсів тощо. Характерним прикладом є MPI, де найвищою абстракцією для передачі даних виступає попередньо визначена структура. Усе це вимагає від програміста

самостійного вирішення питань, високої компетентності та створює широкі можливості для потенційних помилок.

У випадках, коли система включає використання акселераційних пристроїв, мова програмування прототипів має забезпечувати інструменти (вбудовані або зовнішні) для організації обчислень на цих пристроях. Наприклад, якщо акселераторами виступають графічні процесори, необхідна підтримка технологій OpenCL або CUDA у вигляді бібліотек, сумісних із обраною мовою.

У даному дослідженні запропоновано реалізацію описаної технології за допомогою мови C# (зокрема, із використанням WCF (Windows Communication Foundation) та TPL (Thread Parallel Library) в рамках платформи .NET Framework від Microsoft). У цільових тестових конфігураціях планується застосування графічних процесорів як акселераторів, тому для їх інтеграції використовується бібліотека OpenCL.

Такий вибір має обґрунтовані переваги, оскільки C# представляє собою сучасну високорівневу мову програмування з підтримкою множинних парадигм, яка надає весь потрібний функціонал для реалізації запропонованого рішення.

Серед реалізованих функціональностей присутня автоматизована корекція гранулярності паралелізму (по суті — її оптимізація до найбільш ефективного розміру для кожного специфічного вузла в межах кожного індивідуального розрахунку). Попередні дослідження продемонстрували, що при розробці програмних рішень із зменшеною гранулярністю паралелізму засобами C# можливо досягти покращених показників продуктивності у порівнянні з C++ та OpenMP.

Технологічна платформа WCF забезпечує ефективну комунікацію та реалізацію делегованих розрахунків (включаючи відстрочені) у розподілених обчислювальних архітектурах, одночасно дозволяючи інтеграцію гетерогенних компонентів, зокрема систем на базі процесорів Intel та AMD з різноманітними операційними платформами тощо. Додатково, WCF пропонує розширений асортимент оптимізованих протоколів для інформаційного обміну між вузлами у форматі SOAP (Simple Object Access Protocol), включаючи SOAP на основі TCP

(Transmission Control Protocol), UDP (User Datagram Protocol), HTTP (HyperText Transfer Protocol), Message Queues та альтернативні протоколи.

Виконання програмних продуктів, створених з використанням .NET Framework, здійснюється в середовищі віртуальної машини CLR (Common Language Runtime), що може дещо знижувати продуктивність, проте розширює діапазон сумісних мов програмування, забезпечує ізоляцію програмного коду від безпосередньої взаємодії з системою, створюючи уніфіковану модель пам'яті для всіх платформ, гарантуючи незалежність від архітектури процесорів вузлів тощо. За необхідності використання в розподіленій архітектурі вузлів з різними операційними платформами можна застосовувати розширену версію .NET Framework — .NET Core, яка орієнтована на міжплатформну сумісність.

Водночас .NET Framework надає всі необхідні інструменти для взаємодії з API операційної платформи.

Це гарантує незалежність реалізації програмних рішень від версії та виробника компілятора. Наприклад, для мови C++ існують компілятори GNU, компілятори від корпорацій Intel, Microsoft та інших виробників. Функціонування програм може варіюватися залежно від того, яким компілятором вони були скомпільовані. При цьому відсутні гарантії, що на всіх вузлах архітектури буде встановлено ідентичний компілятор C++ однієї версії. Для мови C# доступний єдиний офіційний компілятор від Microsoft, тому проблематика може виникнути виключно через невідповідність версій, але не через загальну несумісність різних компіляторів.

Мова програмування C# підтримує всі розглянуті формати відстрочених розрахунків, включаючи асинхронні паралельні обчислення. У технологічній платформі WCF від початку інтегровано механізми відстрочених передач інформації. C# широко підтримує застосування офлоадних обчислень завдяки наявності підтримки бібліотеки OpenCL. Ця мова також надає можливість створення динамічних структур (DynamicObject та ExpandoObject зі стандартного пакету System.Dynamic), що спрощує процес розробки. З точки зору програмування, робота з цими структурами значно зручніша порівняно з роботою

з аналогічними примітивами в C++. Зручність розгортання програмних рішень, розроблених за допомогою C# WCF у хмарних середовищах, дозволяє проводити більш різноманітне тестування з різними комбінаціями конфігурацій тестових цільових розподілених архітектур.

3.2 Створення контрольних прототипів

З метою здійснення об'єктивного аналізу дієвості пропонованої методології виникає потреба у створенні відповідних програмних реалізацій у контексті моделювання функціонування справжніх додатків, призначених для координації паралельних процесів обчислення в реально існуючих розподілених неоднорідних обчислювальних архітектурах, що включають апаратні прискорювачі. У рамках даних програмних архітектур необхідно здійснити вирішення широкого спектру математичних і практично орієнтованих завдань із застосуванням різноманітних початкових параметрів задля виявлення взаємозв'язків та проведення неупередженого оцінювання продуктивності цих програмних засобів.

Крім того, важливим аспектом є співставлення запропонованої методології з певними базовими взірцями — програмними додатками для координації паралельних обчислювальних процесів у реальних розподілених неоднорідних комп'ютерних архітектурах з апаратними прискорювачами, які також здійснюють розв'язання аналогічних математичних та прикладних завдань із використанням різних початкових параметрів, проте без залучення запропонованих методик та стратегій оптимізації. Розбіжність у параметрах продуктивності, коефіцієнтах прискорення та показниках ефективності між даними контрольними програмними моделями буде безпосередньо демонструвати фактичну результативність пропонованої оптимізації.

Відповідно до визначених вимог, було створено набір із чотирьох програмних модельних архітектур для організації паралельних обчислювальних процесів у розподілених неоднорідних комп'ютерних системах з апаратними прискорювачами, із залученням різноманітних сучасних інструментів, які

розглядалися у першому розділі, а також із впровадженням запропонованої технологічної методології. Основні характеристики даних програмних комплексів подано в таблиці 3.1.

Таблиця 3.1 — Характеристики тестових та контрольних програмних комплексів

№ п. п.	Мова реалізації	Технологія організації паралелізму на рівні розподіленої системи	Технологія реалізації паралелізму на рівні паралельних систем	Технологія реалізації акселераторних обчислень	Підхід до балансування навантаження	Призначення
1	C#	WCF	TPL	OpenCL	Запропонована технологія	Тестовий
2					Лише засобами застосованих технологій та операційних систем вузлів	Контрольний
3	C++	MPI	OpenMP			
4			Intel TBB	Intel TBB		

Як зазначалося раніше, передача даних негативно впливає на відповідні показники систем із локальною пам'яттю. Крім того, як було відмічено в другому розділі, обмін інформацією між вузлами через глобальну мережу не завжди відбувається за однакові проміжки часу. Тому для визначення середніх діапазонів відхилення затримки додатково проводяться вимірювання часу передачі даних шляхом обчислення різниці між моментом відправлення та моментом отримання.

3.3 Опис тестової системи

Створення тестових платформ було здійснено за допомогою інструментарію хмарної екосистеми Google Cloud Console. Дане середовище надає можливості

формування віртуальних обчислювальних машин, зокрема конфігурацій із графічними акселераторами, у різноманітних географічних регіонах. Така функціональність виявляється особливо доречною для здійснення експериментальних досліджень, оскільки дослідження концентрується саме на загальних розподілених архітектурах, де передбачається використання різноманітних обчислювальних елементів з будь-якої географічної точки планети.

На всіх обчислювальних вузлах сформованих розподілених неоднорідних комп'ютерних архітектур було встановлено наступне уніфіковане програмне забезпечення:

— системне програмне забезпечення: Windows Server 2012 R2 Datacenter x64;

— .NET Framework 4.7;

— інструментарій компіляції C++ Microsoft Visual C++;

— OpenCL 2.2;

— OpenMP 3.0;

— Microsoft MPI 9.0.1.

Конфігураційні характеристики створених розподілених неоднорідних обчислювальних архітектур систематизовано в таблиці 3.2.

Таблиця 3.2 — Параметри цільових тестових систем

Система 1 (тестова)				
Вузол	Розміщення	CPU	GPU	ОЗП
1	Лос-Анджелес, США	4-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla K80	15 Gb
2	Бельгія	8-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla P100	30 Gb
3	Гонконг, КНР	16-ядерний вірутальний процесор архітектури Intel SkyLake	2 шт. Nvidia Tesla K80	60 Gb
4	Сідней, Австралія	32-ядерний вірутальний процесор архітектури Intel SkyLake	-	120 Gb
Система 2 (контрольна)				
Вузол	Розміщення	CPU	GPU	ОЗП
1	Лос-Анджелес, США	8-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla K80	30 Gb
2	Бельгія	8-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla K80	30 Gb
3	Гонконг, КНР	8-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla K80	30 Gb
4	Сідней, Австралія	8-ядерний вірутальний процесор архітектури Intel SkyLake	Nvidia Tesla K80	30 Gb

Карту розміщення та принципових (заданих в рамках моделі) зв'язків між вузлами представлено на рисунку 3.1.

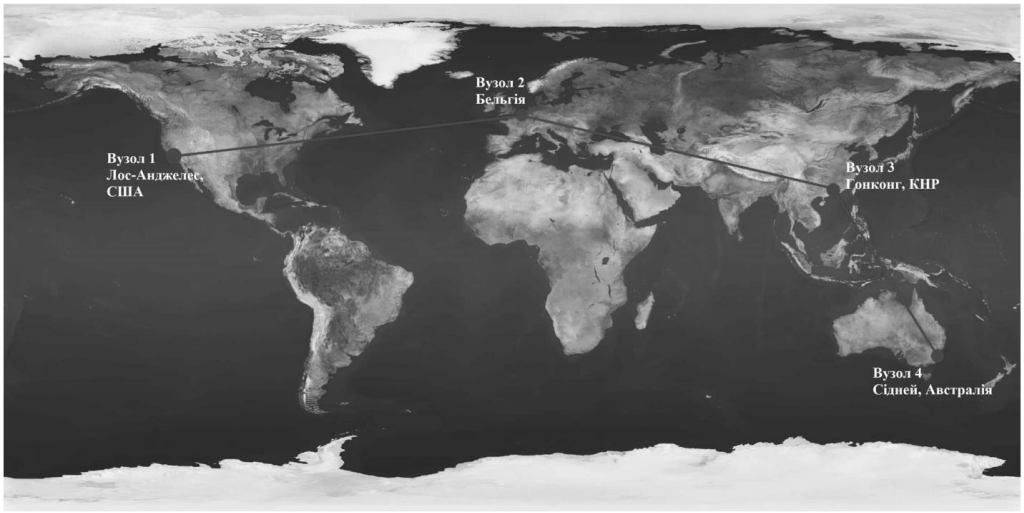

Рисунок 3.1 — Карта розміщення вузлів тестових систем з відображенням умовних логістичних зв’язків між їх вузлами

Згідно з топологічною структурою, зображеною на рисунку 3.1, у всіх експериментальних сценаріях як вузол-координатор обчислювальних процесів було визначено вузол 2 (розташований у Бельгії). При цьому враховано відсутність безпосереднього з'єднання для транспортування даних між вузлом 2 та вузлом 4 (розташованим у Сіднеї), що спричиняє необхідність здійснення комунікації через проміжний вузол 3 (розташований у Гонконзі).

3.4 Опис процесу тестування

Процедура експериментального тестування включає багаторазове послідовне ініціювання всіх модельних програмних прототипів із фіксуванням тривалості їх функціонування, та подальшим встановленням середньої тривалості роботи кожної програмної реалізації як середнього арифметичного всіх часових показників її функціонування.

Додатково, відповідно до попередньо описаної стратегії коротких асинхронних паралельних вимірювань часових міток початку та завершення процесів передачі інформації, здійснювався збір часових параметрів тривалості комунікаційних процесів для аналізу їхнього впливу. Також проводився постійний моніторинг підсумкових середніх параметрів завантаженості графічного акселератора (GPU) кожного обчислювального вузла обчислювальними завданнями для визначення обсягу обчислювальних операцій, який система вважає доцільним виконувати на прискорювачі.

Часові параметри безпосередньої швидкодії програмних засобів для зручності аналізу було трансформовано в параметри абсолютного коефіцієнта прискорення паралельної програмної реалізації. Даний коефіцієнт являє собою співвідношення тривалості виконання послідовного варіанту програми до тривалості виконання паралельної програми на P обчислювальних елементах і демонструє, у скільки разів скоротилася тривалість виконання програми в паралельній архітектурі [17]. Обчислення коефіцієнта прискорення kSU здійснюється за формулою 3.1:

$$k_{su} = \frac{T_1}{T_P} \ , \tag{3.1}$$

де T_1 – час роботи послідовного варіанту програми,

T_P – час роботи паралельного варіанту програми на P обчислювачах (потоках, вузлах, тощо).

Іншим важливим показником ефективності роботи паралельних систем та алгоритмів є коефіцієнт ефективності k_{EF} [17], який вираховується на основі коефіцієнту прискорення, та показує сумарну ефективну загрузку всієї системи (3.2).

$$k_{EF} = \frac{T_1}{T_P * P} * 100\% \ , \tag{3.2}$$

де T_1 – час роботи послідовного варіанту програми,

T_P – час роботи паралельного варіанту програми на P обчислювачах (потоках, вузлах, тощо).

Проте, оцінювання коефіцієнта ефективності для паралельних розподілених неоднорідних обчислювальних архітектур із акселераторами не забезпечує достатньо точного відображення, оскільки такі потужні обчислювальні компоненти, як графічні процесори, що функціонують як акселератори, за формулою враховуються в тому ж обсязі, що й звичайне окреме ядро чи потік центрального процесора.

Тому пропонується застосування непрямих параметрів, зокрема коефіцієнта завантаженості акселератора (який становить відсотковий показник обчислень, виконаних усіма акселераторами системи відносно загального обчислювального процесу), а також коефіцієнта використання мережевих ресурсів (який відображає відсотковий показник сумарної тривалості, витраченої під час розв'язання завдання системою на обмін інформацією між вузлами та компонентами, від загальної тривалості вирішення цього завдання).

Окремо для розподіленої архітектури також необхідно розрахувати варіативність затримки, тобто відсоткові параметри відхилень тривалості передач ідентичних даних між однаковими обчислювальними елементами. Це можна реалізувати, встановивши часові заміри у відповідних сегментах програмного коду, і оскільки на початку планується багаторазове ініціювання програми з ідентичними вхідними параметрами, певними з цих етапів можна скористатися, вважаючи їх аналогічними, і відповідно оцінювати параметри часових витрат на передачу інформації на цих етапах.

Експериментальне дослідження проводилось на декількох поширених прикладних завданнях, кожне з яких має особливу цінність для випробувань, оскільки відповідає певному типовому сценарію в практичній організації паралельних програмних рішень у розподілених архітектурах.

Обчислення добутку двох квадратних матриць являє собою класичне завдання для паралельного програмування, яке ефективно адаптується до розпаралелювання.

Добуток MC двох квадратних матриць MA та MB розмірністю N×N елементів складається з усіх можливих комбінацій скалярних добутків рядків матриці MA і стовпців матриці MB. Елемент матриці MC з індексами i, j представляє скалярний добуток i-го рядка матриці A і j-го стовпця матриці B.

Враховуючи це, можна побудувати паралельний алгоритм множення двох матриць за принципом поділу однієї з них за стовпцями чи рядками на сегменти.

Всім підзавданням передається повністю матриця MA та певний послідовний інтервал рядків матриці MB від i×h до (i+1)×h, де i — порядковий номер підзавдання, h = ⌈N/P⌉, а P — кількість підзавдань. Іншими словами, кожне підзавдання отримує прямокутну матрицю MBh розмірності приблизно ⌈N/P⌉ рядків на N стовпців.

Кожне підзавдання обчислює добуток прямокутної матриці MCh=MA×MBh, де MCh є i-тим сегментом загального результату MC.

Після завершення обчислень кожне підзавдання передає свій проміжний результат координатору обчислень (головному вузлу або основному потоку).

Координатор обчислень розміщує елементи кожного отриманого від підзавдань проміжного результату MCh на відповідну позицію у фінальному результаті MC.

Важливість цього алгоритму для експериментального дослідження пояснюється його здатністю розподіляти інформацію між вузлами не лише фіксованими, а й довільними розмірами блоків, ефективно завантажувати обчислювальні елементи, оскільки процес обчислень не потребує додаткових передач даних, що дозволяє максимально точно розрахувати коефіцієнти прискорення та ефективності. Також він легко піддається прогнозуванню щодо мінімальної кількості необхідних елементарних арифметичних операцій, яка для послідовного варіанту становить N^3, де N — розмірність матриць, а для паралельного варіанту для кожного окремого підзавдання — $N^2 \times \lceil N/P \rceil$, де P — кількість підзавдань.

Відносною одиницею навантаження на підзавдання для цього алгоритму виступає частина рядків матриці.

На рисунку 3.2 відображено залежність коефіцієнта прискорення даної програмної реалізації від значення розмірності матриць (N) для кожного програмного комплексу, в якому проводилися обчислювальні операції.

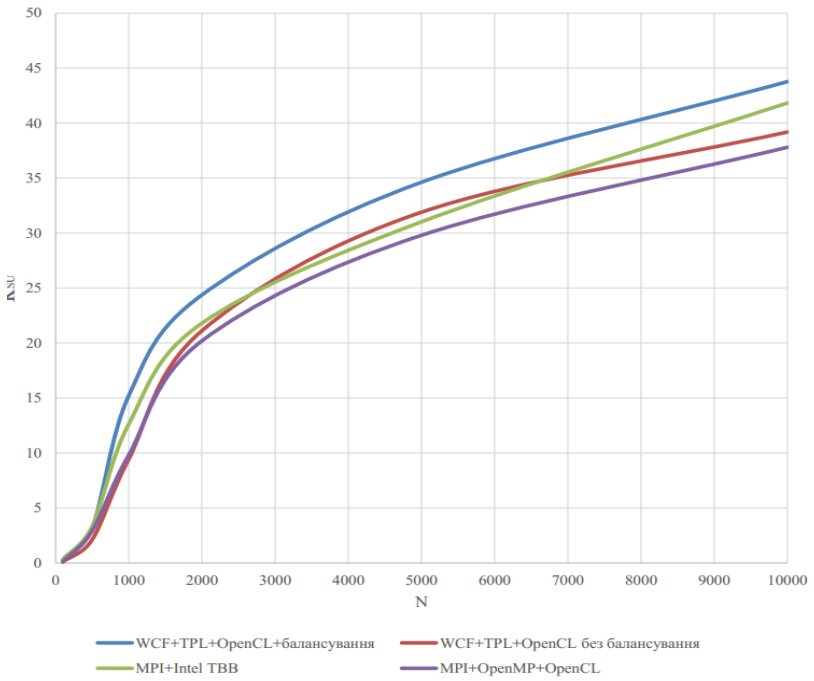

Рисунок 3.2 — Графік залежності коефіцієнту прискорення програми множення матриць від значення розмірності матриць

На рисунку 3.3 зображено графіки залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} даної програми від значення розмірності матриць (N) для кожного програмного комплексу, в якому велося розв’язання.

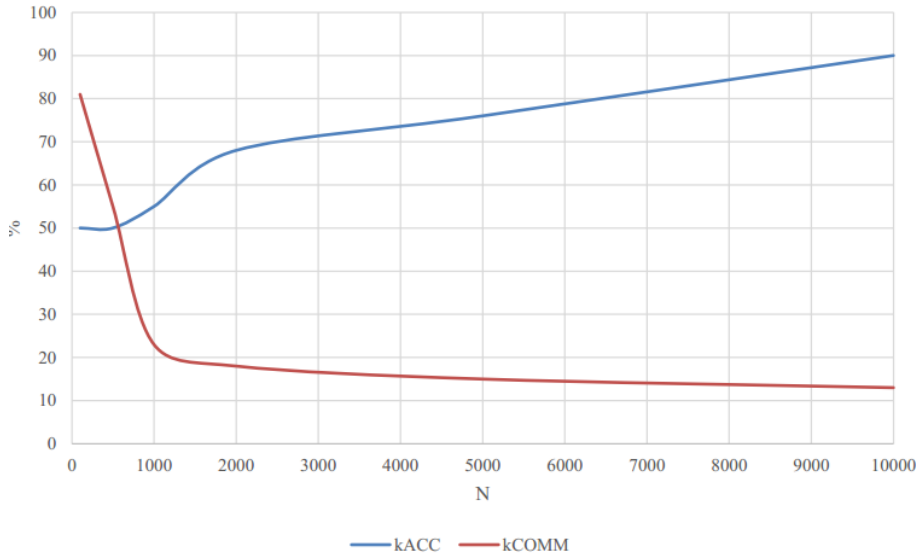

Рисунок 3.3 — Графік залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} програми множення матриць від значення розмірності матриць (N)

Обчислення значень деяких математичних та тригонометричних функцій у комп'ютерних архітектурах часто неможливе прямими методами. Проте розкладення функції в ряд та обчислення наближення цього ряду дозволяє комп'ютерним системам здійснювати розрахунки значень математичних та тригонометричних функцій із заданою точністю.

Для демонстрації розглянемо формулу обчислення гіперболічного арктангенса для значень в інтервалі від -1 до 1. Розрахунки виконуватимемо згідно з формулою (3.3):

$$arth(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots = \sum_{n=0}^{\infty} \frac{1}{2n+1} x^{2n+1} \quad |x| < 1 \tag{3.3}$$

Цей ряд легко обчислюється паралельно, оскільки всі його компоненти є незалежними. Однак, при впровадженні вимоги досягнення результатом певної бажаної точності, у паралельному алгоритмі обчислення цього ряду виникає потреба в постійній комунікації між обчислювальними елементами. Це зумовлено необхідністю після кожного обчислення проміжного результату і відправки його координатору отримувати інформацію про досягнення чи недосягнення необхідної точності на поточній ітерації.

Отже, координатор обчислень відповідає за отримання вхідних параметрів (x та показник точності у форматі дробового числа), передачу значення x іншим вузлам, розподіл позицій членів ряду для обчислення (або номерів поточних ітерацій для всіх вузлів), збір результатів та повідомлення вузлам про продовження або завершення обчислювальних процесів.

Таким чином, цінність цього завдання для експериментального дослідження полягає в тому, що з підвищенням вимог до точності збільшується і частота міжвузлових комунікацій.

Відносною одиницею навантаження на підзавдання для цього алгоритму є один елемент ряду. Однак, з огляду на невелику кількість вузлів, доцільніше визначити одиницею навантаження не одиночний елемент ряду, а певну групу елементів, тобто сегмент ряду.

На рисунку 3.4 показано залежність коефіцієнта прискорення даної програмної реалізації від значення точності (ε) для кожного програмного комплексу, що використовувався у дослідженні.

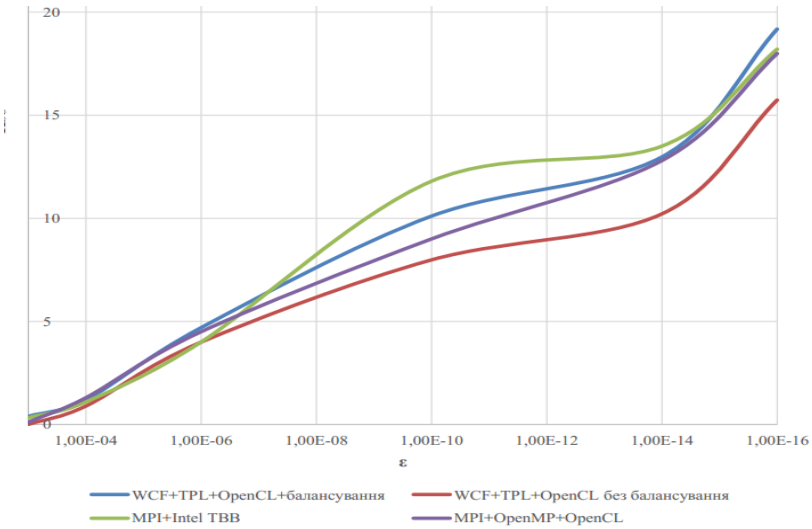

Рисунок 3.4 — Графік залежності коефіцієнту прискорення програми обрахування наближення ряду від значення точності (ε) для кожного програмного комплексу

На рисунку 3.5 зображено графіки залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} даної програми від значення точності (ε) для кожного програмного комплексу, в якому велося розв’язання.

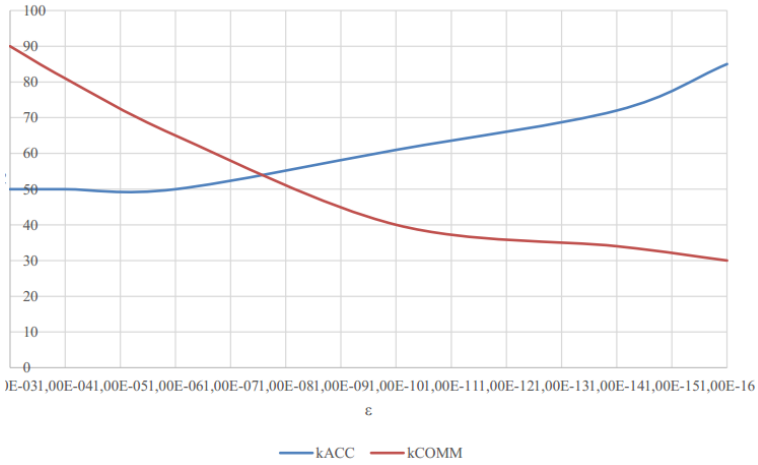

Рисунок 3.5 — Графік залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} програми обрахування наближення ряду від значення точності (ε)

Перефразований текст

Парадигма MapReduce представляє базовий підхід у сфері технологій

великих даних, орієнтований на стандартизацію розподіленої обробки великомасштабних інформаційних масивів, зокрема для реалізації операцій агрегації даних.

Алгоритм визначення частотності входження лексичних одиниць у текстових документах відповідно до даної парадигми передбачає наступну послідовність дій:

На стадії Map головний вузол здійснює декомпозицію документа на окремі фрагменти та забезпечує їх дистрибуцію між обчислювальними вузлами відповідно до розробленої методології.

Після отримання призначених фрагментів кожен вузол безпосередньо виконує Map-операцію, трансформуючи кожну лексичну одиницю у структуру "ключ-значення", де ключовим елементом є саме слово, а відповідним значенням - одиниця.

Обчислювальні вузли реалізують Reduce-операцію, що передбачає консолідацію сформованих структур "ключ-значення" шляхом підрахунку кількості входжень ідентичних ключів. При виявленні співпадіння ключових елементів відбувається елімінація однієї відповідної пари із структури, а значення іншої пари інкрементується на одиницю.

Обчислювальні вузли транслюють координуючому вузлу власні агреговані структури "ключ-значення", після чого координатор виконує заключну Reduce-операцію над усіма отриманими частковими результатами.

Дана задача є типовим практичним застосуванням для розподілених кластерних обчислювальних архітектур, що характеризується високою здатністю до паралелізації і в рамках запропонованого підходу дозволяє оцінити ефективність балансування навантаження.

Базовою одиницею обчислювального навантаження для цього алгоритму виступає текстовий сегмент - упорядкована послідовність лексичних елементів.

На рисунку 3.6 представлено залежність показника прискорення розробленої програмної реалізації від обсягу документа (загальної кількості слів N) для кожного програмного комплексу в межах проведеного експерименту.

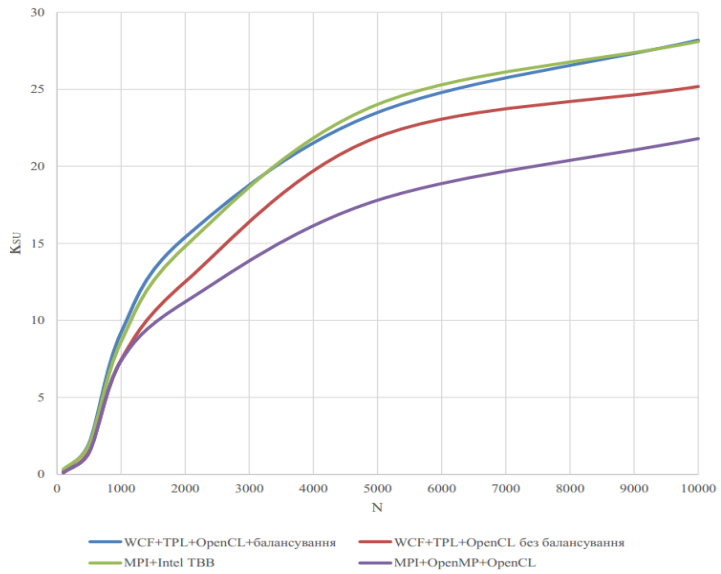

Рисунок 3.6 — Графік залежності коефіцієнту прискорення програми обрахування кількості зустрічей кожного слова у документі від значення розміру документу (кількості слів в ньому N)

На рисунку 3.7 зображено графіки залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} даної програми від значення розміру документу (кількості слів в ньому N) для кожного програмного комплексу, в якому велося розв’язання.

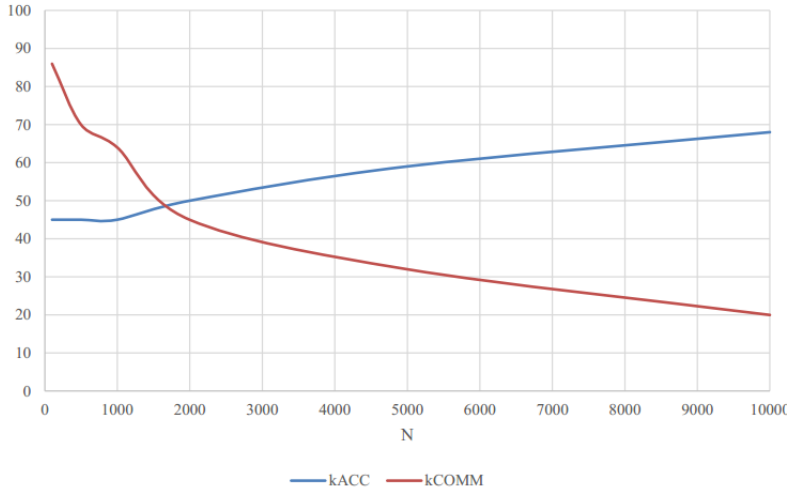

Рисунок 3.7 — Графік залежності коефіцієнту використання акселераторів k_{ACC} та коефіцієнту використання мережі прискорення k_{COMM} програми обрахування кількості зустрічей кожного слова у документі від значення розміру документу (кількості слів в ньому N)

Суттєвим фактором, що впливає на продуктивність розподілених архітектур, комунікаційним середовищем яких є глобальна мережа, і який може стати причиною фундаментальних помилок при оцінюванні навантаження та реалізації статичного балансування в таких системах, є варіативність затримки. Наступне експериментальне дослідження для обох досліджуваних технологій організації рівня розподіленої системи (WCF та MPI) аналізує середні параметри абсолютного відхилення тривалості передачі відповідних пакетів інформації від найбільшого із зафіксованих на всіх тестах середнього значення.

На рисунку 3.8 подано діаграму, отриману в результаті цього експериментального дослідження.

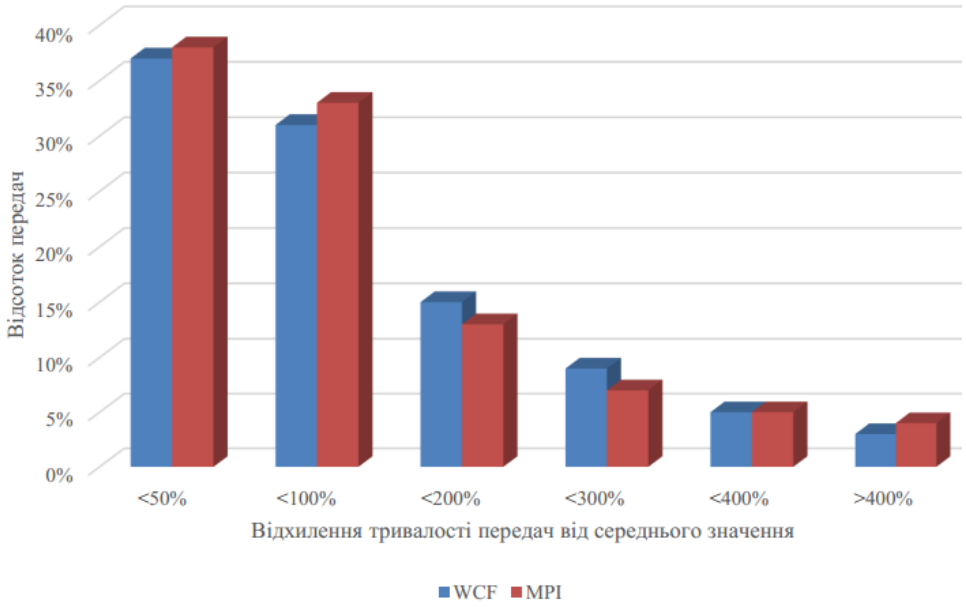

Рисунок 3.8 — Середні показники відхилення за модулем латентності від найбільш частого (найоптимальнішого) середнього значення

3.5 Висновки до розділу 3

У цьому розділі викладено процес створення програмного забезпечення для оптимізації паралельних обчислювальних процесів у розподілених неоднорідних

комп'ютерних архітектурах шляхом збалансованого розподілу навантаження між їхніми обчислювальними вузлами.

Було розроблено комплект експериментальних програмних рішень для функціонування розподілених неоднорідних комп'ютерних архітектур із акселераторами, які було впроваджено у двох реальних хмарних РГКС. У межах кожного програмного комплексу було реалізовано розв'язання низки поширених математичних та прикладних завдань:

— множення двох матриць;

— обчислення наближення функції за допомогою розкладу в ряд;

— підрахунок частоти появи всіх слів у документі за моделлю MapReduce.

На підставі результатів обчислення кожного з цих завдань із різними вхідними параметрами можна сформулювати такі висновки:

— застосування всіх розглянутих інструментів та технологій організації паралельних, розподілених та делегованих обчислювальних процесів демонструє високу ефективність і призводить до значного зростання коефіцієнтів прискорення програмних реалізацій;

— запропонований підхід до організації паралельних обчислювальних процесів в РГКС із акселераторами на базі використання технологій WCF, TPL та OpenCL демонструє вищу ефективність порівняно з використанням технологій MPI у поєднанні з OpenMP+OpenCL або Intel TBB, але тільки для тих завдань, де відсутній інтенсивний обмін інформацією між вузлами під час обчислювальних процесів;

— запропонована технологія оптимізації паралельних обчислювальних процесів в РГКС із акселераторами на основі балансування навантаження при її впровадженні в програмних комплексах, що функціонували в системі 1 (зі значною неоднорідністю), забезпечила додаткове підвищення коефіцієнтів прискорення в середньому на 10-17%.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи було розроблено систему оптимізації паралельних обчислень у розподілених гетерогенних комп’ютерних середовищах.

Було проведено аналіз існуючих рішень організації паралельних обчислень в гетерогенних комп’ютерних середовищах. Проаналізовано процес обробки задач в розподіленій комп’ютерній системі. Було розглянуто сучасні підходи до паралельного програмування та здійснено порівняння моделей паралельного програмування. Виявлено проблему відсутності системи балансування навантаження в розподілених комп’ютерних системах.

Описано організацію структури та режиму роботи з довільними обчисленнями розподіленої гетерогенної комп’ютерної системи. Для ефективної роботи планувальників запропоновано комплексне оцінювання як вхідної задачі, так і всіх елементів системи, при цьому оцінювання елементів враховує особливості задачі, розв’язання якої планується в системі.

Описано процес розробки програмного забезпечення для оптимізації паралельних обчислень в розподілених гетерогенних комп’ютерних системах шляхом балансування навантаження на їх вузли.

Розроблено пакет тестових програмних комплексів для організації роботи розподілених гетерогенних комп’ютерних систем із акселераторами, які було розгорнуто в двох реальних хмарних РГКС. В рамках кожного із програмних комплексів було організовано розв’язання ряду поширених математичних та прикладних задач

За результатами обрахунку кожної із цих задач із різними вхідними параметрами виявлено, що запропонована технологія оптимізації паралельних обчислень в РГКС із акселераторами на основі балансування навантаження при його застосуванні в сильно неоднорідних програмних комплексах забезпечив додаткове зростання коефіцієнтів прискорення в середньому на 10-17%.

Отже всі поставлені задачі виконано, мету роботи досягнуто.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sutter, H.: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software / H. Sutter // Dr. Dobby's Journal. – 2005. –Vol. 30, no. 3.

2. Martonosi, Margaret. Parallelism, heterogeneity, communication: Emerging challenges for performance analysis / Margaret Martonosi // Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on IEEE. – Washington, DC, USA: IEEE Computer Society, 2012. 124 с.

3. Software as a service for data scientists / Bryce Allen, John Bresnahan, Lisa Chilers [et al.] // Commun. ACM. – 2012. – Vol. 55, no. 2. 81-88 с.

4. Тизунь Віталій Юрійович. Система розпізнавання тексту на основі гетерогенних комп'ютерних систем : дис. маг. техн. наук / Тизунь Віталій Юрійович. –Київ, 2019. – 89 с. – Бібліогр. : 85–88 с.

5. Tanenbaum, Andrew S. Distributed Systems: Principles and Paradigms (2nd Edition) / Andrew S. Tanenbaum, Maarten van Steen. – Upper Saddle River, NJ, USA: Prentice-Hall, Inc. – 2006.

6. Дослідження ефективності дрібнозернистого паралелізму в багатоядерних комп'ютерних системах. / В.В. Демчик, О.В. Корочкін, О.В. Русанова // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. – 2020. – № 66. 56-61 с.

7. Cheng, L. Intelligent scheduling for simultaneous CPU-GPU applications : M.Sc. in Computer Sciences thesis [Електронний ресурс] / L. Cheng: Graduate College, University of Illinois at Urbana-Champaign, Illinois, USA. – 2017. – Режим доступу до ресурсу: http://rsim.cs.uiuc.edu/Pubs/Lin_thesis.pdf (дата звернення 01.05.2025).

8. "Безпека, відмовостійкість, інтелект" (ICSFTI2018), Київ, Україна. – 2018, С. 362 – 368. – Режим доступу до ресурсу: http://comsys.kpi.ua/upload/GN_330-336_DemchykValerii.pdf (дата звернення 01.05.2025).

9. Стіренко С. Г. Організації паралельний обчислювальних процесів в кластерних системах [Текст] / С.Г. Стіренко, – К.: «Три К», 2014. – 196 с.

10. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник [Текст] / Жуков І.А., Корочкін О.В. // – К.: Корнійчук, 2005. – 226 с. – ISBN 996-7599-36-1.

11. Коцовський В. М. Теорія паралельних обчислень: навчальний посібник. Ужгород: ПП «АУТДОР-Шарк», 2021. 188 с. URL: https://dspace.uzhnu.edu.ua/jspui/bitstream/lib/38994/1/%D0%9D%D0%B0%D0%%BF%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA.pdf

12. Сабат Н.В. Паралельні та розподілені обчислення : конспект лекцій/ Н.В.Сабат, В.Б. Кропивницька. – Івано-Франківськ : ІФНТУНГ, 2017. – 80 с. 3.

13. Семеренко, В. П. Технології паралельних обчислень : навч. посіб. Вінниця: ВНТУ, 2018. – 104 с. URL: https://www.researchgate.net/publication/334710599_V_P_Semerenko_TEHNOLOGII_PARALELNIH_OBCISLEN_Ministerstvo_osviti_i_nauki_Ukraini_Vinnickij_nacionalnij_tehnicnij_universitet.

14. Яровий А. А. Методи та засоби організації високопродуктивних паралельно ієрархічних обчислювальних систем із рекурсивною архітектурою: монографія. Вінниця: ВНТУ, 2016. 363 с.

15. Novillo, D. OpenMP and automatic parallelization in GCC / D. Novillo // Proceedings of the GCC developers' summit. – San Francisco, CA, USA ACM, 2006. 132-141 с.

16. Служба підтримки Microsoft. URL: https://support.microsoft.com/uk-ua

17. Технічна документація, API, зразки коду. URL: https://docs.microsoft.com/uk-ua/

18. Open Grid Forum. URL: http://www.gridforum.org

19. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») [Електронний ресурс] / уклад. О.Д. Азаров, С.В. Богомолов, С. І. Швець. – Вінниця : ВНТУ, 2023. – 105 с.

ДОДАТОК А

Технічне завдання

Міністерство освіти та науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ ВНТУ

д.т.н., проф.

_____ О. Д. Азаров

"21" 03 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Система оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах»

08-54.БКР.003.00.000 ТЗ

Керівник: к.т.н., доцент каф. ОТ

_____ Колесник І.С.

Студентка групи КІ-23мсз

_____ Кривоносова З.В.

Вінниця 2025

1 Найменування та область застосування

Робоча назва проекту «Система оптимізації паралельних обчислень у розподілених гетерогенних комп’ютерних середовищах», розробляється для навчання студентів створювати системи паралельних обчислень у розподілених комп’ютерних середовищах.

2 Основи для розробки

Основою для розробки є дисципліни Web-програмування, Об’єктно-орієнтоване програмування, технології розподілених систем та паралельних обчислень.

3 Мета та призначення розробки

Експлуатаційне призначення розробки — розробка системи оптимізації паралельних обчислень у розподілених гетерогенних комп’ютерних середовищах.

4 Етапи БДР та очікувані результати

Робота виконується в п’ять етапів, що наведені в таблиці 4.1.

Таблиця 4.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз завдання. Вступ	21.03.2025	25.03.2025	Вступ
2	Аналіз предметної області організації паралельних обчислень	26.03.2025	01.04.2025	розділ 1
3	Розробка системи балансування навантаження в розподілених КС	02.04.2025	25.04.2025	Розділ 2
4	Реалізація оптимізованих паралельних обчислень в розподілених КС	26.04.2025	02.05.2025	Розділ 3
5	Оформлення пояснювальної записки	03.05.2025	13.05.2025	ПЗ, презентація

5 Матеріали, що подаються до захисту БКР

Пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відзив наукового керівника, рецензія опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

6 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженою наказом ректора.

7 Вимоги до оформлення БКР

Вимоги до БКР наведені нижче:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи».

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Система оптимізації паралельних обчислень у розподілених гетерогенних комп'ютерних середовищах

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 25,69%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Азаров О.Д., зав. каф. ОТ _____
(прізвище, ініціали, посада) (підпис)

Крупельницький Л.В., доц. каф ОТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колесник І.С., доц. каф. ОТ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Кривоносова З.В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

СИСТЕМА ОПТИМІЗАЦІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ У РОЗПОДІЛЕНИХ ГЕТЕРОГЕННИХ КОМП’ЮТЕРНИХ СЕРЕДОВИЩАХ

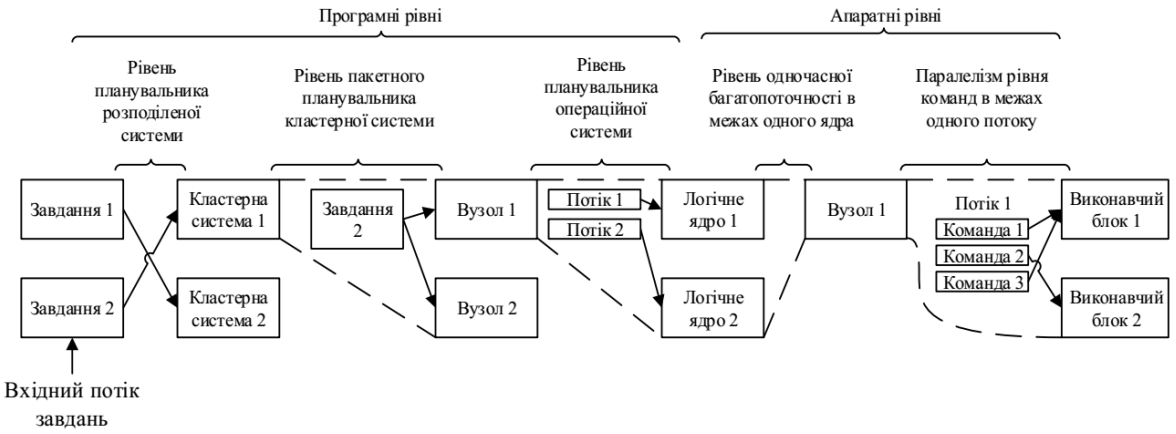

Рисунок Б.1 — Схема процесу обробки завдань в розподіленій комп’ютерній системі

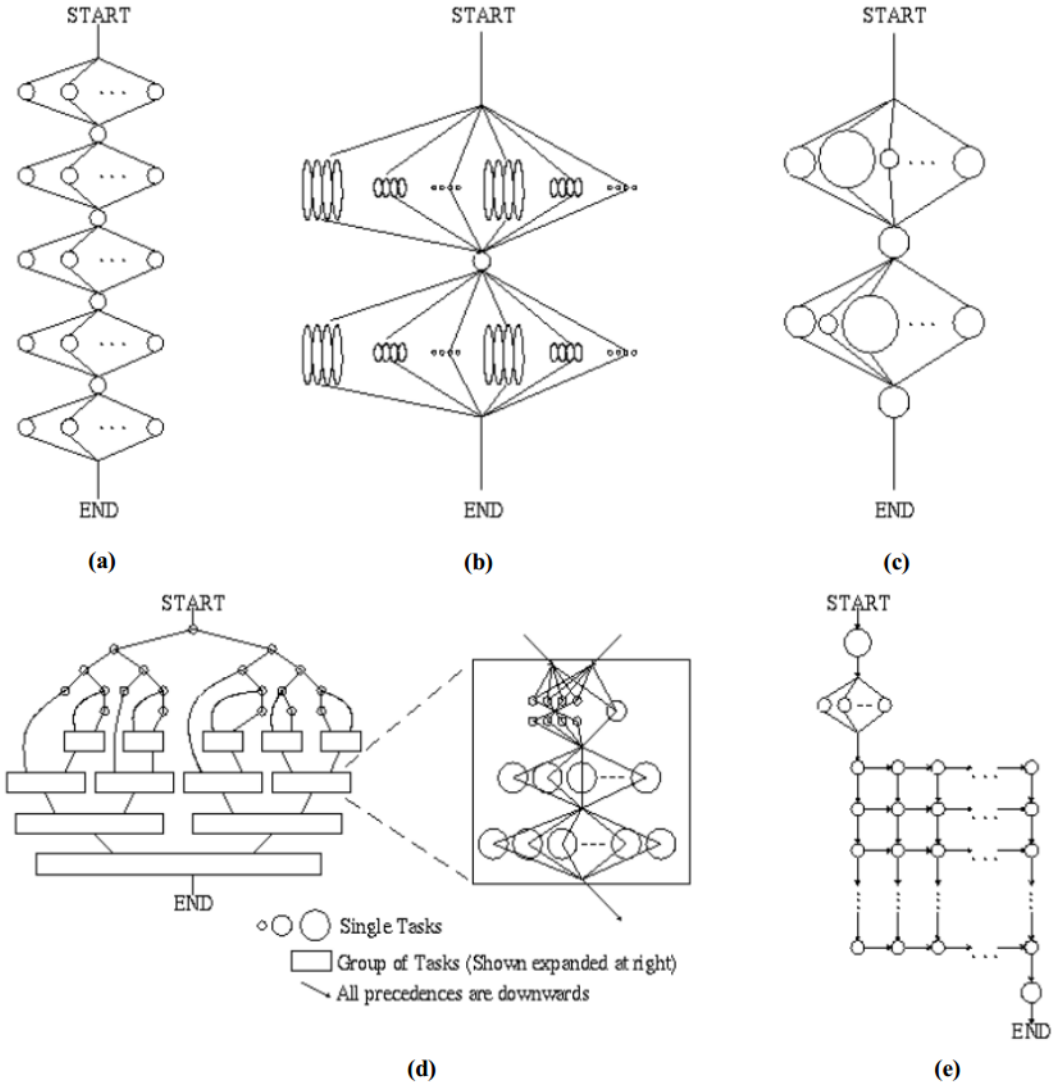

Рисунок Б.2 — Приклади найрозповсюдженіших схем роботи (графів паралельних задач) паралельних програм

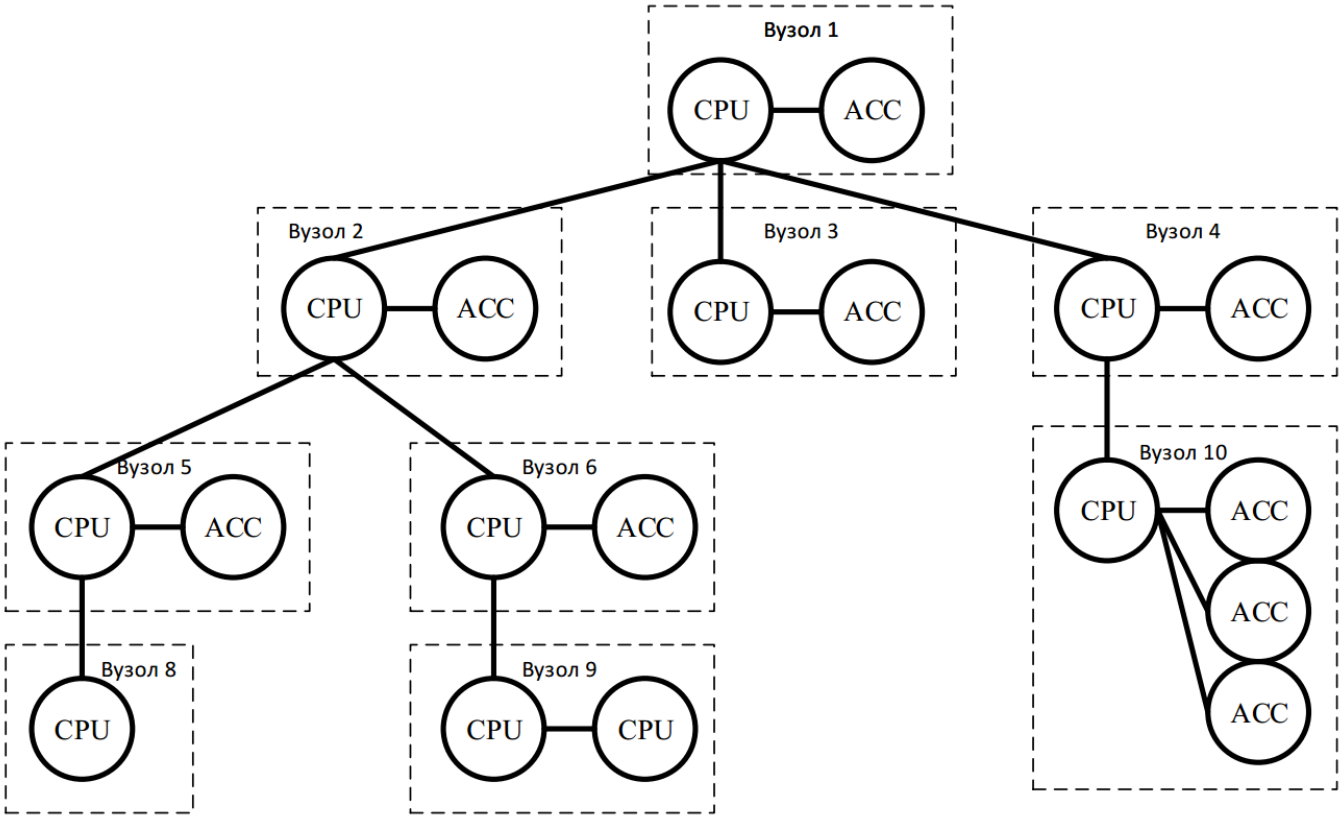

Рисунок Б.3 — Схематичний приклад графу для системи з 10 різноманітних за складом вузлів

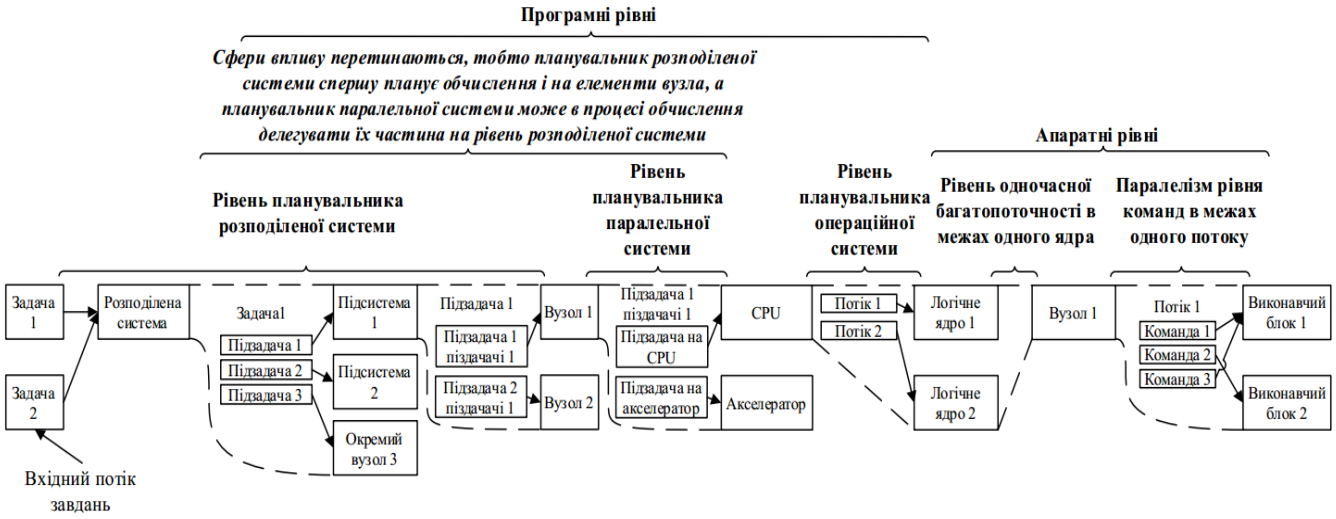

Рисунок Б.4 — Модифікована схема процесу обробки задач в розподілених гетерогенних комп’ютерних системах

Рисунок Б.5 — Карта розміщення вузлів тестових систем з відображенням умовних логістичних зв’язків між їх вузлами

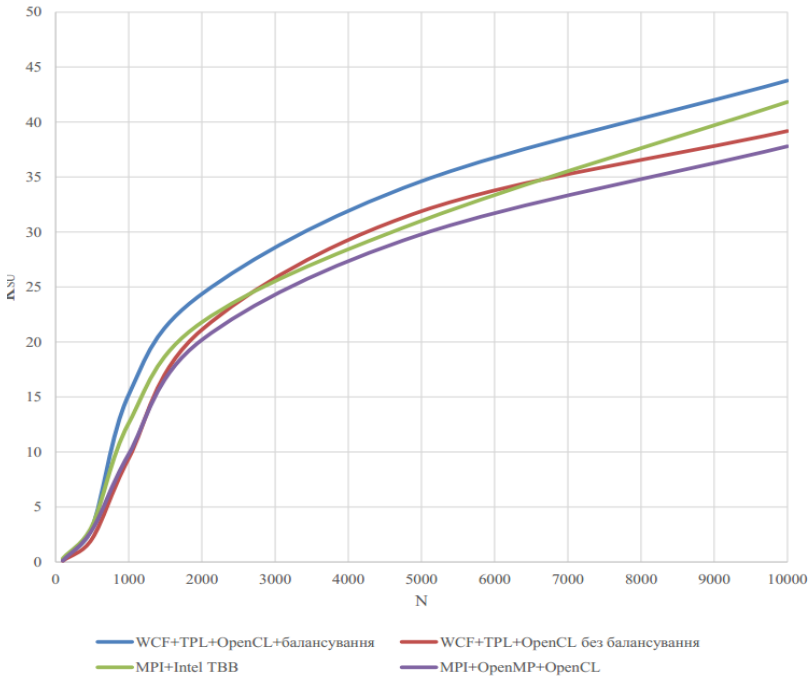

Рисунок Б.6 — Графік залежності коефіцієнту прискорення програми множення матриць від значення розмірності матриць

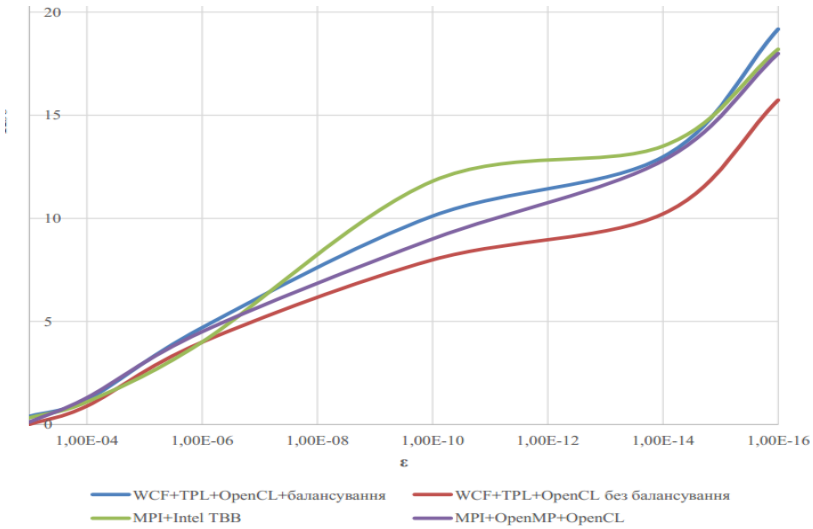

Рисунок Б.7 — Графік залежності коефіцієнту прискорення програми обрахування наближення ряду від значення точності (ε) для кожного програмного комплексу

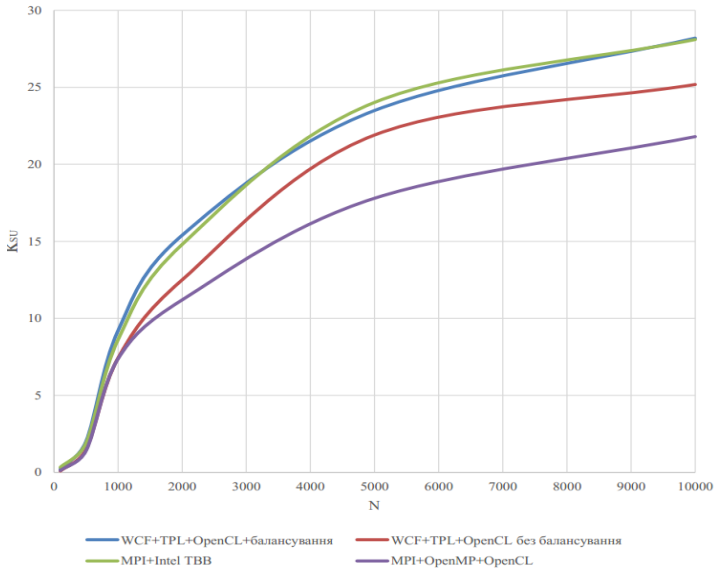

Рисунок Б.8 — Графік залежності коефіцієнту прискорення програми обрахування кількості зустрічей кожного слова у документі від значення розміру документу (кількості слів в ньому N)

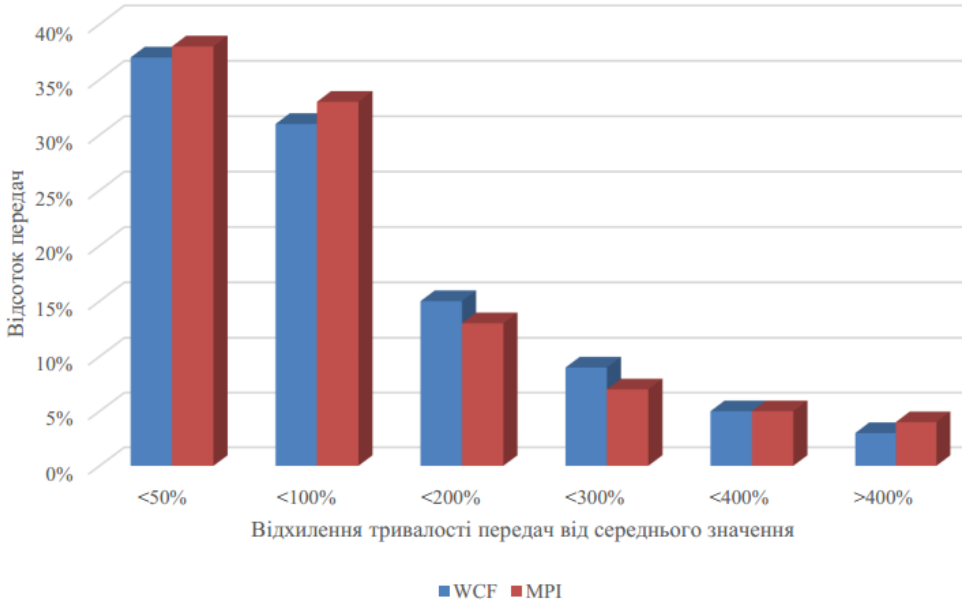

Рисунок Б.9 — Середні показники відхилення за модулем латентності від найбільш частого (найоптимальнішого) середнього значення