Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Підвищення ефективності комп'ютерної системи засобами паралельної обробки інформації
(тема)

Виконав:
студент II курсу, групи КСМм-22-1
Гулак А.С.
(прізвище, ініціали)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Піскарьов О.М.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ ______________ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри ________________
(підпис)
“____”________________ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Гулаку Антону Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Підвищення ефективності комп'ютерної системи засобами паралельної обробки інформації

затверджена наказом по університету від “ 06 ” листопада 2023 р. № 1298 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи
   1) аналіз методів паралельної обробки даних
   2) розробка підвищення ефективності комп’ютерної системи;
   3) створення демонстраційної системи.

4. Перелік питань, що потрібно опрацювати у роботі
   1) Огляд стану паралельної обробки даних
   2) Аналіз принципів та методів паралельної обробки інформації
   3) Розробка методу підвищення ефективності комп’ютерної системи
   4) Моделювання роботи системи
   5) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (слайдів) 14 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд стану паралельної обробки даних	07.11.23-18.11.23	
2	Аналіз принципів та методів паралельної	20.11.23-30.11.23	
	обробки інформації		
3	Розробка методу підвищення	1.12.23-09.12.23	
	ефективності комп’ютерної системи		
4	Моделювання роботи системи	11.12.23-23.12.23	
5	Оформлення матеріалів кваліфікаційної роботи	24.12.23-05.01.24	
6	Подання роботи керівникові та її	06.01.24-10.01.24	
	попередній захист		
7	Подання роботи на рецензування	11.01.24-13.01.24	

Дата видачі завдання 06 листопада 2023 р.

Студент ____________________
(підпис)

Керівник роботи ____________________ доц. Піскарьов О.М.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 99 с., 20 рис., 1 табл., 2 дод., 17 джерел.

ПАРАЛЕЛЬНА КОМП’ЮТЕРНА СИСТЕМА, РОЗПОДІЛЕНА КОМП’ЮТЕРНА СИСТЕМА, ШТУЧНИЙ ІНТЕЛЕКТ, ІНТЕРФЕЙС ПЕРЕДАЧІ ПОВІДОМЛЕНЬ, БАЛАНСУВАННЯ НАВАНТАЖЕННЯ.

Метою кваліфікаційної роботи є вдосконалення засобів паралельної обробки інформації для підвищення ефективності комп'ютерної системи.

У ході виконання кваліфікаційної роботи було проведено аналіз поточного стану паралельної обробки даних, розглянута еволюція обчислювальних платформ, класифікація архітектур комп'ютерних систем, а також принципи та методи паралельної обробки інформації. Здійснено огляд технологій програмування паралельних КС. Запропоновано метод підвищення ефективності розподіленої паралельної комп’ютерної системи. Проведено моделювання роботи розподіленої паралельної комп’ютерної системи для обчислення наближення ряду та підрахунок кількості зустрічей слова за допомогою моделі MapReduce.

Визначено, що запропонований метод багаторівневого балансування навантаження є ефективним для підвищення продуктивності програмних комплексів, в умовах неоднорідних розподілених КС із акселераторами, де розподілені вузли різних класів, використання багаторівневого балансування навантаження дало значний приріст коефіцієнтів прискорення.

ABSTRACT

Master's thesis: 99 pages, 20 figures, 1 tables, 2 appendices, 17 sources.

PARALLEL COMPUTER SYSTEM, DISTRIBUTED COMPUTER SYSTEM, ARTIFICIAL INTELLIGENCE, MESSAGE PASSING INTERFACE, LOAD BALANCING.

The purpose of the qualification work is to improve the means of parallel information processing to increase the efficiency of a computer system.

In the course of the qualification work, the current state of parallel data processing was analyzed, the evolution of computing platforms, the classification of computer system architectures, as well as the principles and methods of parallel information processing were considered. The technologies of programming parallel CS are reviewed. A method for increasing the efficiency of a distributed parallel computer system is proposed. A simulation of a distributed parallel computer system is carried out to calculate the approximation of a series and count the number of word occurrences using the MapReduce model.

It is determined that the proposed method of multi-level load balancing is effective for improving the performance of software systems, in conditions of heterogeneous distributed CS with accelerators, where nodes of different classes are distributed, the use of multi-level load balancing has given a significant increase in acceleration factors.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 ОГЛЯД ПОТОЧНОГО СТАНУ ПАРАЛЕЛЬНОЇ ОБРОБКИ ДАНИХ 10
1.1 Історичний огляд та розвиток паралельних обчислень 10
1.2 Еволюція обчислювальних платформ для паралельної обробки 12
1.3 Класифікація архітектур комп'ютерних систем 18
2 ПРИНЦИПИ ТА МЕТОДИ ПАРАЛЕЛЬНОЇ ОБРОБКИ ІНФОРМАЦІЇ 21
2.1 Рівні паралельності в КС 21
2.2 Сучасні моделі паралельної обробки інформації 24
2.2.1 Модель MapReduce 24
2.2.2 Модель SIMD 28
2.2.3 Dataflow model 30
2.3 Огляд технологій програмування паралельних КС 31
2.3.1 Технологія низькорівневої передачі повідомлень 33
2.3.2 Технологія розпаралелювання з застосуванням препроцесору 34
2.3.3. Технологія відвантажених обчислень 42
2.3.4 Технологія відкладених обчислень 43
3 РОЗРОБКА МЕТОДУ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНОЇ КОМП’ЮТЕРНОЇ СИСТЕМИ 46
3.1 Порядок оцінки вхідного завдання 49
3.2 Процедура ініціалізації КС 51
3.3 Оцінка ребер графу системи 53
3.4 Алгоритм підвищення ефективності КС 56
3.5 Процес делегування обрахунків між вузлами КС 63

4 МОДЕЛЮВАННЯ РОБОТИ РОЗПОДІЛЕНОЇ ПАРАЛЕЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ 64
4.1 Вибір прототипів компонентів системи 64
4.2 Вибір засобів програмної реалізації 65
4.3 Вибір програмних комплексів та КС 67
4.4 Моделюванная роботи КС 70
4.4.1 Моделювання обчислення наближення ряда 71
4.4.2 Моделювання підрахунку кількості зустрічей слова за допомогою моделі MapReduce 74
ВИСНОВКИ 78
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 80
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 82
ДОДАТОК Б Наукові публікації за темою кваліфікаційної роботи 90

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КС– комп’ютерна система

ПКС – паралельна комп’ютерна система

РКС – розподілена комп’ютерна система

ШІ – штучний інтелект

CPU – центральний процесор (англ., Central Processing Unit)

GPU – графічний процесор (англ., Graphics Processing Unit),

GPGPU – обчислення загального призначення на графічних процесорах (англ., General-purpose computing on graphics processing units)

MPI – інтерфейс передачі повідомлень (англ., Message Passing Interface)

OpenMP – відкритий стандарт для розпаралелювання програм (англ., Open Multi-Processing)

SISD – Single Instruction Single Data

MISD – Multiple Instruction Single Data

SIMD – Single Instruction Multiple Data

MIMD – Multiple Instruction Multiple Data

ВСТУП

У сучасному світі паралельні КС відіграють важливу роль у різних сферах людської діяльності, таких як наука, промисловість, медицина, фінанси та інші. Зі зростанням обсягів даних та складності обчислювальних задач виникла необхідність вдосконалення паралельних алгоритмів обчислень, які здатні ефективно розподіляти завдання між багатьма процесорами або обчислювальними вузлами. Ці алгоритми є ключовими для розвитку високопродуктивних КС та додатків, таких як великі бази даних, системи штучного інтелекту та обробки зображень. Найбільш об'ємні обчислення здійснюються переважно на розподілених системах. В останні роки значну популярність також набувають гетерогенні системи, які включають різноманітні обчислювальні елементи або так звані акселератори обчислень (графічні процесори). З появою технологій загального програмування на графічних процесорах (GPGPU), таких як шейдери загального призначення, вони стали придатні для виконання різноманітних обчислень. Такий підхід породжує нові виклики - планування обчислень та балансування навантаження, для вирішення яких існує набір стандартних методів розподіленої системи. Але більшість з цих методів спрямовані на конкретні види завдань. Адаптації були розроблені лише на основі декількох таких методів, але вони не завжди підтримують більшість технік паралельного програмування, особливо нерегулярний паралелізм. Крім того, характерною рисою їх функціонування є необхідність здійснення компромісів, будь то витрати часу чи оптимальність отриманого розподілу.

Отже, метою даної кваліфікаційної роботи є аналіз ефективності різних алгоритмів паралельних обчислень та їх вдосконалення. Для досягнення цієї мети будуть проведені експериментальні дослідження різних алгоритмів та оцінені показники їх ефективності в різних умовах використання.

1 ОГЛЯД ПОТОЧНОГО СТАНУ ПАРАЛЕЛЬНОЇ ОБРОБКИ ДАНИХ

1.1 Історичний огляд та розвиток паралельних обчислень

У сучасному світі існує безліч проблем, які потребують ефективного використання комп'ютерних ресурсів. Багато додатків, традиційно пов'язані з науковою галуззю, вимагають високої продуктивності. Проте, згідно з прогнозами, програми штучного інтелекту (ШІ) та машинного навчання в майбутньому стануть ключовими користувачами великомасштабних обчислень [1]. Деякі приклади таких програм включають:

- моделювання великих пожеж;
- моделювання цунамі та штормових хвиль від ураганів;
- розпізнавання голосу для комп'ютерних інтерфейсів;
- моделювання поширення вірусів та розробка вакцин;
- моделювання кліматичних умов протягом десятиліть і століть;
- розпізнавання зображень для технології безпілотного автомобіля;
- зменшення енергоспоживання для мобільних пристроїв.

Використання методів підвищення швидкодії, дозволяє вирішувати великі проблеми та обробляти об'ємні набори даних, одночасно запускаючи симуляції в декілька разів швидше. Зазвичай програми не використовують повністю обчислювальні можливості сучасних комп'ютерних систем. Паралельні обчислення стають ключем до розкриття їх потенціалу [2].

Паралельні обчислення - це підхід до виконання обчислень, при якому обчислювальна задача розбивається на менші підзадачі, які виконуються паралельно (одночасно) на різних процесорах чи обчислювальних вузлах. Застосування паралельних обчислень дозволяє використовувати потенціал багатьох обчислювальних ресурсів, щоб прискорити виконання завдань.

Для покращення продуктивності програм, спершу необхідно ідентифікувати частини коду, які можна виконувати паралельно. Далі

розробляють паралельні алгоритми, які розділяють обчислювальну задачу на незалежні частини. У програмі використовують механізми паралельного виконання, такі як потоки або процеси, для одночасного вирішення цих частин задачі.

Паралельні обчислення – це виконання декількох операцій одночасно. Повне використання паралельних обчислень вимагає свідомої розробки та оптимізації програм, що потребує активного зусилля з боку програміста. Спочатку необхідно ідентифікувати та виявити потенціал паралелізму в програмі. Можливість виконання команд або операцій у будь-якому порядку вимагає свідомого поєднання, припускаючи, що заходи з синхронізації та безпеки вжиті. Для досягнення ефективного паралельного виконання важливо правильно використовувати наявні ресурси. При цьому слід враховувати необхідність впровадження заходів синхронізації та безпеки для уникнення конфліктів та забезпечення коректності виконання операцій.

Паралельні обчислення породжують нові проблеми та виклики, яких не існує в послідовних обчислювальних середовищах. Адаптація до додаткових труднощів паралельного виконання вимагає зміни процесів та певного рівня досвіду та практики. Вирішення цих викликів передбачає вміння пристосовувати код, враховуючи особливості паралельного середовища та забезпечуючи ефективність обчислень при одночасному врахуванні аспектів синхронізації та безпеки.

Паралельні обчислення - це системний підхід, що включає в себе виявлення можливостей паралелізму в алгоритмах, їхню імплементацію у програмному забезпеченні та розуміння компромісів і важливих міркувань, пов'язаних із обраною паралельною реалізацією.

Однією з основних мотивацій для використання паралельних обчислень є підвищення продуктивності, що охоплює швидкість виконання, масштабність проблеми та енергоефективність. Важливі рішення, такі як включення паралелізму, часто приймаються на етапі проектування. Обґрунтований дизайн є ключовим етапом для досягнення успіху, і

уникнення цього етапу може призвести до проблем у подальшому. Також важливо зберігати реалістичні очікування та розуміти як наявні ресурси, так й характер проекту.

Історичний огляд та розвиток паралельних обчислень охоплюють значний період часу, починаючи з ранніх спроб використання паралельних обчислень до досягнень в сучасній галузі. Початкові спроби використання паралельних обчислень відзначалися експериментальними та обмеженими можливостями апаратного забезпечення. Важливим етапом було створення архітектур з подвійними процесорами та введення концепцій, таких як векторизація та піпелайнінг.

1.2 Еволюція КС для паралельної обробки інформації

Ера векторизації, яка припала на 1970-1980-і роки, характеризувалася розвитком векторних обчислювальних систем, зокрема в суперкомп'ютерах Cray. Це дозволило підвищити продуктивність завдяки обробці декількох даних одночасно. Розширення векторних інструкцій та регістрів стали важливими досягненнями цього періоду. У середині 1980-х було запущено новий тип паралельних обчислень - проект Caltech Concurrent Computation побудував суперкомп'ютер для наукових застосувань на базі 64 процесорів Intel 8086/8087 (рисунок 1.1). Ця система показала, що велика продуктивність може бути досягнута за допомогою доступних масово мікропроцесорів.

З появою 1990-их років наступила експансія масштабування паралельних систем. Зростання обсягів даних та складності задач призвело до розробки масштабованих систем паралельних обчислень. Розширення паралельних алгоритмів та моделей програмування стали активно вивчатися для розв'язання великих завдань. Масивно-паралельні процесори (MPP) стали головними у обчислювальних системах, і суперкомп'ютер ASCI Red

(рисунок 1.2) у 1997 році перевищив планку у один трильйон операцій з плаваючою точкою в секунду.

Рисунок 1.1 – Суперкомп’ютер Caltech Concurrent Computation

2000-і роки призначені для появи багаторівневих систем. З'явлення багаторівневих обчислювальних систем, які поєднують в собі векторні та паралельні архітектури, відкрило нові можливості. Розвиток кластерних та грід-систем дозволив об'єднувати потужності різних обчислювальних вузлів.

Рисунок 1.2 – Суперкомп’ютер ASCI Red

З початку 2010-их років наступає епоха масового паралелізму. Запровадження багатоядерних процесорів та графічних обчислювальних одиниць (GPU) відкрило нові горизонти. Сучасні тенденції включають розвиток технологій паралельного програмування, таких як MPI, OpenMP, та інших. Паралельні обчислення активно використовуються у великому спектрі галузей, від наукових досліджень та інженерії до штучного інтелекту та аналізу великих даних.

Історичний розвиток паралельних обчислень свідчить про постійний ріст та вдосконалення технологій та методів, що відбуваються у цій галузі.

В останні роки спостерігається інтенсивний розвиток паралельних обчислень, зумовлений кількома суттєвими тенденціями та інноваціями. Однією з ключових аспектів цього розвитку є широке поширення многоядерних процесорів, які надають змогу виконувати кілька завдань одночасно.

Зростання обчислювальної потужності паралельних систем стає можливим завдяки появі нових архітектур, спрямованих на оптимізацію роботи з багатьма ядрами. Крім того, розвиток графічних обчислювальних одиниць (GPU) відкриває нові перспективи для паралельних обчислень. Великі технологічні компанії, такі як NVIDIA та AMD, активно працюють над вдосконаленням GPU-архітектур і розширенням їх можливостей для обробки великого обсягу даних одночасно.

Паралельність стає ключовим аспектом великих обчислювальних задач, таких як наукові дослідження, штучний інтелект, обробка великих даних і симуляції. З'являються нові мови програмування та інструменти, які спрощують розробку паралельних програм і підвищують їхню продуктивність.

Спрямованість на розширення можливостей паралельних обчислень в сучасному світі свідчить про їхню важливість для різноманітних галузей, від наукових досліджень і аналізу даних до високопродуктивних обчислювальних застосувань у промисловості та технологічному секторі.

Сучасні архітектури паралельної обробки інформації відрізняються своєю різноманітністю та глибокою спеціалізацією на різних завданнях. Інновації включають багаторівневі архітектури, що поєднують векторні, масштабовані та інші підходи для оптимізації продуктивності та ефективної роботи з обчислювальними завданнями.

Протягом історії сучасної комп’ютерної техніки, основними критеріями ефективності процесорів та обчислювальних систем були час, необхідний для вирішення завдань, та обсяг доступної пам'яті. Проте в останні роки спостерігається значний ріст об'ємів пам'яті в комп'ютерних системах, зробивши час вирішення завдань єдиним ключовим показником ефективності [3]. Сучасні алгоритми та методи оптимізації обчислень акцентуються саме на мінімізації цього показника, як на апаратному, так і на програмному рівнях.

Спочатку з апаратної точки зору основним методом прискорення роботи ядра було збільшення його тактової частоти. Однак цей підхід має свої обмеження, і на сучасному етапі частотні характеристики чипів досягли практично максимальних значень для кремнієвих чипів. Наступним кроком стало виділення в межах одного процесору кількох незалежних ядер, спроможних виконувати обчислення паралельно. Системи, які використовують такі процесори, стали відомі як паралельні КС.

Втім, крім значного зростання вимог до програмного забезпечення, призначеного для таких систем, з'явилася необхідність серйозного ускладнення інших апаратних компонентів для забезпечення ефективної роботи кожного додаткового ядра. Таким чином, хоча зростання кількості ядер відповідало потребам звичайних користувачів, воно не встигало за вимогами науки та бізнесу. У той самий час розвиток мережевих технологій продовжувався, і вони вже мали здатність забезпечувати швидку передачу значних обсягів даних. Таким чином, відповіддю на проблему прискорення обчислень в комп'ютерних системах стала ідея розподілу обчислень між кількома ідентичними паралельними комп'ютерними системами через

локальну комунікаційну мережу. Цей новий клас обчислювальних систем отримав назву кластерні комп'ютерні системи.

У той самий період розвиток комп'ютерних мереж забезпечив прийнятну швидкість передачі даних через глобальну мережу. Це, спільно з появою на ринку різноманітних і доступних за ціною процесорних елементів, створило умови для визначення більш широкого класу систем - розподілених комп'ютерних систем. У таких системах елементи можуть додаватися в необмеженій кількості, мати різні характеристики, включаючи структуру кластерних систем, і розташовуватися як в одному приміщенні, так і в різних кінцях світу.

Іншим способом оптимізації часу виконання програм в комп'ютерних системах стало розподілення обчислень в самій системі. Оскільки процесор виступає центральним елементом системи, навіть під час виконання великої обчислювальної задачі він здійснює значну кількість інших, не пов'язаних з поточною задачею обчислень. Супервізор операційної системи розміщує всі ці завдання, особливо ті, які мають вищий пріоритет, на вершині стеку завдань. Отже, користувацькі обчислення, в будь-якому випадку, будуть періодично зупинятися на короткі періоди, а ресурси системи використовуватимуться для обробки більш важливих завдань. Таким чином, розвантаження процесора шляхом делегування виконання допоміжних та системних завдань до окремих спеціалізованих обчислювачів, відомих як сопроцесори або акселератори, відіграє важливу роль у підвищенні швидкості обчислень на центральному процесорі.

Одним з ключових напрямків є використання багатоядерних процесорів та графічних обчислювальних одиниць (GPU), які дозволяють виконувати паралельні обчислення на великих масштабах. Це стало можливим завдяки поширенню масового використання многоядерних архітектур, що відкриває нові можливості для обробки декількох завдань одночасно.

Графічний процесор (GPU – англ. graphics processing unit) є важливим спеціалізованим акселератором, який винайденням вирішив одну з найважчих задач для CPU – рендерінг користувацького інтерфейсу. У спрощеному форматі, модель GPU можна описати як набір великої кількості однотипних простих процесорних елементів. Кожен з цих елементів є значно більш доступним за свої аналоги в CPU, що призвело до значного росту кількості ядер в GPU. У сучасних GPU зазвичай відсутні півтисячі ядер. Однак на практиці виявилося, що в гетерогенних комп'ютерних системах, що включають як CPU, так і GPU, якщо вони не задіяні в складні графічні завдання, обчислювальна потужність GPU є зайвою. Обчислення відбуваються настільки швидко, що більшість ядер GPU просто очікує наступного завдання від CPU.

Використання вільної обчислювальної потужності GPU, яка є значною та доступною, представляє собою логічне та очевидне рішення для збільшення продуктивності обчислень. З винайденням та інтеграцією програмованих шейдерних блоків в GPU виникла можливість програмування звичайних користувацьких обчислень на цих пристроях. Ця техніка отримала назву GPGPU (англ. General-purpose computing on graphics processing units, неспеціалізовані обчислення на графічних процесорах) і стала значним проривом в розвитку комп'ютерної техніки. Перші графічні процесори від Nvidia, що підтримували технологію CUDA (від англ. Compute Unified Device Architecture, уніфікована обчислювальна архітектура пристрою - реалізація техніки GPGPU від цієї компанії), не відрізнялись дешевизною та доступністю. Проте швидко з'явились інші реалізації GPGPU, зокрема фреймворк OpenCL, який був розроблений Apple для мови програмування C++. OpenCL поширюється під вільною ліцензією і дозволяє програмувати загальні обчислення для GPU, включаючи не лише від Nvidia, але і від інших виробників, таких як AMD. Наприклад, відеокарти від компанії AMD, які зазвичай є дешевшими та простішими, також підтримують OpenCL.

Отже, винайдення техніки GPGPU та розвиток технологій, таких як CUDA та OpenCL, не лише значно підвищили швидкість обчислень, але також дали змогу невеликим підприємствам, а також навчальним закладам, організувати власні потужні обчислювальні центри. Розподілені комп’ютерні системи, які ґрунтуються на гетерогенних системах з CPU та GPU, забезпечують достатню швидкість обчислень й водночас коштують значно дешевше, ніж системи, що побудовані виключно на великій кількості CPU або GPU. Практика також підтверджує, що техніка GPGPU стала важливою складовою стрімкого розвитку систем штучного інтелекту.

Сучасні принципи прискорення вирішення об’ємних задач в комп’ютерних системах основані переважно на концепції розподілених комп’ютерних систем, включаючи їх гетерогенні варіанти. Для досягнення максимальної продуктивності таких систем ключовим фактором у розробці цільового програмного забезпечення є збалансований розподіл навантаження на кожну паралельну підсистему розподіленої системи, а також на кожен кінцевий процесор або акселератор (якщо вони присутні), з урахуванням їхньої відмінності в продуктивності [4].

Крім того, технології квантових обчислень та обчислення з використанням квантових бітів представляють сучасні дослідження в області архітектур паралельної обробки, обіцяючи переваги у вирішенні складних обчислювальних задач.

Загалом, сучасні архітектури паралельної обробки інформації активно розвиваються в різних напрямках, враховуючи потреби високопродуктивних обчислень та обробки великих обсягів даних.

1.3 Класифікація архітектур комп'ютерних систем

Є кілька класифікацій архітектур паралельних комп'ютерних систем, проте найбільш прийнятною вважається класифікація, запропонована Майклом Фліном у 1966 році.

Його класифікація базується на аналізі потоку інформації, який розділяється на потоки даних між основною пам'яттю та процесором, а також потік команд, які виконує процесор [5]. Ця система класифікації визначає чотири основних архітектурних класи (рисунок 1.3):

- SISD = Single Instruction Single Data;
- MISD = Multiple Instruction Single Data;
- SIMD = Single Instruction Multiple Data;
- MIMD = Multiple Instruction Multiple Data.

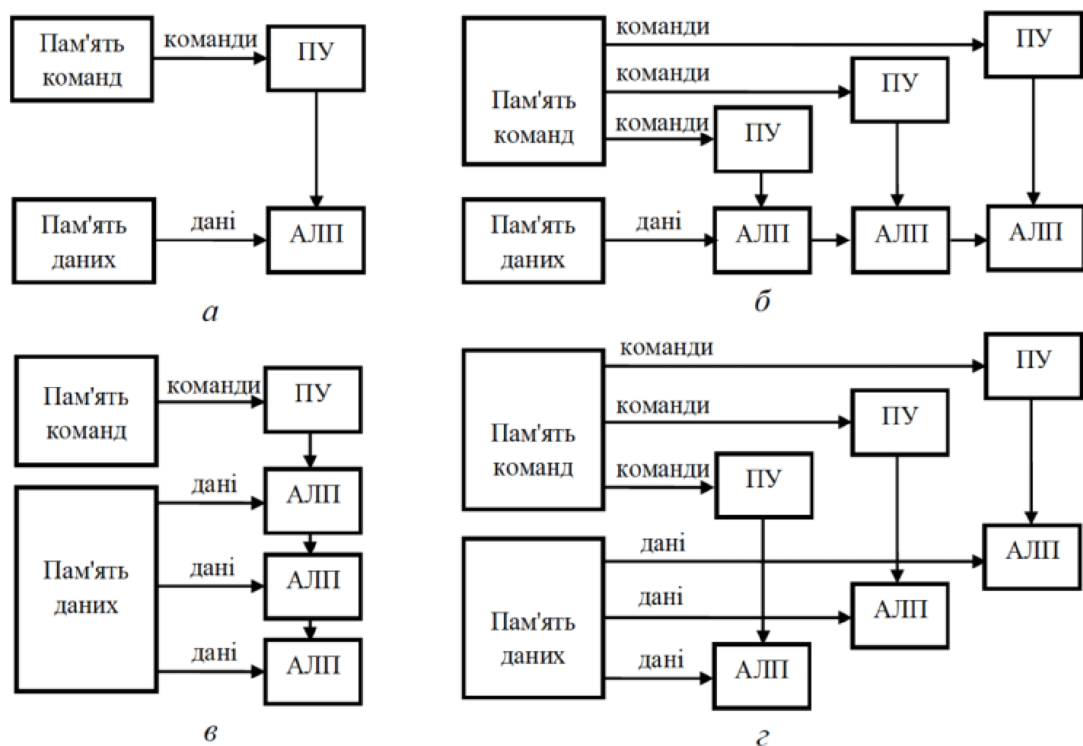


Рисунок 1.3 – Архітектура КС за класичною класифікацією Фліна: а) – SISD; б)–MISD; в) – SIMD; г) – MIMD

Класифікація комп'ютерів паралельної дії є завдяки різноманітності архітектур та систем доволі складною завданням. Незважаючи на широке використання класифікації Флінна, виникає потреба в більш точних та деталізованих підходах для врахування різноманітності сучасних систем – так звана розширена класифікація Фліна (рисунок 1.4).

Розподілу машин SIMD на підгрупи, відображає те, що навіть в межах одного класу може існувати значна різноманітність. У першій підгрупі представлені суперкомп'ютери та інші машини, які оптимізовані для роботи з векторами даних. У другій підгрупі знаходяться системи, такі як ILLIAC IV, які використовують незалежні АЛП для обробки команд.

Категорія MIMD також стала різноманітною, розгалужуючись на мультипроцесори і мультикомп’ютери. Мультипроцесори поділяються за реалізацією спільної пам'яті, існуючи у формах з пам'яттю спільного використання [6]. Це різновиди, такі як UMA (Uniform Memory Access), NUMA (Non-Uniform Memory Access) та CC-NUMA (Cache-Coherent Non-Uniform Memory Access). Мультикомп’ютери, з іншого боку, використовують передачу повідомлень для обміну інформацією між окремими вузлами.

Рисунок 1.4 – Розширена класифікація Фліна

Таким чином, існує велике різноманіття паралельних КС, які постійно вдосконалюються за своєю архітектурою, але для поліпшення існуючих є необхідність впровадження нових підходів до програмування.

2 ПРИНЦИПИ ТА МЕТОДИ ПАРАЛЕЛЬНОЇ ОБРОБКИ ІНФОРМАЦІЇ

2.1 Рівні паралельності в КС

Рівень паралельності в паралельних КС визначається ступенем структуризації одиниць даних та обчислень, на якому відбувається розпаралелювання обробки інформації [7]. Цей рівень визначається на кількох рівнях, серед яких можна виокремити:

- рівень завдань або процедур;
- рівень програм;
- рівень команд або арифметичних виразів;
- рівень розрядів.

Ці рівні визначають різні аспекти та розгалуження паралельності в обчисленнях у паралельних обчислювальних системах.

На рівні завдань, різні сегменти або "процеси" тієї самої програми виконуються паралельно, зі стримуванням кількості операцій обміну даними між цими процесами до мінімуму (рисунок 2.1).

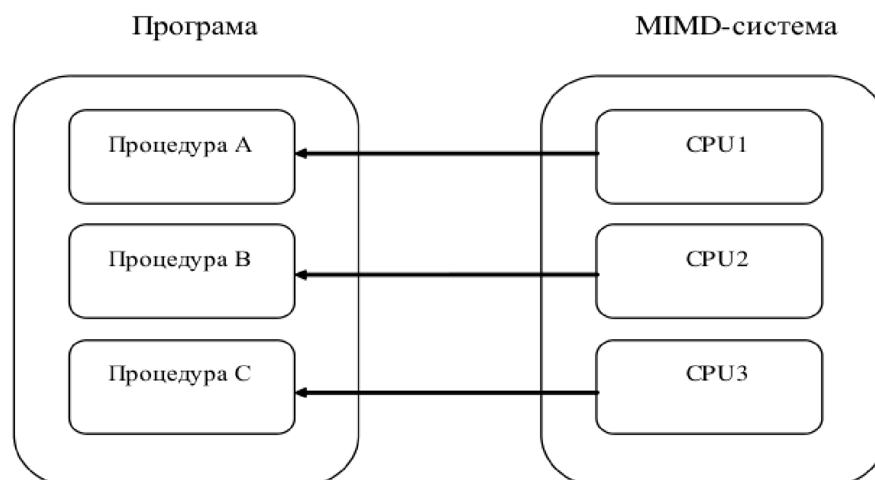


Рисунок 2.1 – Паралельність на рівні процедур

Головна мета процедурного рівня паралелізму полягає в реалізації загального паралельного оброблення інформації. Це досягається шляхом розбиття проблеми на паралельні завдання або частини, які обробляються різними процесорами. Такий підхід призначений для підвищення обчислювальної продуктивності.

На рівні програм одночасно або з часовим розподілом виконуються складні програми (рисунок 2.2). Комп'ютер, що виконує ці програми, не обов'язково має паралельну структуру; достатньо, щоб на ньому була встановлена багатозадачна операційна система, реалізована, наприклад, як система розподілу часу. У цій системі для кожного користувача планувальник виділяє відрізок часу залежно від його пріоритету. Кожен користувач отримує доступ до ресурсів центрального процесорного блоку лише протягом короткого періоду, після чого встає в чергу на обслуговування.

Рисунок 2.2 – Паралельність на програмному рівні

У випадку, коли в системі відсутня достатня кількість процесорів для всіх користувачів (або процесів), що, ймовірно, є найбільш типовою ситуацією, система моделює паралельне обслуговування користувачів за допомогою "квазіпаралельних" процесів.

На рівні команд (арифметичних виразів), арифметичні вирази виконуються паралельно покомпонентно, зокрема застосовуючи значно

простіші синхронні методи. Наприклад, при додаванні матриць (рисунок 2.3) операція синхронно розпаралелюється, й кожен процесор обчислює один (або декілька) елементів матриці.

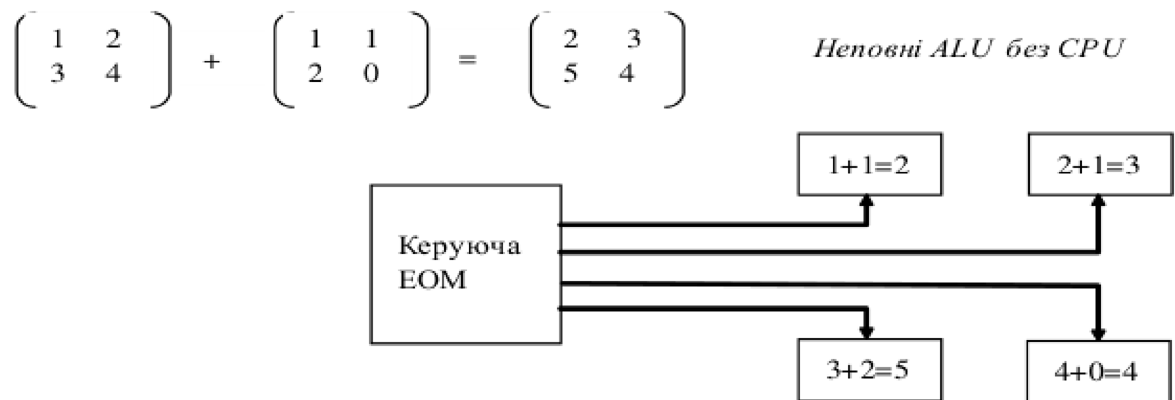


Рисунок 2.3 – Паралельність на рівні операцій

Використовуючи nn процесорних елементів, можна отримати суму двох матриць розмірністю nn за час виконання однієї операції додавання (за винятком часу, потрібного на виконання операцій читання та запису даних). Цьому рівню притаманні засоби векторизації, що відомі як "паралельність даних".

На рівні розрядів відбувається паралельне виконання бітових операцій в межах одного інформаційного слова (рисунок 2.4.). Паралельність на рівні бітів присутня в будь-якому працюючому мікропроцесорі. Наприклад, у 8-розрядному арифметико-логічному пристрої побітова обробка виконується паралельними апаратними засобами.

```
      01011101
AND   11011000
      01011000
```

Рисунок 2.4 – Паралельність на рівні бітів

В певній мірі рівень паралелізму обчислень, який реалізується в паралельній обчислювальній системі, залежить від виду паралелізму, що вбудований в архітектуру системи [8].

Використання методів паралельної обробки в мультипроцесорних системах є визначальним при розробці КС, що здатні вирішувати задачі у різних галузях науково-технічних обчислень, таких як прогнозування глобальних змін клімату, обчислення океанських течій, моделювання складних технічних систем та інші.

2.2 Сучасні моделі паралельної обробки інформації

2.2.1 Модель MapReduce

Фреймворк програмного забезпечення та модель програмування MapReduce використовуються для обробки великих обсягів даних. Map та Reduce – це дві стадії роботи програми MapReduce. В сучасних КС, орієнтованих на обробку даних, генерується величезний обсяг даних через постійне збирання інформації про особистості, бізнеси, системи та організації завдяки алгоритмам та додаткам. Складність полягає в тому, як швидко і ефективно переробити цей обсяг даних, не втрачаючи корисних висновків.

Загалом MapReduce розбиває вхідні дані на частини та розподіляє їх між іншими комп'ютерами. Вхідні дані розбиваються на пари ключ-значення. На комп'ютерах у кластері паралельні map-процеси обробляють розбиті дані. Процес reduce об'єднує результат у вихідну пару ключ-значення, після чого дані записуються у розподілену файлову систему Hadoop (HDFS).

Зазвичай програма MapReduce працює на тих самих комп'ютерах, що й розподілена файлова система Hadoop. Час виконання завдання драматично скорочується, коли фреймворк виконує завдання на вузлах, які зберігають дані. У першій ітерації MapReduce використовувалися кілька компонентів демонів, включаючи TaskTrackers та JobTracker.

TaskTrackers – це агенти, встановлені на кожному комп'ютері у кластері, для виконання завдань map та reduce. JobHistory Server – це компонент, що відстежує завершені завдання та зазвичай розгортається як окрема функція або разом із JobTracker. JobTracker - це головний вузол, який керує всіма завданнями та ресурсами у кластері.

Демони JobTracker та TaskTracker з ранніх версій MapReduce та Hadoop замінені компонентами ResourceManager та NodeManager "Ще одним договором про ресурси" або YARN, які були представлені разом з випуском MapReduce та Hadoop V.2. Процес подання завдань та планування у кластері обробляє ResourceManager, який встановлюється на головному вузлі. Він також керує виділенням ресурсів та моніторингом завдань.

NodeManager працює на робочих вузлах разом із Resource Manager для виконання дій та моніторингу використання ресурсів. NodeManager може використовувати інші демони для допомоги у виконанні завдань на рабочому вузлі. MapReduce (рисунок 2.5) використовує величезні розміри кластерів для виконання паралельних операцій для розподілу вхідних даних та компіляції виводів. Завдяки цьому можна розподіляти завдання між практично будь-якою кількістю серверів, оскільки розмір кластера майже не впливає на результати обробки завдання.

Отже, архітектура Hadoop в цілому та MapReduce роблять розробку програм простішою. Багато мов програмування підтримують MapReduce, включаючи C, C++, Java, Ruby, Perl та Python. Для генерування завдань без переймання за координацію або комунікацію між вузлами програмісти можуть використовувати бібліотеки MapReduce.

MapReduce регулярно повідомляє про статус кожного вузла головному вузлу. Якщо вузол не реагує, як очікувалося, головний вузол перерозподілить завдання на інші доступні вузли кластера. Це забезпечує стійкість і дозволяє MapReduce працювати на доступних серверах загального використання.

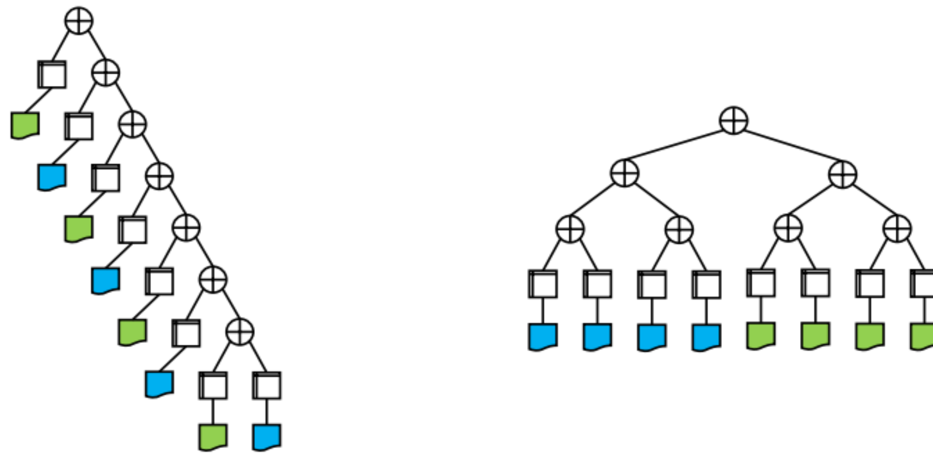


Рисунок 2.5 – Дерево обчислень моделі MapReduce

Програма MapReduce працює у трьох фазах: фазі map, фазі shuffle та фазі reduce. Завдання map або мапера полягає у обробці вхідних даних на цьому рівні. У більшості випадків вхідні дані зберігаються у файловій системі Hadoop як файл або каталог (HDFS). Функція мапера отримує вхідний файл по рядках. Мапер обробляє дані та створює кілька невеликих фрагментів даних. Shuffle and Sort (перемішування та сортування) - проміжні пари, створені на етапі Map, групуються за ключем і передаються до етапу Reduce. Вони також сортуються за ключем для ефективності передачі до задач Reduce. Етап Reduce отримує дані, згруповані за ключем, від етапу Shuffle and Sort. Кожна задача Reduce виконує функцію Reduce над кожним ключем, обробляючи його значення та генеруючи вихідні дані.

Наступні функції відрізняють MapReduce: висока масштабованість, універсальність, безпека, доступність, швидкість, надійність та висока доступність.

Фреймворк з відмінною масштабованістю Apache Hadoop MapReduce, через його здатність розподіляти та зберігати великі обсяги даних на численних серверах. Можна покращити можливості вузлів або додати будь-яку кількість вузлів (горизонтальна масштабованість), щоб отримати високу обчислювальну потужність. Завдяки програмуванню Hadoop MapReduce,

організації можуть виконувати додатки з використанням величезних наборів вузлів, можливо, з використанням тисяч терабайтів даних. Фреймворк MapReduce підтримує дані з різних джерел, включаючи електронну пошту, соціальні медіа та потоки кліків на різних мовах. Щодо масштабованості, обробка даних за допомогою старих, традиційних систем управління реляційними базами даних була не такою простою, як з системою Hadoop. У таких ситуаціях необхідно зменшити обсяг даних та виконувати класифікацію на основі припущень про те, які дані можуть бути важливими для організації, тобто видаляти сирий матеріал. Модель програмування MapReduce у масштабованій архітектурі Hadoop допомагає в таких ситуаціях.

Паралельна обробка, що використовується в програмуванні MapReduce, – один з його ключових компонентів. Завдання поділяються в програмній парадигмі для одночасного виконання незалежних дій. У результаті програма працює швидше через паралельну обробку, що спрощує обробку кожного завдання. Кілька процесорів можуть виконувати ці розділені завдання завдяки паралельній обробці. Внаслідок цього весь софтвер працює швидше. Кожного разу, коли колекція інформації надсилається на одну машину, ті ж самі дані передаються на інші вузли. Тому, навіть якщо один вузол виходить з ладу, резервні копії завжди доступні на інших вузлах та можуть бути відновлені за потреби. Це забезпечує високу доступність даних [9].

Функція стійкості до збоїв Hadoop забезпечує, що навіть якщо один із DataNode виходить з ладу, користувач все ще може отримати доступ до даних з інших DataNode, які мають копії цих даних. Крім того, кластер Hadoop з високою доступністю включає два або більше активних та пасивних NameNode, що працюють в режимі гарячої резервності. Активний NameNode - це активний вузол. Пасивний вузол - це резервний вузол, який застосовує зміни, внесені в журнали редагування активного NameNode, до свого простору імен.

2.2.2 Модель SIMD

SIMD – це техніка паралельних обчислень, яка дозволяє виконувати одну інструкцію одночасно на кількох елементах даних. Тобто дозволяє виконувати одну й ту ж операцію на кількох елементах даних за один тактовий цикл. На відміну від архітектур із однією інструкцією та однією даними (SISD), де кожна інструкція працює з одним елементом даних за раз.

Ця модель добре підходить для завдань, які включають виконання однієї й тієї ж операції на кількох елементах даних одночасно. У контексті штучного інтелекту / машинного навчання та науки про дані, SIMD може бути застосований до широкого спектру операцій, включаючи, але не обмежуючись. SIMD може використовуватися для прискорення математичних операцій, таких як додавання, віднімання, множення та ділення на великих масивах або матрицях. Це особливо корисно в обчисленнях лінійної алгебри, що часто зустрічаються в алгоритмах машинного навчання. SIMD можна використовувати для обробки великого обсягу даних зображень та звуку ефективно. Операції, такі як згортка, фільтрація та вилучення ознак, можна прискорити за допомогою інструкцій SIMD. SIMD може бути використаний для прискорення трансформацій та маніпуляцій даними, таких як операції над окремими елементами, сортування та пошук у великих наборах даних. Це особливо важливо при обробці та аналізі даних у реальному часі.

Концепція SIMD сягає свої коріння у 1960-х роках, коли вона була вперше введена в контексті векторної обробки. Векторні процесори, такі як CDC STAR-100 та Cray-1, були ранніми прикладами апаратних архітектур, що реалізували принципи SIMD. Ці машини були розроблені для обробки масивів елементів даних паралельним та ефективним способом. З часом SIMD еволюціонувала і стала стандартною функцією у сучасних процесорах. На сьогоднішній день набори інструкцій SIMD зустрічаються у широкому спектрі процесорів та графічних процесорів, включаючи Intel SSE (Streaming

SIMD Extensions), ARM NEON та NVIDIA CUDA. Ці набори інструкцій надають набір спеціалізованих інструкцій, які можна використовувати для використання паралельності SIMD.

У моделях глибокого навчання для завдань розпізнавання зображень часто використовуються згорткові шари. SIMD може бути використаний для прискорення операції згортки, що призводить до швидшого часу виведення результатів.

Попередня обробка даних є важливим етапом у робочих процесах штучного інтелекту / машинного навчання. SIMD може бути використаний для прискорення операцій, таких як масштабування ознак, нормалізація та трансформація даних, що призводить до швидшого підготовчого часу даних.

Постійний розвиток штучного інтелекту / машинного навчання та науки про дані робить оптимізацію та прискорення обчислень все більш цінним. Глибоке розуміння SIMD та його застосувань може відокремити вчених-даналітиків та практиків у галузі штучного інтелекту / машинного навчання, зробивши їх більш здатними розробляти ефективні алгоритми та використовувати апаратні можливості [10].

SIMD може бути особливо корисним при роботі з великими наборами даних, обчислювально інтенсивними моделями або вимогами до обробки в реальному часі. Це дозволяє краще використовувати наявні апаратні ресурси та може призвести до значного зростання продуктивності. Інструкції SIMD часто вимагають вирівнювання даних в пам'яті для оптимальної продуктивності. Забезпечення належного вирівнювання структур даних може уникнути можливих штрафів продуктивності. Ретельне вивчення шаблонів доступу до даних може покращити використання SIMD. Послідовний доступ та мінімізація залежностей даних можуть призвести до кращої продуктивності. Сучасні компілятори часто надають оптимізації, які автоматично векторизують код за допомогою інструкцій SIMD. Розуміння параметрів компілятора та використання відповідних прапорців може допомогти максимізувати продуктивність.

2.2.3 Dataflow model

Модель потоків даних (Dataflow Model) – це концептуальний підхід до організації обчислювальних процесів, де програми розглядаються як мережа обробки даних, в якій кожен обчислювальний елемент (операція) виконується тільки тоді, коли всі його вхідні дані готові та доступні. Це поняття стало одним з ключових у сучасних комп'ютерних системах та програмному забезпеченні.

У моделі потоків даних, дані переміщуються через систему в спосіб "потоку" або "потоків". Кожна операція чекає на вхідні дані, обробляє їх і генерує вихідні дані, які стають вхідними для наступних операцій. Цей процес визначає порядок виконання операцій та забезпечує логічний потік даних в системі.

Оскільки кожна операція не залежить від попередніх результатів і може виконуватися незалежно, модель потоків даних дозволяє ефективно виконувати декілька операцій паралельно. Це значно покращує продуктивність системи та прискорює обчислення.

Завдяки направленості потоків та паралельному виконанню, дані можуть легко розподілятися між різними вузлами обчислень або обчислювальними пристроями. Це робить модель потоків даних відмінним інструментом для систем, що потребують масштабування.

Підхід Dataflow дозволяє оптимізувати використання ресурсів, оскільки кожна операція працює тільки з тими даними, які необхідні для її виконання. Це сприяє зменшенню навантаження на систему та підвищенню її продуктивності. У сферах аналізу великих обсягів даних (Big Data), модель потоків даних використовується для оптимізації та прискорення обчислень, що дозволяє швидше аналізувати величезні обсяги інформації.

В графічних процесорах (GPU), які широко використовуються в обчислювальних завданнях, модель потоків даних Dataflow використовується

для розподілу та паралельного виконання обчислень, підвищуючи продуктивність.

У сферах, пов'язаних з обробкою медіаданих в реальному часі, модель потоків даних використовується для неперервної обробки великих обсягів даних, що дозволяє побудувати швидку та ефективну систему обробки відео та аудіо. Модель потоків даних стає все більш важливою в світі сучасних обчислювальних систем. З розвитком областей штучного інтелекту, Internet of Things (IoT), облачних технологій та інших, потреба у швидкій та ефективній обробці даних стає критичною. Модель Dataflow надає засоби для оптимізації цих процесів [10].

У майбутньому, розвиток технологій обробки даних, включаючи розширення областей штучного інтелекту, машинного навчання та інших, може призвести до ще більшого використання моделі потоків даних. Це може стати важливим для оптимізації обчислень у реальному часі та розвитку нових обчислювальних архітектур.

2.3 Технології програмування паралельних КС

Динамічний розвиток паралельних комп'ютерних систем вимагає постійного удосконалення та створення нових моделей і технологій для їх програмування. З проведених аналізів видно, що ключовим аспектом є вибір рівня абстракції у прийнятій моделі. Використання низькорівневих абстракцій може забезпечити максимальну ефективність системи, оскільки розробник працює з концепціями, найближчими до самої системи. Однак це вимагає адаптації задачі до абстракцій системного рівня, що лише теоретично може призвести до значного прискорення. Також, через сильну залежність системного та прикладного рівнів, принципові помилки можуть виникнути на обох рівнях. З іншого боку, використання високорівневих абстракцій дозволяє працювати з концепціями, найближчими до прикладного рівня та бізнес-логіки. Це дозволяє звільнити розробника від необхідності

організації системної взаємодії, перекладаючи цю відповідальність на систему. Проте, можливі принципові помилки тільки на прикладному рівні.

Сучасні методи організації паралельних обчислювальних систем все більше використовують високорівневі абстракції з метою спрощення роботи розробників. Головна ідея полягає в тому, щоб надати можливість розробникам працювати з математичними та бізнес-логічними об'єктами, використовуючи паралелізм на загальному рівні, із уникненням необхідності вручну організовувати всі комунікації, синхронізації та інші аспекти.

Деякі сучасні математичні бібліотеки надають можливість використання функцій як у послідовному, так і у паралельному режимі, зберігаючи при цьому інтерфейс функції [10]. Також існують підходи, що дозволяють описувати паралелізм лише на абстрактному рівні задач та їхніх зв'язків [9]. Важливо відзначити, що всі ці підходи націлені на використання в системах зі спільною пам'яттю.

Найбільш перспективні паралельні комп’ютерні системи будуються на засадах розподілених систем. Це стосується не лише систем з акселераторами, але й усіх розподілених систем взагалі. Такі системи базуються на моделях, що відповідають системам із локальною пам'яттю.

Адаптація моделей із високим рівнем абстракції у таких системах вимагає від програмного забезпечення, яке втілює функціонування таких систем, вирішення ряду завдань. Серед них — забезпечення автоматичного розбиття та передачі даних, надсилання цих частин конкретним вузлам системи та, за необхідності, збір результатів від цих вузлів. При цьому необхідно враховувати можливу гетерогенність системи, тобто наявність різних складових елементів з різними характеристиками та потужністю, а також продуктивність комунікативних каналів між елементами системи.

Окремо необхідно виділити задачу доступності даних, яка є ключовою. В системах зі спільною пам'яттю, у зв'язку з використанням спільного адресного простору всіма потоками, ця задача значно спрощується, тому що дані доступні для всіх потоків за замовчуванням. А в системах з локальною

пам'яттю та/або акселераторами додатково необхідно вручну забезпечувати доступність даних, шляхом копіювання їх в пам'ять обчислювального елементу системи. Передавати повністю всі дані було б простіше, але зазвичай мережевий час до 10000 разів дорожчий за процесорний, тому чим менші обсяги даних будуть курсувати системою, тим меншими будуть простої обчислювачів. Тому планувальники обчислень намагаються розташувати дані таким чином, щоб забезпечити максимальну локалізацію даних, тим самим максимально зменшити комунікативні затримки системи в цілому.

2.3.1 Технологія низькорівневого передавання повідомлень

MPI (Message Passing Interface) є широко використовуваною технологією для реалізації моделі низькорівневої передачі повідомлень в системах з локальною пам'яттю. Ця технологія дозволяє обмінюватися повідомленнями між різними елементами програми і використовується для побудови паралельних програм, які виконуються на розподілених системах. Основним принципом MPI є обмін повідомленнями між окремими задачами (процесами) у розподіленій системі. MPI зазвичай використовується в контексті процедурної парадигми мови програмування C, і він надає функції та процедури для створення, керування та обміну повідомленнями між процесами. Однак існують адаптації MPI для інших мов програмування, таких як C++. MPI дозволяє розробникам ефективно обирати, як саме будуть обмінюватися повідомленнями між задачами, і надає розширений інтерфейс для керування процесами та комунікацією між ними. Використання MPI вимагає від розробників більшого рівня відповідальності за управління розподіленими ресурсами, але в той же час це дозволяє ефективно використовувати ресурси в розподіленому середовищі.

Буферизовані передачі в MPI передбачають копіювання даних від відправника до виділеного буфера, дозволяючи призупинити обчислення

лише на час цього копіювання, а також можливість створення буферів у процесах-приймачах. Це забезпечує відкладення передачі даних до моменту їхньої активної потреби процесом-приймачем, що сприяє більш ефективній роботі планувальників. Однак буферизовані передачі вимагають додаткового обсягу пам’яті. Повідомлення MPI можуть виконувати обробку відправлених даних, що дозволяє адаптувати дані відповідно до особливостей роботи з пам’яттю операційної системи вузла-приймача [10].

MPI у сучасних реалізаціях також має адаптації для ситуацій, коли вузли-приймачі розташовані в межах одного процесора. У цьому випадку не відбувається буферизація повідомлень, а використовуються засоби операційної системи для розподілу пам'яті між процесами. Проте основними недоліками MPI залишаються його низькорівневість, особливо прив'язка до процедурної парадигми, орієнтація на гомогенні системи та канали зв'язку. Це також призводить до відсутності оптимізованих для цього планувальників та відсутності можливості динамічної регуляції зерна паралелізму.

2.3.2 Технологія розпаралелювання з застосуванням препроцесору

Стандарт OpenMP представляє собою один з найбільш поширених відкритих стандартів для паралельного програмування, що використовує потоки та спільну пам'ять. Розвиток OpenMP, який спостерігаємо сьогодні, вимагав певного часу, й цей стандарт продовжує розвиватися. Зародження OpenMP почалося наприкінці 1990-х, коли кілька виробників обладнання представили свої власні реалізації. У 1994 році була неуспішна спроба стандартизації цих реалізацій у проекті стандарту ANSI X3H5. Тільки після впровадження широкомасштабних багатоядерних систем наприкінці 90-х років відбулося відновлення підходу до OpenMP, що призвело до виходу першого стандарту OpenMP у 1997 році.

В наш час OpenMP є стандартним портативним API для розробки паралельних програм з використанням спільної пам'яті та потоків. Його

визначає простота використання, швидке впровадження та необхідність невеликих змін у вихідному коді, які, як правило, вводяться у вигляді прагм або директив. У мовах C і C++, термін "прагма" відноситься до операторів препроцесора, тоді як у Fortran, як директиви, вони представлені у вигляді коментарів, щоб забезпечити стандартний синтаксис програми, коли OpenMP не використовується. Ці терміни, "прагма" і "директива", часто використовуються взаємозамінно. Щоб вказати компілятору місце для ініціювання потоків OpenMP, необхідно використовувати вказані прагми або директиви, які визначаються у коді. Важливо зазначити, що, хоча для використання OpenMP необхідний підтримуючий його компілятор, більшість сучасних компіляторів вже включають таку підтримку за замовчуванням.

OpenMP забезпечує доступність процесу розпаралелювання навіть для початківців, що дозволяє легко ознайомитися з масштабуванням програм за межі одного ядра. Його простота використання через прагми і директиви дозволяє швидко впроваджувати паралельні блоки коду.

Описана тенденція до використання паралельних абстракцій вищого рівня, які не потребують значних змін у структурі та інтерфейсі програми, відкриває шлях до спрощення паралельного програмування та зменшення вимог до високої кваліфікації розробників.

Ця технологія дозволяє організовувати паралельне виконання коду з допомогою невеликих змін в послідовному коді програми. Однак для досягнення ефективності використання OpenMP часто вимагає ручного позначення залежностей між даними та різними частинами програми.

Оскільки внесення директив до вже існуючих програм не вимагає значних модифікацій вихідного коду, технологія OpenMP знаходить широке застосування при введенні паралелізму до існуючих програм. Багато сучасних трансляторів підтримують директиви препроцесору, які постачаються бібліотекою OpenMP 3.0 [22-24]. Розробка OpenMP орієнтована на результати аналізу статистичних даних складних наукових обчислень та реалізацій алгоритмів. Регулярний паралелізм та виконання

циклічної обробки алгоритмів займає в середньому до 80% часу виконання програм [25].

Для ефективного використання паралелізму необхідно не тільки пряма реалізація, але й підтримка та оптимізація до особливостей архітектури цільової системи та реалізованого алгоритму, включаючи динамічний режим роботи. У OpenMP була включена бібліотека часу виконання, яка відповідає за ці обов'язки. Крім механізмів самого паралелізму, ця бібліотека також містить потужні механізми статичного та динамічного планування. Розробник може вибрати тип планування, який він вважає найбільш доцільним для конкретної ситуації. Варто відзначити, що ці механізми планування не заміщують планувальники операційної системи, але доповнюють їх, дозволяючи більш гнучко керувати ресурсами, виділеними системою для програми.

OpenMP надає розробникам спеціальні директиви для різних режимів доступу до даних у паралельних програмах. Ці директиви дозволяють позначити дані як заборонені для доступу з усіх потоків, спільні для всіх потоків без копіювання, розподілені між потоками з копіюванням на час виконання потоків, розподілені між потоками з копіюванням, де оригінал змінної модифікується лише з потоку, який закінчить роботу останнім, розподілені між потоками з копіюванням, де оригінал змінної модифікується за результатами роботи всіх потоків з виконанням редукування, а також розподілені між потоками з виділенням відповідної пам'яті та ручним описом процесу копіювання. Ці директиви допомагають розробникам ефективно управляти доступом до даних у паралельних програмах.

Загалом, доступ до даних може відрізнятися для різних структур даних, і можна встановлювати спосіб доступу за замовчуванням. Система також може автоматично копіювати дані при необхідності, без явного втручання користувача. Однак для уникнення непередбаченої поведінки програми при використанні спільних ресурсів для багатьох потоків, все ще вимагається ручне визначення критичних секцій.

Ці особливості визначають орієнтацію технології OpenMP на роботу в системах зі спільною пам'яттю. Таким чином, вважається, що швидкість доступу до даних та їх копіювання однакова для кожного потоку, припускаючи однакові показники продуктивності для кожного ядра системи. Навіть при можливості використання технології в неоднорідних системах пам'яті (наприклад, NUMA-системах), врахування неоднорідності пам'яті не враховується, що може вплинути на продуктивність програми в цілому. Внутрішня підсистема керування ресурсами також орієнтована на цю концепцію, використовуючи метод розподілу завдань через викрадення (work-stealing), який конвертує програму в готовий до виконання набір підзавдань та формує чергу їх виконання.

Зазначені дві особливості визначають орієнтацію технології OpenMP на роботу в системах зі спільною пам'яттю. У таких системах припускається, що швидкість доступу до даних та швидкість їх копіювання однакові для кожного потоку, або вважаються аналогічними абсолютним показникам продуктивності кожного ядра системи. Хоча технологія OpenMP може бути також застосована в системах із неоднорідною пам'яттю (наприклад, NUMA-системах), врахування неоднорідності пам'яті в цьому випадку не відбувається, що може суттєво вплинути на швидкодію програми в цілому. Внутрішня підсистема керування ресурсами також спрямована на цю концепцію, використовуючи підхід розподілу робіт через викрадення (англ. work-stealing). Цей метод передбачає перетворення програми в готовий до виконання набір підзавдань та створення черги для їх виконання.

Для адаптації технології розпаралелювання на основі директив препроцесору до використання в більш перспективних системах із локальною пам'яттю, запропоновані два різних підходи.

Перший підхід до адаптації технології розпаралелювання на базі директив препроцесору OpenMP до систем із локальною пам'яттю передбачає автоматичний перехід до технології на базі інтерфейсу низькорівневої передачі повідомлень, зокрема за допомогою організації додаткового

транслятора. Цей транслятор здійснює перетворення коду стандарту OpenMP в код стандарту MPI (Message Passing Interface). Однак цей підхід виникає з принципових складнощів та включає в себе ряд проблем. Важливою є та, що OpenMP не підтримує зміну змісту операцій в процесі виконання, що вимагає строгого дотримання послідовної несуперечливості. Це може призводити до значних накладних витрат і складнощів у міжпотоковій взаємодії, а також до чекання потоків на доступ до даних. Крім того, не враховується потенційна гетерогенність системи, оскільки для передачі даних використовуються блокуючі колективні передачі даних MPI. Ці передачі вимагають часу, пропорційного відстані між комунікативно віддаленими вузлами, що може значно уповільнити ефективність системи в цілому.

Другий підхід, який не вимагає перетворень вихідних кодів та використовує розширення трансляторів мов C/C++, відрізняється від першого. У цьому випадку використовуються спеціалізовані бібліотеки для кластерних систем від компанії Intel, які розширюють транслятори. Основна ідея полягає в аналізі коду щодо доступу до даних з різних частин програми. За результатами аналізу формуються оптимальні передачі лише тих даних, які є необхідними вузлам, зменшуючи локальність даних та комунікативні витрати часу.

Цей підхід також має свої обмеження, наприклад, для операцій з покажчиками аналіз даних може бути ускладненим, і дані можуть відтворюватись на кінцевих вузлах не завжди вірно. Також цей підхід орієнтований на застосування в гомогенних системах кластерного типу, де існують рівноцінні канали комунікації.

Обидва підходи, хоч і розширюють технологію розпаралелювання для використання в системах з локальною пам'яттю, не враховують гетерогенність системи, що може бути важливим аспектом при розробці для сучасних архітектур.

Навіть при спрощенні досягнення помірного рівня паралелізму за допомогою OpenMP, оптимізація може стати завданням складним. Однією з

причин цього є розслаблена модель пам'яті, яка допускає стан гонитви потоків. Розслаблена модель означає, що значення змінних в основній або кеш-пам'яті не оновлюються негайно. Це може бути ефективним, але призводити до різних результатів від виконання до виконання через часові відмінності між операціями з пам'яттю в кожному потоці, що має спільні змінні. Давайте розглянемо кілька термінів: розслаблена модель пам'яті – значення змінних в основній або кеш-пам'яті всіх процесорів не оновлюються негайно; стан гонитви – ситуація, коли можливо кілька результатів, і результат залежить від часу виконання потоків.

Використання OpenMP для реалізації паралельності може бути достатньо простим, але оптимізація цього процесу може становити виклик через розслаблену модель пам'яті, яка допускає можливість стану гонитви між потоками. Ця модель пам'яті затримує оновлення основної пам'яті, що може викликати потенційні відмінності в результатах через різні часові затримки між операціями пам'яті в різних потоках. При цьому OpenMP відрізняє публічні змінні, які можуть бути доступні та модифіковані будь-яким потоком, від приватних змінних, що є локальними для конкретного потоку. Для розуміння цих термінів слід мати певне уявлення про те, як взаємодіє пам'ять у програмі, що використовує паралельні потоки [9].

Основна увага директив OpenMP спрямована на спільне використання ресурсів та співпрацю, а не на явному визначенні пам'яті чи розташуванні даних. Обсяг пам'яті для змінних визначається неявними правилами, які вимагають від програмістів розуміння їх. Методи керування пам'яттю, такі як "перший дотик", відводять пам'ять поблизу того потоку, який першим звертається до неї. "Перший дотик" гарантує, що фізично виділена пам'ять створюється при першому доступі до неї, що є важливим при роботі з кількома областями пам'яті або архітектурами NUMA (нерівномірного доступу до пам'яті).

Для забезпечення коректної взаємодії між потоками, OpenMP використовує операції бар'єру та очищення. Операція очищення гарантує, що

значення переміщується між потоками для уникнення умов конкуренції, тоді як бар'єр синхронізує потоки та скидає локально змінені значення. Однак важливо уникати надмірної синхронізації, яка може виникнути внаслідок частих бар'єрів та операцій очищення, особливо в поєднанні з невеликими паралельними областями, оскільки це може призвести до зниження продуктивності.

OpenMP має обмеження на одному вузлі, й його можливості розширення пам'яті обмежені пам'яттю на даному вузлі. Для паралельних додатків з високими вимогами до пам'яті, OpenMP може використовуватися разом із методами паралельної роботи з розподіленою пам'яттю, такими як стандарт MPI (рисунок 2.6).

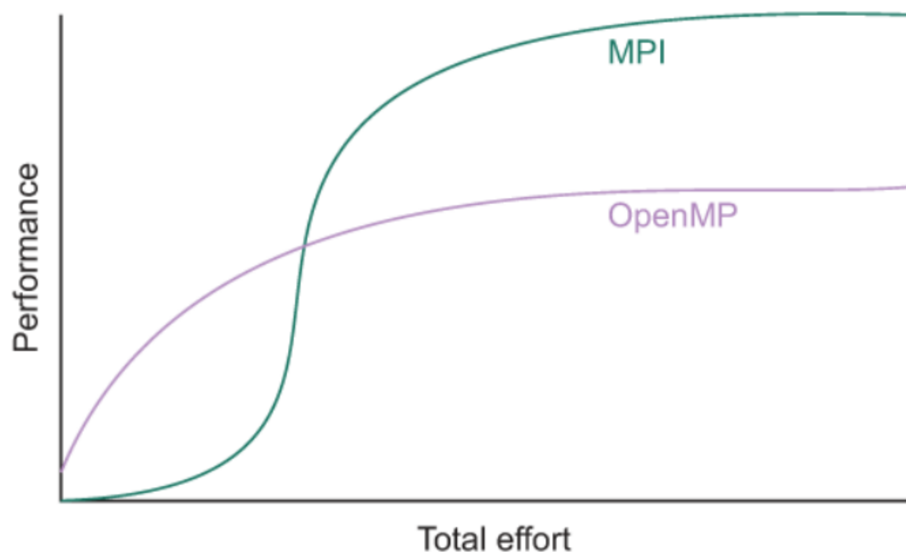


Рисунок 2.6 – Візуалізація технологій програмування

OpenMP, розвиваючись з часом, вводить нові функції та можливості, такі як паралелізм завдань, покращення паралелізму циклів, векторизація та розвантаження на прискорювачі, такі як графічні процесори.

Існують три різні варіанти використання OpenMP: OpenMP на рівні циклу, високорівневий OpenMP та поєднання MPI із OpenMP. OpenMP на

рівні циклу є ефективним, коли програма потребує помірного прискорення та має достатньо ресурсів пам'яті. Цей метод дозволяє швидко та легко впроваджувати паралелізм потоків у програму, особливо для циклів із високовитратними обчисленнями. Використання паралелі OpenMP досягається за допомогою прагм або директив parallel do перед циклами.

Високорівневий OpenMP призначений для додатків, що мають на меті кращу паралельну продуктивність. Він використовує низхідний підхід, охоплюючи всю систему та взаємодіючи з системою пам'яті, ядром та апаратним забезпеченням. Цей підхід усуває витрати на запуск потоку та накладні витрати на синхронізацію, що забезпечує покращену масштабованість. MPI в поєднанні із OpenMP використовується для розширення паралелізму розподіленої пам'яті. Це передбачає використання OpenMP на обмеженій групі процесів для додавання додаткового рівня паралельної реалізації для високої масштабованості. Це може бути корисним в області нерівномірного доступу до пам'яті (NUMA), де вартість доступу до пам'яті однакова. Гібридні реалізації MPI + OpenMP також можна використовувати для використання гіперпотоків для кожного процесора. Вивчення методів OpenMP на рівні циклу є достатнім для гібридного підходу, але рекомендується поступовий перехід до більш ефективної та масштабованої реалізації OpenMP. Такий підхід дозволяє поліпшити час доступу до пам'яті та забезпечити баланс навантаження, усуваючи повільний доступ до пам'яті в області нерівномірного доступу до пам'яті (NUMA). Використання політики ОС "першого дотику" може призвести до підвищення продуктивності на 10-20%. Важливо також враховувати вплив конфігурації NUMA на продуктивність, особливо з процесором Skylake Gold 6152, чий продуктивність значно знижується під час доступу до віддаленої пам'яті. Для визначення конфігурації системи рекомендується використовувати команди numactl і numastat у Linux.

2.3.3. Технологія обчислень з відвантаженням

Технологія відвантажених обчислень використовує акселератори для обчислювальних процесів. Будь-яка система з акселератором автоматично вважається не лише розподіленою, а й гетерогенною. Оскільки система є розподіленою, то найбільш підходящою моделлю паралельного програмування для неї є відправка повідомлень. Проте немає жодних перешкод для розподілення даних між центральним процесором та акселератором, та реалізації обчислень на кожному з них окремо як на системах зі спільною пам'яттю.

Найпоширенішим типом акселераторів наразі є графічні процесори. Перші графічні процесори компанії Nvidia, які підтримували технологію CUDA (Compute Unified Device Architecture- уніфікована обчислювальна архітектура пристрою) реалізація техніки GPGPU від цієї компанії, не відрізнялись дешевизною та доступністю. Проте досить швидко з’явились інші реалізації GPGPU, серед яких варто відмітити фреймворк OpenCL, який дозволяє програмувати загальні обчислення для GPU не лише від компанії Nvidia, а, наприклад, і компанії AMD. Також, окрім того, що OpenCL підтримує більше GPU ніж CUDA, зазвичай такі GPU дешевші та простіші, аж до звичайних користувацьких відеокарт. Технології Intel Threading Building Blocks та CUDA-MPI надають підтримку акселераторних обчислень в системах зі спільною пам'яттю (Intel TBB) або локальною (CUDA-MPI). У випадку CUDA-MPI вимагається повнозв'язність та гомогенність на рівні каналів зв'язку, архітектури вузлів та продуктивності вузлів. Хоча гомогенність не є безпосередньою умовою функціонування, методи балансування навантаження, вбудовані в ці технології, не враховують можливу значну різницю в потужності складових елементів системи, таких як процесори та акселератори що може призвести до вибору одного з елементів та одного з каналів як еталону, й втрати частини потужності продуктивніших вузлів.

2.3.4 Технологія обчислень з відкладенням

Класична модель обчислень передбачає, що останньою дією команди, процедури або функції є повернення результату (явне або неявне). Тобто, ланцюжок роботи програми передбачає, що користувач сам декларує прямий порядок виклику функцій, кожна з яких буде повертати результат, який може бути використаний наступними за порядком функціями.

Технологія відкладених обчислень передбачає інший порядок – рекурсивний. Ідея полягає в тому, що відкладена функція не буде виконуватись в програмі доти, допоки результат її роботи не буде явно запитано в коді. Це передбачає те, що виконання функції можна відкладати безкінечно довго, аж до критичного часу, тобто моменту першого запиту її результату в коді. Цей зручний інструмент дозволяє планувальникам (ручним або автоматичним планувальникам середовища виконання або операційної системи) розширити можливості до розподілу навантаження на системні ресурси, пам'ять та мережу, що створює значний заділ до покращення результатів роботи цих планувальників. Також відкладені обчислення є важливим елементом інтерпретованих мов програмування.

Найчастішим втіленням та застосуванням на практиці відкладених обчислень є відкладені функції передачі даних та функції зворотного виклику (коллбеки). В загальному, практика використання відкладених функцій та інструментів їх реалізації присутні в багатьох сучасних мовах програмування [11], їх концепція має загальну назву "ліниві обчислення" (англ. lazy evaluation). Ця концепція має три основні складові: відкладені, мінімальні та ынкрементальні обчислення.

Відкладені обчислення (англ. deferred evaluation) – обчислення виконуються не за порядком їх опису в коді, а лише за безпосереднім їх запитом.

Мінімальні обчислення (англ. circuit evaluation) – обчислення виконуються рівно в тому обсязі, якого достатньо для отримання результату.

Широко застосовуються для тонкої оптимізації на рівні мови. Наприклад, в мовах Java та C#, якщо в умовній конструкції з логічним оператором AND перший операнд false, то нема потреби перевіряти другий операнд, оскільки результат в будь-якому випадку буде false;

Інкрементальні обчислення (англ. incremental evaluation) – для отримання кожного окремого значення обчислення виконуються лише один раз. Їх реалізація найчастіше пов’язана із використанням чистих функцій, що є однією з основ функціональної парадигми програмування. Чистою є функція, яка має єдиний результат, значення якого визначається лише аргументами. Із цього випливає, що для кожної окремої можливої комбінації вхідних значень результат завжди буде незмінним, і відповідно, при повторній подачі такої комбінації аргументів немає потреби вираховувати тіло функції, а достатньо повернути попередній результат. Для реалізації цього повернення в сучасних мовах (як на внутрішньому, так і на прикладному рівнях) використовуються механізми мемоізації – збереження в пам’ять результатів попередніх викликів чистих функцій.

Якщо функція зворотного виклику виконується асинхронно в окремому потоці або процесі, це відкриває можливості для реалізації паралелізму в мовах, які спочатку були однопоточними, таких як JavaScript та Node.js. Також, підтримка асинхронних паралельних функцій зворотного виклику була реалізована в мовах C# та Java, а з 2011 року ця можливість була додана до стандартної бібліотеки Futures мови C++ [12].

При проектуванні сучасних КС з використанням засобів паралельного програмування спостерігається тенденція приховувати всю низькорівневу системну реалізацію, надаючи розробникам можливості оперувати більш високорівневими абстракціями, які дозволяють не вимагати адаптувати прикладні задачі та бізнес-логіку до особливостей архітектури цільової системи, тим самим даючи можливість зосередитись на реалізації прикладної задачі, додатково забезпечуючи уникнення потенційних помилок на рівні паралельної системи. Так, відповідно, технології відвантажених обчислень

стають дедалі популярнішими в гетерогенних паралельних системах, де використовуються акселератори.

Так, сучасні засоби паралельного програмування для систем з локальною пам'яттю перш за все орієнтовані на гомогенні системи, зокрема на повнозв'язні кластерні. Навіть ті, які підтримують відвантажені обчислення, зазвичай передбачають використання гомогенних акселераторів. При цьому в них не передбачається застосування технологій відкладених обчислень.

Так, описана проблема полягає не в тому, що система не буде працювати у разі гетерогенності її вузлів, а в тому, що балансування навантаження, яке є невідмінною частиною ефективної роботи будь-якої паралельної чи розподіленої системи, буде виконуватись не оптимально, відповідно до коефіцієнтів прискорення та ефективності вузлів та елементів вузлів системи, які будуть нижчими, ніж могли би бути. Існує проблема відсутності методу балансування навантаження в розподілених КС, при роботі якого одночасно враховувалися би такі фактори, як: гетерогенність кінцевих вузлів системи; наявність неоднорідних акселераторів в цих вузлах; потенційну неповнозв'язність системи; гетерогенність комунікативного середовища системи.

Хоча модель із автоматичним створенням потоків та модель із відсутніми блокуванням є найбільш абстрактними та найперспективнішими, вони поки що мають обмеження на класи задач, які можуть бути реалізовані в таких системах, і самі являються складними в реалізації, перш за все через відсутність поняття даних як елементу паралельної задачі в таких системах. Оптимальніше наразі зупинити вибір на моделі із динамічним створенням потоків, увівши до нього деякі особливості автоматичного створення потоків, такі як автоматичний поділ задач та даних за заданим розробником шаблоном (наприклад, як це робиться в реалізації механізму Fork-Join (RecursiveTask, RecursiveAction) пакету java.util.concurrent мови Java) [13] та автоматичну регуляцію зерна паралелізму.

3 РОЗРОБКА МЕТОДУ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ПАРАЛЕЛЬНОЇ КОМП’ЮТЕРНОЇ СИСТЕМИ

У якості методу підвищення ефективності КС пропонується вдосконалення методу перерозподілу навантаження в комп'ютерних системах, а також способу організації таких систем. Система, що вдосконалюється, складається з набору вузлів та відповідних зв’язків між ними. Кожен вузол являє собою паралельну комп’ютерну системою зі спільною або локальною пам’яттю, при цьому, наявність акселератора визначає тип пам’яті кожного вузла. Якщо акселератор присутній, вузол автоматично вважається системою з локальною пам’яттю, що складається з акселератора та центрального процесорного елементу, який разом з оперативною пам’яттю вузла утворює систему зі спільною пам’яттю. Такий вузол вважається також системою зі спільною пам’яттю у випадку відсутності акселератора. Отже, в кожному вузлі на найнижчому рівні організації присутня паралельна комп’ютерна система зі спільною пам’яттю.

У системі з вузлами, обладнаними акселераторами, можуть існувати такі типи зв'язків: на рівні розподіленої системи – будь-які доступні комунікативні канали в межах локальної та/або глобальної комп'ютерної мережі й на рівні вузла - інтерфейс взаємодії акселератора з материнською платою КС.

Головною метою системи є організація паралельного вирішення задач, тому необхідно розглянути різні способи її впровадження. Найбажанішим варіантом є створення моделі на найвищому рівні абстракції, де потрібно лише математичне представлення, а відповідальність за виділення потенційно паралельних секцій та їх виконання покладається на систему. Зупинимося на підході високого рівня абстракції, де вимагається математичне представлення вирішення задачі з мінімальним ручним виділенням секцій, які можна виконати паралельно. За основу взято схему

програмного інтерфейсу, запропоновану у [11], з деякими змінами. Модифікація полягає в розширенні спектру операцій, а саме: загальне розділення даних, загальний збір даних, загальне розділення задачі, часткове розділення даних, частковий збір даних, часткове розділення задачі. Перші три функції доступні користувачу, а останні три є службовими, виконання яких є важливим етапом програми, але вони приховані від користувача. Операції з даними доцільно модифікувати, використовуючи технології OpenMP та MPI для КС зі спільною та локальною пам'яттю відповідно. У зв'язку з особливостями роботи алгоритму оцінки задачі, з'являються можливості для автоматичної обробки доступності даних. Такий інтерфейс є досить ефективним для організації паралельного виконання більшості прикладних математичних задач. У дослідженні [3_2] наведені схеми роботи, які застосовуються до більшості сучасних задач паралельного програмування. Що підтверджується також тим, що 80% роботи програм [3_3] займає виконання циклічних та ітеративних конструкцій (рисунок 3.1).

Розглянутий інтерфейс не охоплює всі можливі типи завдань - наприклад, при роботі з розподіленою системою у режимі розподіленого сервера схема роботи буде значно відрізнятися. Тому раціонально залишити можливість використання низькорівневих примітивів.

В сучасних загальноприйнятих підходах до програмування вхідні паралельні задачі представляються відповідно до одного з двох наступних паттернів паралелізму: Data-driven parallelism та Task-driven parallelism.

Ідея паттерну "Data-driven parallelism" полягає в тому, що кожен обчислювач в системі виконує одну і ту ж послідовність операцій над своєю частиною вхідних даних, незалежно від інших, а отримані результати кожного обчислювача збираються та формуються у єдиний фінальний результат.

Ідея паттерну "Task-driven parallelism" полягає в протилежному – у системі між обчислювачами курсує один й той самий набір даних, та кожен

обчислювач виконує над ним свої набори операцій, а результат роботи фінального обчислювача й є остаточним результатом.

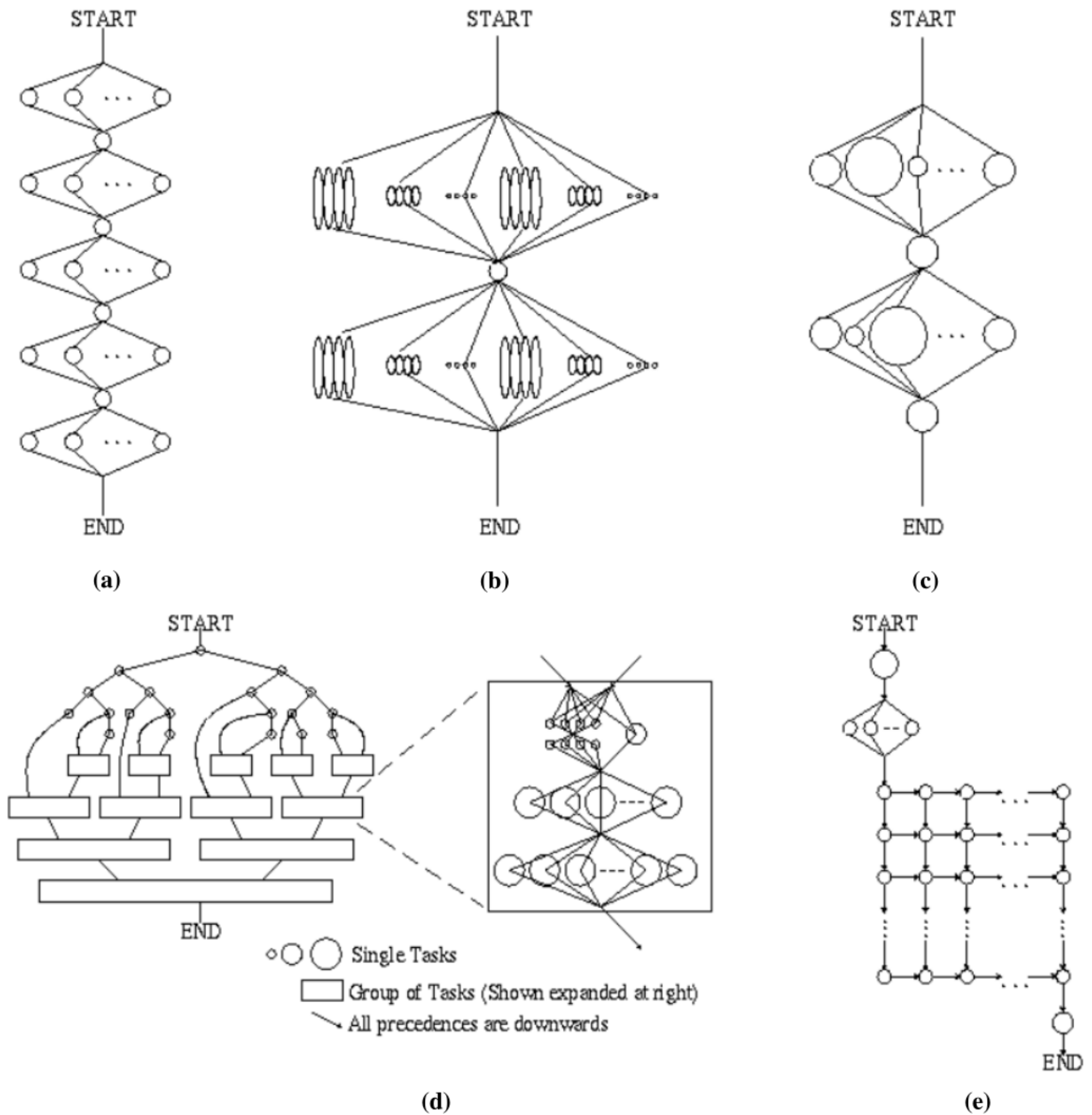


Рисунок 3.1 – Схематичні приклади схем роботи паралельних програм

Майже всі класифікації, наведені на рисунку 3.1, відносяться до "Data-driven parallelism" й відзначимо конструкцію, яка призводить до двох важливих понять – регулярного та нерегулярного паралелізму (рисунок 3.1, d). Регулярний паралелізм відноситься до випадку, коли кожен

потік виконує однаковий набір інструкцій. У випадку, коли деякі потоки виконують інший набір інструкцій, такий паралелізм вважається нерегулярним.

Отже, "Task-driven parallelism" завжди вважається нерегулярним, тоді як "Data-driven parallelism" може бути як регулярним, так і нерегулярним. Конкретно конструкція (рисунок 3.1, *e*) відповідає нерегулярному "Data-driven parallelism". Оскільки, як було зазначено, більшість паралельних математичних задач підпадають під схему "Data-driven parallelism", то основна увага в описаній системі надається саме регулярному та умовно-нерегулярному режимам (коли нерегулярність операцій є тенденцією, а не винятком (рисунок 3.1, *b,c*)).

3.1 Порядок оцінки вхідного завдання

З метою оптимізації та генерації коду необхідно мати абстрактне синтаксичне дерево вхідного завдання, яке відображає структуру програми та дозволяє виконувати оптимізації та генерацію вихідного коду.

Для прикладу, розглянемо наступний псевдокод, що демонструє обчислення алгоритму Евкліда для знаходження найбільшого спільного дільника двох цілих чисел (Листінг 3.1).

Листінг 3.1 – Алгоритм Евкліда для знаходження найбільшого спільного дільника двох цілих чисел

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

Синтаксичне абстрактне дерево коду матиме вигляд, наведений на рисунку 3.2.

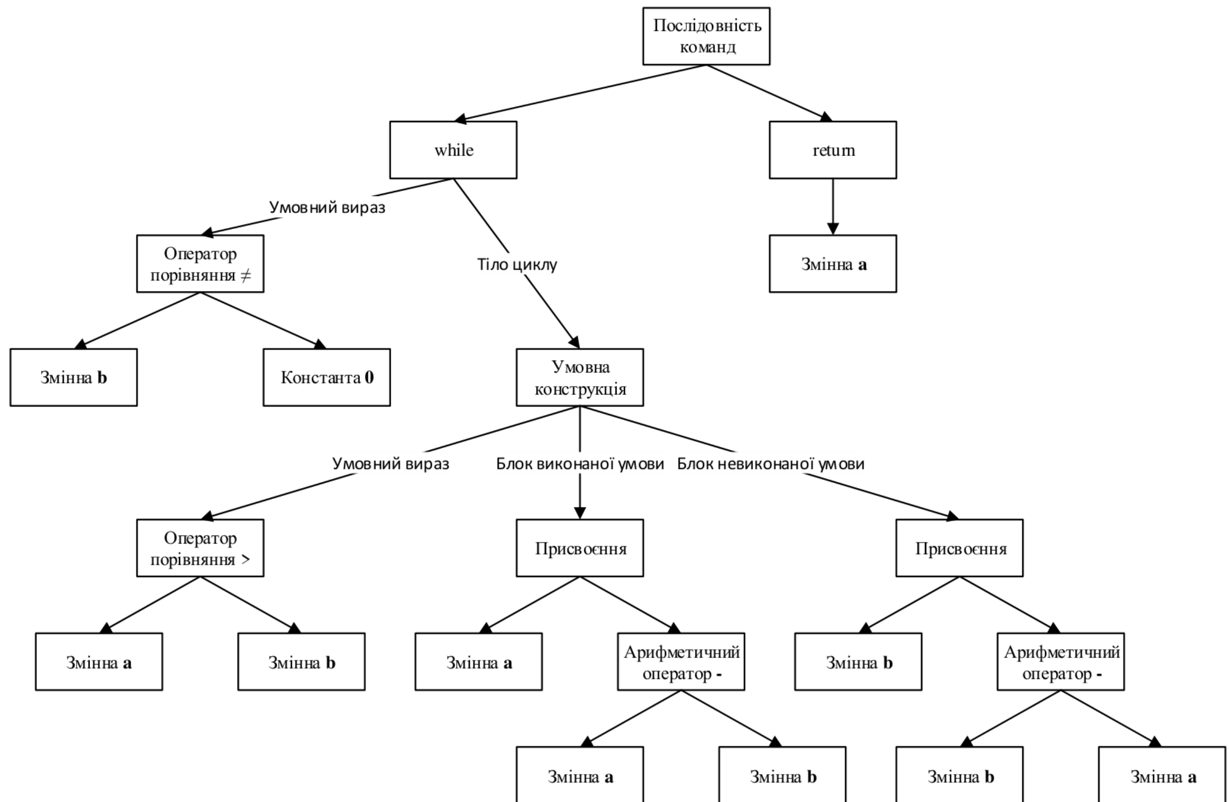


Рисунок 3.2 – Синтаксичне дерево для алгоритму Евкліда

Враховуючі "Data-driven parallelism", вважаємо, що кожен кінцевий потік буде виконувати однакові дії. Таким чином, в кожній КС виконання будь-якого алгоритму зводиться до виконання послідовності елементарних арифметичних операцій [3_4], можна провести орієнтовну оцінку об'ємності таких операцій, присутніх в задачі.

Для цього під час обходу графа задач методом обходу в глибину виявляємо наявність циклічних конструкцій та визначаємо їх потужність. У випадку, коли межі циклічної конструкції залежать від об'єму певного набору даних, можемо використати показник повного об'єму цих даних, оскільки на цьому етапі програми він вже буде відомий. Після цього ми отримуємо набір значень $I_1..I_N$, де I – потужність конкретного циклу, N – порядковий номер циклу. В подальшому знову ведеться пошук в глибину, але лише від вершин циклічних конструкцій, тобто проходяться цикли від 1 до N. Проводиться

підрахунок значень C_{SUM}, C_{SUB}, C_{MUL}, C_{DIV}, C_M – кількості елементарних операцій відповідно - додавання, віднімання, множення, ділення та присвоєння, що зустрілись в цьому циклі. В кінці кожного обходу кожне значення домножується на відповідний цьому циклу показник потужності I. Далі модифікуємо отримані значення відповідно до наявності позациклових елементарних операцій, збільшивши відповідні значення C_{SUM}, C_{SUB}, C_{MUL}, C_{DIV}, C_M на кількість виявлених відповідних операцій. Отримані в результаті фінальні значення показників можуть будуть використані в алгоритмах роботи інших складових.

За абстрактним синтаксичним деревом також можна встановити присутність спільних даних. Оскільки операції розділення даних та розділення задачі визначаються окремими функціями в коді, можна стверджувати, що всі дані, які одночасно містяться в дереві в тій гілці, що відповідає за задачу, яка підлягає розділенню, та гілках, що відповідають за попередні конструкції, вимагають копіювання. Ті самі дані, які, крім того, знаходяться ще в тих гілках, виконання яких передбачається наступним після розділення, потребують організації обробки з безпекою потоку.

3.2 Процедура ініціалізації КС

КС складається із набору розподілених вузлів, які мають між собою слабкі зв'язки, при цьому передбачена початкова рівноправність всіх вузлів, що означає, що будь-який вузол може взяти на себе роль ініціатора обчислень. Операція ініціалізації системи розпочинається із запуску цієї ініціалізації вузлом, який розпочинає обчислення. З урахуванням того, що система може змінюватися шляхом додавання або вилучення вузлів, жоден вузол на початку не має повної інформації про всю систему. Перед початком обчислень вузол має лише дані про ті вузли, яким він може делегувати обчислення. Ці дані, що знаходяться в конфігураційному файлі кожного вузла, представлені набором значень Z_i.

Згідно із цими відомостями відбувається логістична перебудова системи з метою створення орієнтованого ациклічного граф-дерева. У цьому графі кореневим елементом є вершина-ініціатор. На першому рівні розташовані вершини, яким безпосередньо може делегувати обчислення ініціатор. На наступному рівні розташовані вершини, якими можуть делегувати обчислення вершини першого рівня, і так далі. Акселератори розглядаються як індивідуальні вузли, з'єднані лише із вузлами, в яких вони встановлені. На рисунку 3.3 подано схематичний вигляд такого графу для системи з 10 різноманітними вузлами.

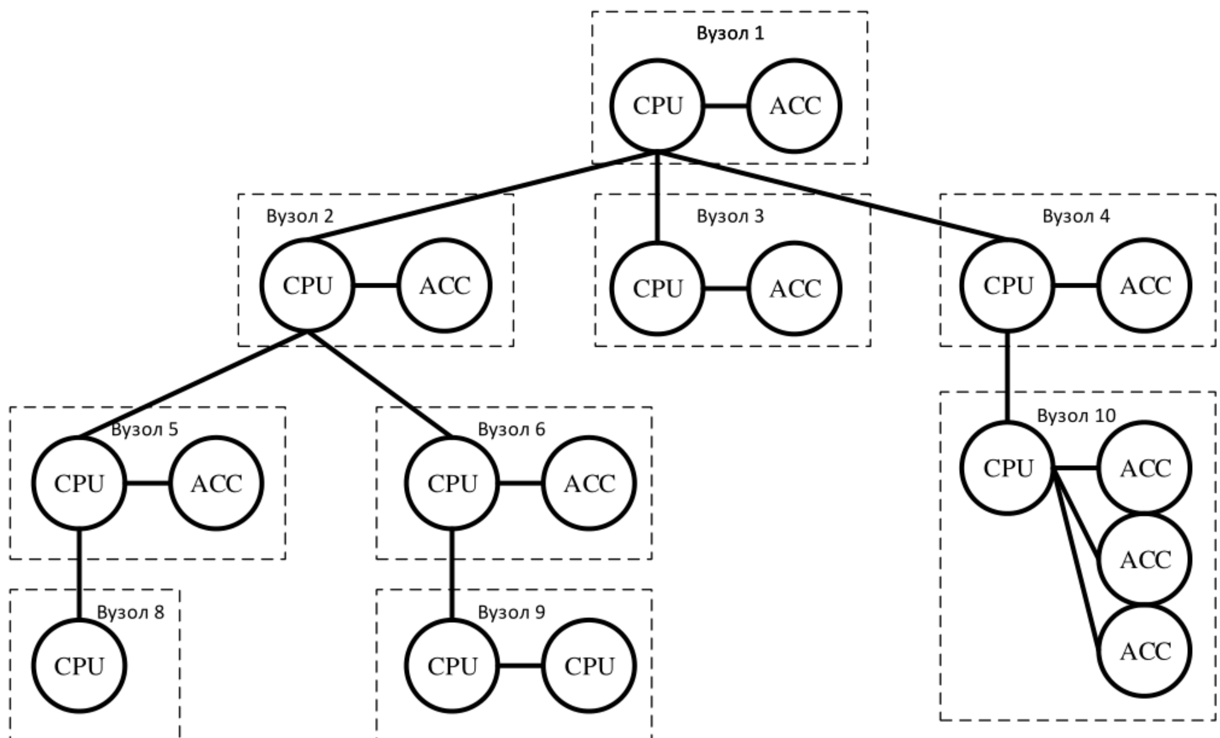


Рисунок 3.3 – Схематичний вигляд графу для КСз 10 різноманітних вузлів

Далі необхідно оцінити граф-дерево з відносними показниками продуктивності для кожного вузла та його компонентів.

3.3 Оцінка графу системи

Для вагомості ребер спочатку потрібно оцінити абсолютну пропускну здатність каналів зв'язку між вузлами. Якщо на вузлі є акселератори, то слід врахувати інтерфейси зв'язку між процесорним елементом вузла та акселераторами. Для абсолютної оцінки пропускної здатності інтерфейсів зв'язку з акселераторами, достатньо провести тестові передачі та отримання пакетів даних від них, заміряючи відповідні показники часу. Тобто, пропускна здатність інтерфейсу WI визначатиметься за формулою (3.1):

$$W_I = \frac{2 * S}{\Delta T_{SEND} + \Delta T_{RECV}} \ , \tag{3.1}$$

де ΔT_{SEND} – різниця між початком й завершенням процесу відправки даних, ΔT_{RECV} – різниця між початком й завершенням процесу отримання даних, S – обсяг відправлених даних.

Для оцінки абсолютної пропускної здатності каналів зв'язку між вузлами можна використовувати утиліту traceroute. Дані, отримані за допомогою цієї програми, менше піддаються впливу відсутності гарантій щодо єдиного маршруту проходження пакету, ніж ті, що отримані за допомогою програми ping. Пропускна здатність комунікаційного каналу між двома вузлами WC буде визначатися за наступною формулою (3.2):

$$W_C = \sum_{i=1}^{N} \left(\frac{S_i}{\sum_{j=1}^{M} Tj} \right) , \tag{3.2}$$

де N – кількість запусків 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 запуск пакету.

Отримані абсолютні показники пропускної здатності каналів зв'язку та інтерфейсів слід окремо перевести у відносні, так само як і показники продуктивності елементів. Однак в цьому випадку вагування буде проводитися не відносно сумарного показника, а відносно найкращого показника. Тобто, відносний показник пропускної здатності каналу WCR буде визначатися за формулою (3.2)

$$W_{CR} = \frac{W_C * 100}{W_{CB}} \ , \tag{3.3}$$

де W_C – абсолютний показник пропускної здатності каналу, W_{CB} – абсолютний показник найшвидшого із каналів в системі.

Відповідно, відносний показник пропускної здатності інтерфейсу взаємодії з акселератором визначатиметься за формулою (3.4:

$$W_{IR} = \frac{W_I * 100}{W_{IB}} \ , \tag{3.4}$$

де W_I – абсолютний показник пропускної здатності інтерфейсу, W_{IB} – абсолютний показник найшвидшого із інтерфейсів в системі.

В результаті отримані відносні показники продуктивності каналів та інтерфейсів і будуть відповідати вагам відповідних ребер графу системи.

Для вагомості вершин спершу слід отримати абсолютні показники продуктивності елементів вузлів. З початку необхідно отримати набір показників T_{SUM}, T_{SUB}, T_{MUL}, T_{DIV}, T_M – середній час виконання кожним елементом операцій додавання, віднімання, множення, ділення і присвоєння відповідно. Для цього необхідно на кожному елементі, на центральних процесорних елементах й на акселераторах, виконати заздалегідь задану

кількість кожної з цих операцій над завчасно згенерованими випадковими різнотипними даними й виміряти час, який витрачено на кожну операцію.

Після отримання середніх часових показників за формулою (3.5) можна обчислити абсолютний показник продуктивності елементу вузла системи:

$$W_{EA} = \frac{P * \left(\frac{C_{SUM}}{T_{SUM}} + \frac{C_{SUB}}{T_{SUB}} + \frac{C_{MUL}}{T_{MUL}} + \frac{C_{DIV}}{T_{DIV}} + \frac{C_{M}}{T_{M}}\right)}{\Delta L * \Delta t * \Delta M}, \quad (3.5)$$

де P – кількість потоків 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.

В результаті отримаємо відносні показники продуктивності елементів системи, які будуть відповідати вагам відповідних вершин графу системи. Таким чином, на цьому етапі ми отримаємо логістичну карту системи для конкретного обчислення. Фінальний приклад, з урахуванням ваги вершин, до якої включено вагу ребер, такої карти для схеми зі структурою, що відповідає схемі на рисунку 3.3, показано на рисунку 3.4.

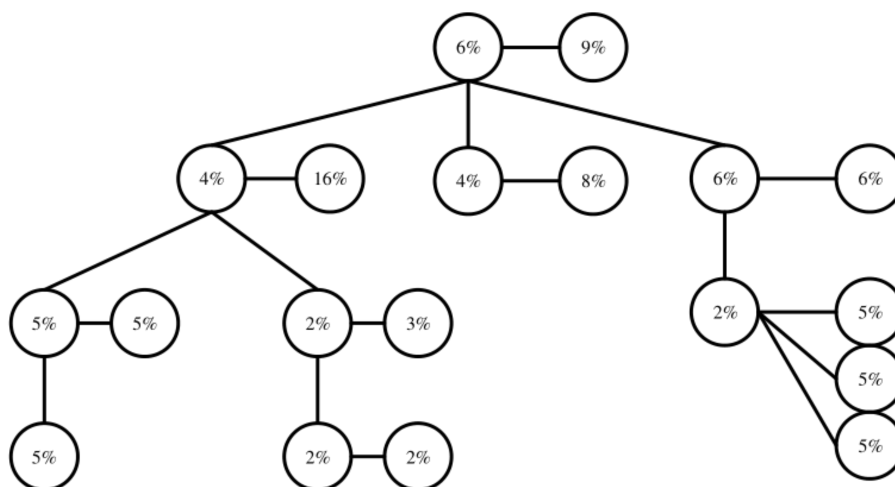


Рисунок 3.4 – Схема графу з урахуванням ваги ребер

Після завершення цього етапу КС готова до виконання безпосередніх обчислень й з цього моменту балансування навантаження на різних рівнях КС будуть повністю контролювати та регулювати ці обчислення.

3.4 Алгоритм підвищення ефективності КС

В сучасних підходах до організації комп'ютерних систем можна запропонувати один з можливих засобів підвищення їх ефективності -це використання балансування навантаження.

Балансувальник, який реалізує статичне балансування, виконує свою основну роботу перед безпосереднім запуском паралельної програми. Основуючись на інформації про систему (кількість елементів, їх продуктивність, зв'язки між елементами, пропускна здатність каналів) та данних про задачу (кількість підзадач, наявність зв'язків між ними, об'єм та інтенсивність пересилок між підзадачами, наявність спільних ресурсів), балансувальник виконує пошук оптимального розподілу підзадач між обчислювальними елементами, використовуючи певний критерій ефективності (зазвичай мінімізація часу роботи або максимізація коефіцієнта ефективності). Після цього роль балансувальника зводиться до почергового запуску задач на виконання, відповідно до розробленого плану, без змін у процесі виконання.

Задача балансування є NP-повною, що означає, що отримане балансувальником рішення є лише наближенням до ідеального. Це наближення зазвичай вимагає великої кількості часу для його пошуку за складними алгоритмами, що може значно збільшити сумарний час роботи системи, що є значним недоліком. Також, статичне балансування має недоліки у врахуванні поточних показників стану вузлів та стану обчислень під час їх виконання. Рішення, отримане статичним балансувальником, може бути нівельовано у випадку, якщо один із вузлів системи від'єднається під

час обчислень, що призведе до необхідності повторного ребалансування завдань, які залишились невиконаними.

Незважаючи на ці недоліки, статичний балансувальник має свої переваги. Відсутність моніторингу системи під час обчислень є перевагою, оскільки система не витрачає ресурси на моніторинг та динамічне ребалансування. Це дозволяє системі використовувати всі доступні ресурси лише для виконання обчислень.

На відміну від статичного балансувальника, динамічний балансувальник майже не витрачає часу на підготовку попереднього оптимального рішення і відразу приступає до призначення підзадач обчислювальним вузлам та організації передач. Під час його роботи постійно ведеться моніторинг показників стану системи та черг підзадач, на основі яких приймаються рішення щодо розподілу наступних підзадач між вузлами.

Серед переваг динамічного балансувальника можна відзначити його високу гнучкість й здатність максимально задіяти всі наявні ресурси КС. Рішення, знайдені динамічним балансувальником, виявляються більш якісними порівняно із тими, що були знайдені статичним балансувальником. При динамічному балансуванні зростає стресостійкість системи, а витрати на обробку стресових ситуацій зменшуються, оскільки в разі втрати зв’язку з одним із вузлів, балансувальник просто додасть в чергу підзадачі, виконання яких було призначено втраченому вузлу. Недоліком є те, що система повинна виділяти певну частину ресурсів, (обчислювальні та комунікативні), для роботи процесу моніторингу вузлів. Крім того, для ефективного вирішення задачі балансування, з урахуванням апаратного опису, необхідно внести модифікації в схему організації роботи КС. Пропонується вилучити абстрактний рівень КС й ввести новий абстрактний рівень – рівень паралельної системи, а також рівень планувальника паралельної системи. Ця модифікація суттєво розширює можливості схеми та полегшує подальший процес балансування (рисунок 3.5).

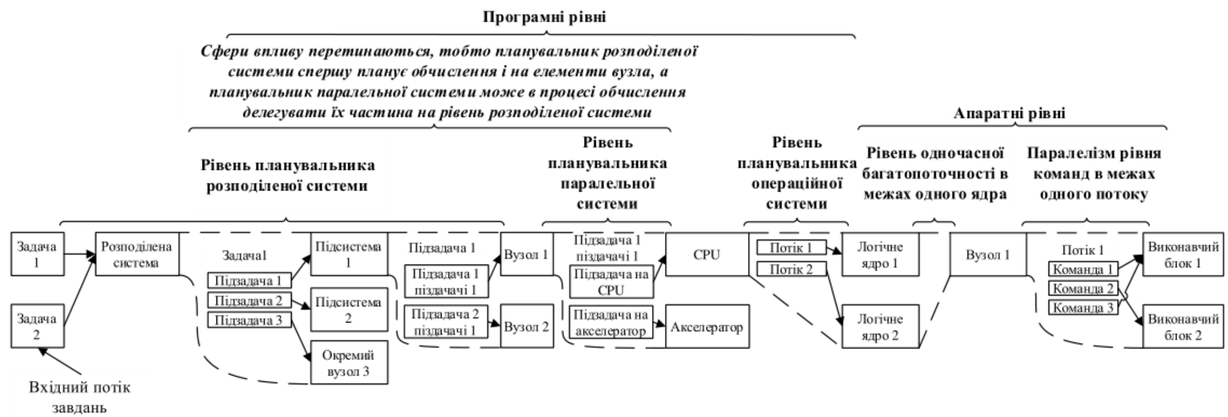


Рисунок 3.5 – Схема процесу обробки задач в КС

Статичне балансування має свої недоліки, такі як:

- наближеність отриманого рішення – отримане статичне балансування є наближеним до ідеального рішення, оскільки воно базується на передбаченнях та апроксимаціях обсягу даних і продуктивності вузлів;

- додаткові витрати часу на етапі підготовки – використання статичного балансування перед запуском програми вимагає часу на аналіз системних параметрів та розподіл завдань. Ці додаткові витрати можуть бути значними;

- низька стресостійкість – статичне балансування може бути менш стресостійким, оскільки не враховує змін у стані системи та вузлів під час виконання обчислень.

Враховуючи ці недоліки, пропонується застосовувати початкове статичне балансування на рівні розподіленої КС, у який відсоток використання вузла залежить від об'єму даних та продуктивності вузла на рівні системи. Це забезпечить ефективне попереднє наближення рішення задачі балансування протягом короткого проміжку часу

Згідно з описом функцій для розпаралелювання програми, перші три функції (загальне розділення даних, загальний збір даних й загальне розділення задачі) – взаємопов'язані. На вузлі-ініціаторі обчислень спочатку виконується функція загального розділення даних. Ця функція приймає дані,

які будуть розділені між вузлами, й виконує поділ відмічених даних, які можуть бути векторною структурою в пам'яті, такою як масиви, матриці, списки, символи рядків, рядки документів, пікселі зображення тощо. Якщо тип даних неподільний, то ці дані просто копіюються.

В той самий час, в системі, залежно від кількості вузлів, формується набір динамічних об'єктів, клас яких може додатково (хоча це не є обов'язковим) успадковувати від класу паралельної задачі або потоку. Ці об'єкти в основному мають ідентифікатори (які відповідають ідентифікаторам вузлів у системі), а також динамічні поля, куди можна додавати дані, які слід надіслати вузлам.

Далі в системі створюється набір динамічних об'єктів, клас яких може також, але не обов'язково, наслідувати клас паралельної задачі або потоку, залежно від кількості вузлів. Ці об'єкти обладнані полями-ідентифікаторами, які відповідають ідентифікаторам вузлів в системі, а також динамічними полями для даних, які необхідно передати вузлам.

Останньою універсальною функцією є "збір даних", ця функція визначає, які саме дані будуть збиратися на вершині-ініціаторі після виконання обчислень, а також, за потреби, описує варіації механізму збору цих даних. У разі наявності aeyrws] він автоматично викликається після закінчення роботи потокової та акселераторної функцій у динамічному об'єкті, а відповідні функції створюються в динамічних об'єктах. Після виконання цього кроку система готова до розпочатку обчислень.

Динамічне балансування надає можливість здійснювати наступні функції:

- компенсація неефективності статичного балансувальника – під час роботи системи динамічне балансування може безпосередньо коригувати відсоток ефективності, який може бути втрачений при використанні статичного балансувальника для розподілу навантаження в системі;

- збільшення стресостійкості системи – динамічне балансування сприяє високій стресостійкості системи, зменшуючи втрати в разі обриву

зв'язку навіть у випадку втрати зв'язку з усіма вузлами;

- впровадження технології відкладених обчислень – динамічне балансування робить використання технології відкладених обчислень більш простим, враховуючи потреби системи на етапі роботи, що є значно простішим у порівнянні з плануванням на етапі статичного балансування.

Динамічний балансувальник відповідає за ініціацію та проведення обчислень, ведучи свою роботу на кінцевих вузлах, однак в системі існує механізм централізації, що передбачає можливість часткового переходу на більш високий рівень абстракції, а саме на рівень розподіленої системи, у деяких режимах роботи.

На початковому етапі використання технології відкладених обчислень, динамічний балансувальник кожного вузла ініціює відправку запиту до вузла-ініціатора обчислень, в якому він висловлює бажання отримати відповідний динамічний об'єкт від конкретного вузла-автора запиту. Цей запит може бути направлений безпосередньо або через проміжні вузли. Отримані запити надсилаються на вузол-ініціатор та зберігаються у формі асинхронної черги колбеків. Оскільки дані, що передаються вузлам, вже розподілені так, щоб збалансувати об'єм пакету даних для конкретного вузла з пропускною здатністю каналу зв'язку з цим вузлом, порядок видавання даних не має значення. Коли динамічний об'єкт повністю готовий для відправлення на відповідний вузол, відповідний запит вибирається з асинхронної черги, і сереалізований динамічний об'єкт передається до цільового вузла.

При отриманні вузлом-реципієнтом призначеного динамічного об'єкту відбувається його десереалізація та подальший розбір на менші підзадачі. Механізм цього розбиття в цілому аналогічний тому, що використовується на етапі статичного балансування, але при цьому проміжки даних рівні між собою. Задача розбиття полягає в тому, щоб створити кілька менших об'єктів, кількість яких заведомо перевищує кількість ядер або потоків у центральному процесорному елементі вузла.

При наявності акселератора вузол приймає два динамічних об'єкти: один призначений для обчислень на центральному процесорному елементі, а інший для обчислень на акселераторі. Механізм обробки об'єкту, призначеного для центрального процесорного елементу, описаний раніше.

Механізм обробки об'єкту, призначеного для акселератора, включає поділ за аналогічною схемою, але на цей раз на три об'єкти. В системі виділяються два потоки, які відповідальні за взаємодію з акселератором [13]. Це обумовлено використанням інтерфейсу PCI Express, що працює в послідовному повнодуплексному режимі. Таким чином, два потоки необхідні для ситуації, коли один потік відправляє дані, а інший в той самий час приймає результати.

Також на кожному вузлі існує окремий потік-монітор, призначений для виконання операцій моніторингу стану паралельної системи та її складових. Цей потік виходить з блокування з певною фіксованою частотою й проводить огляд показників елементів системи, таких як різниця системного показника завантаженості між двома останніми вимірюваннями, різниця тактової частоти між двома останніми вимірюваннями, температурна різниця між двома останніми вимірюваннями та стан черг обробки потокових та акселераторних функцій. Отримані дані цей монітор передає динамічному планувальнику, і відповідний потік-монітор блокується до наступного вимірювання. Динамічний планувальник, згідно з інформацією від монітору, приймає рішення щодо подальшої обробки враховуючи такі аспекти [14]:

Якщо поточний рівень завантаження елементу не досягає бажаного показника, планувальник відправляє для виконання відповідну кількість динамічних об'єктів із черги для цього елементу. Це може включати один або кілька об'єктів, залежно від різниці між поточним і бажаним рівнем завантаження. Якщо є заблоковані об'єкти, які вже проходили обробку, їх обробка відновлюється в першу чергу. У випадку, коли черга пуста, вузол позначає готовність для прийому додаткових обчислень.

Якщо поточний рівень завантаження елементу перевищує бажаний показник, то виконання одного або декількох динамічних об'єктів з тих, що наразі обробляються, блокується. Після цього відбувається ребалансування між вузлами системи.

Монітор може бути активований й в позачерговому режимі у випадку отримання запиту для уточнення показників. Динамічне балансування також проводиться частково, надаючи вищий пріоритет підзадачам, які вже присутні на конкретному вузлі. Крім того, виконується балансування між вузлами на рівні розподіленої системи, використовуючи механізм ребалансування. Цей механізм передбачає делегування обчислень від одного вузла до іншого. Передача процесів в рамках КС є складною задачею - складність виникає через необхідність передачі не лише самого процесу, але й всього його контексту, включаючи стан стеку, дані та адаптацію до різноманітності архітектур вузлів, яка особливо актуальна для акселераторів. Таким чином, пропонується механізм, який не передає частину вже виконуваного процесу, а акумулює частину обчислень, готових до виконання, і передає їх для подальшої обробки. В описаному алгоритмі роботи потоку-монітору вузла передбачено сценарій, коли вузол входить в режим готовності прийняти додаткові обчислення. Оскільки в системі існує часткова, а не повна зв'язність, сигнал про готовність вузла надсилається вузлу-ініціатору. Ці сигнали збираються в один список та сортуються за продуктивністю вершин, від найпотужнішої до найменш потужної. Вузол-ініціатор також отримує сигнали про перевантаженість вузла, що означає запит від інших вузлів для прийняття частини їх обчислень. При отриманні запиту на делегування обчислень іншому вузлу, вершина-ініціатор перевіряє стан списку готових вузлів та обирає найпотужніший вузол. Потім відправляється уточнювальний сигнал, що ініціює позачерговий моніторинг даного вузла для перевірки поточних показників. Якщо перевірка підтверджує готовність вузла прийняти обчислення, викликається механізм делегування обчислень між вузлами. У випадку, коли перевірка не

підтверджує готовність вузла, сигнал від цього вузла видаляється зі списку сигналів, і перевіряється наступний вузол за списком. Також можна проводити уточнення актуальності делегування від вузла, від якого надходить запит.

3.5 Процес делегування обрахунків між вузлами КС

Описаний механізм делегування обчислень використовує наступн три функції для розпаралелювання програми: часткове розділення даних, частковий збір даних та часткове розділення задачі. Ці функції подібні до відповідних функцій статичного планувальника, проте розбиття відбувається на два рівномірних проміжки даних, що породжує два нових динамічних об'єкти. Один з об'єктів залишається в черзі вузла, на якому відбувається розподілення, а інший відправляється вузлу-ініціатору. При отриманні цього об'єкта вузол-ініціатор передає його вузлу, готовому прийняти обчислення, і об'єкт одразу запускається на обробку. Такий механізм працює як з об'єктами центральних процесорних елементів так і з об'єктами акселераторів. Функція збору даних включає асинхронний збір даних за запитом через механізм коллбеків. При обробці об'єктів, призначених для вузла з самого початку, збір даних спочатку відбувається на цьому вузлі, а потім передається вузлу-ініціатору. При обробці делегованих об'єктів результати обчислень передаються по тому ж маршруту назад до вузла, на якому планувалась обробка цього об'єкту [13].

Виконання описаного механізму динамічного балансування за такою схемою також забезпечує автоматичну регуляцію паралелізму, а користувач може розподілити код між вузлами системи в рамках коду для досягнення додаткового рівня паралелізму й, відповідно, підвищення ефективності роботи КС.

4 МОДЕЛЮВАННЯ РОБОТИ РОЗПОДІЛЕНОЇ ПАРАЛЕЛЬНОЇ КОМП’ЮТЕРНОЇ СИСТЕМИ

4.1 Вибір прототипів компонентів КС

Запропонований метод передбачає створення рівноправної КС, де програмне забезпечення кожного вузла може працювати як в режимі серверу (вузол-ініціатор обчислень), так й в режимі клієнту (вузол-приймач обчислень). Що дозволяє КС бути більш гнучкою та адаптивною до різноманітних завдань. Однак, в архітектурі сучасних комп'ютерних систем виявляється недолік – сервіс-орієнтованість програмного забезпечення. Це вказує на важливість пошуку шляхів інтеграції та спрощення взаємодії між різними компонентами системи. Основними проблемами такого підходу є необхідність установлення окремих програмних комплексів та реалізація зв'язків між ними користувачем, що може призводити до помилок та невідповідностей.

Для вирішення цих проблем необхідно розглядати процес створення, яка об'єднає функції обчислень, керування, балансування навантаження та планування обчислень. Це може полегшити конфігурацію та взаємодію між різними частинами системи, забезпечуючи однорідний та інтуїтивно зрозумілий інтерфейс користувача.

Запропоновано зберегти сервісно-компонентну архітектуру всередині програмного комплексу, але представляти її для користувача як єдиний повнозв’язний програмний продукт. Кожен компонент в цій архітектурі виконує конкретні функції, спрощуючи управління та роботу розподіленої системи. Основні компоненти цієї архітектури включають: ініціалізатор системи, тестувальник потужності, аналізатор прикладного коду, балансувальник навантаження та Монітор стану системи.

Ініціалізатор системи відповідає за запуск та ініціалізацію всіх компонентів системи при її старті. Може також відігравати роль ініціатора обчислень.

Тестувальник потужності здійснює вимірювання та тестування потужності вузлів системи, щоб оптимізувати розподіл завдань.

Аналізатор прикладного коду відповідає за аналіз та оцінку коду, який буде паралельно виконуватися, щоб оптимізувати розподіл завдань та використання ресурсів.

Балансувальник навантаження використовує отриману інформацію від тестувальника потужності та аналізатора прикладного коду для розподілу обчислювального навантаження між вузлами системи.

Монітор стану системи відстежує стан всіх елементів системи, збирає дані щодо завантаженості, температури, та інших параметрів для надання цільовим компонентам інформації для оптимального управління.

Така архітектура дозволяє створити єдину платформу, яка забезпечить управління та використання ресурсів КС, а також дозволить користувачам сприймати її як єдиноцінний програмний продукт, забезпечуючи при цьому внутрішню гнучкість та ефективність. Забезпечення відповідності між інтерфейсами взаємодії різних компонентів системи та задекларованими зв'язками між етапами функціонування системи є критичним для правильної роботи системи.

4.2 Вибір засобів програмної реалізації КС

Основні засоби програмування для комп'ютерних систем - MPI та Intel TBB мають деякі недоліки. Перш за все, ці технології орієнтовані на гомогенні повнозв'язні системи, що може становити проблему в гетерогенних середовищах. Крім того, відсутність вбудованої підтримки акселераторів на рівні планування та балансування обчислень ускладнює їхнє ефективне використання в системах, що використовують акселератори [15].

Незважаючи на високу швидкодію програм, створених за допомогою MPI та Intel TBB, та загальну зручність їх використання, основною їх проблемою залишається застарілість орієнтації. Ця проблема стає ще більш актуальною з року в рік. Однією з ключових причин цього є обмеженість цільових мов. OpenMP та MPI були розроблені перш за все для мов Fortran та C, а Intel TBB – для C++. Зараз мова C перейшла в основний чинник системного програмування та підтримки існуючого коду.

Орієнтація OpenMP та MPI на процедурне програмування стає проблемою у зв'язку зі значними змінами у парадигмах програмування. У сучасних умовах вже визнані складніші парадигми, такі як об'єктно-орієнтована, функціональна та реактивна. Хоча розробка продовжується, обмеженість цих технологій у підтримці сучасних підходів ускладнює їх використання. Щодо Intel TBB, його орієнтація на C++ також має свої недоліки. Навіть якщо C++ залишається потужною та високопродуктивною мовою, нові мови програмування наздоганяють її в питаннях продуктивності.

Низький рівень абстракції у використанні MPI призводить до необхідності виділення значного проміжного рівня в програмному комплексі для організації взаємодії між шарами бізнес-логіки та комунікації. Виникає необхідність ручного вирішення проблем, що вимагає високої кваліфікації розробника та створює значний простір для потенційних помилок.

Основні вимоги до мови програмування включають підтримку функціонального програмування, зокрема, можливість передавати функції як аргументи іншим функціям, а також можливість створення динамічних об’єктів в межах мови реалізації. Необов'язково, але бажаною є підтримка засобів для організації обчислень на акселераторах, якщо такі передбачаються в системі. Наприклад, для графічних процесорів очікується наявність реалізації технологій, таких як OpenCL або CUDA, у вигляді бібліотек, які сумісні з обраною мовою програмування.

Таким чином, здійснюється реалізація запропонованого методу з використанням мови програмування C# та технологій, таких як Windows

Communication Foundation (WCF) та Thread Parallel Library (TPL) в рамках фреймворку .NET компанії Microsoft. Оскільки в цільових тестових системах передбачається використання акселераторів у вигляді графічних процесорів, то для їх підтримки використовується бібліотека OpenCL.

4.3 Вибір програмних комплексів та КС

Для оцінки ефективності запропонованого методу, необхідно реалізувати його у вигляді моделей програм для паралельних обчислень в розподілених комп'ютерних системах з акселераторами. В рамках яких слід вирішити різноманітні математичні та прикладні задачі з різними вхідними параметрами, що дозволить виявити необхідні залежності та провести оцінку ефективності. Крім того, необхідно провести порівняння запропонованого методу з контрольними зразками, в рамках яких також необхідно вирішувати відповідні різноманітні математичні та прикладні задачі з різними вхідними параметрами, але без застосування запропонованого методу. Різниця у показниках швидкодії, прискорення та ефективності між цими контрольними програмними прототипами та програмами, й покаже на реальну ефективність запропонованого рішення.

Критичним впливом на швидкодію, коефіцієнти прискорення та ефективності паралельних програм, реалізованих за моделями зі спільною пам'яттю, є пересилки даних у системах з локальною пам'яттю. Крім того, обмін даними між вузлами через глобальну мережу не завжди може виконуватись за однакові проміжки часу. З метою виявлення середніх діапазонів відхилення латентності виконуються заміри часу на пересилки даних, рахуючи різницю часу відправлення та часу прийому.

З метою мінімізації впливу заміру часу на роботу програми, виконання всіх вимірів на всіх вузлах відбувається в паралельних асинхронних функціях. Обмін результатами та їх співставлення відбуваються після закінчення безпосередніх обчислень.

Реалізовано чотири програмні модельні комплекси для паралельних обчислень в розподілених КС з акселераторами. Вони використовують різні сучасні засоби, а також запропонований підхід. Характеристики цих програмних комплексів наведено в таблиці 4.1.

Таблиця 4.1 – Характеристики програмних комплексів КС

№ п. п.	Мова реалізації	Технологія організації паралелізму на рівні розподіленої КС	Технологія реалізації паралелізму на рівні паралельних КС	Технологія реалізації обчислень	Підхід до балансування навантаження	Призначення
1	C#	WCF	TPL	Open_GL	Новий метод	Тест
2	C#	WCF	TPL	Open_GL	Новий метод	Тест
3	C++	MPI	OpenMP	Open_GL	Існуючий метод	Контроль
4	C++	MPI	Intel TBB	Intel TBB	Існуючий метод	Контроль

Тестові системи можуть розгорнуті за допомогою хмарного сервісу у якому є можливість розгортати різні екземпляри віртуальних машин, включаючи ті, що мають графічні процесори (акселератори) як у рамках одного хабу так й у різних хабах.

Параметри тестових та контрольних КС наступні;

- 1 вузол: 4xCPU Intel SkyLake, GPU Nvidia Tesla K80, RAM 15 Gb;
- 2 вузол: 8xCPU Intel SkyLake, GPU Nvidia Tesla P100, RAM 30 Gb;
- 3 вузол: 16xCPU Intel SkyLake, GPU 2xNvidia Tesla K80, RAM 15 Gb;
- 4 вузол: 32xCPU Intel SkyLake, GPU - відсутня, RAM 15 Gb.

На всіх вузлах створених розподілених гетерогенних комп'ютерних систем встановлено ідентичне програмне забезпечення: Windows Server 2019 Datacenter x64 , .NET Framework 4.7, компілятор C++ Microsoft Visual C++, OpenCL 2.2, OpenMP 3.0, Microsoft MPI 9.0.1.

Карта розміщення та зв'язки між вузлами представлені на рисунку 4.1.

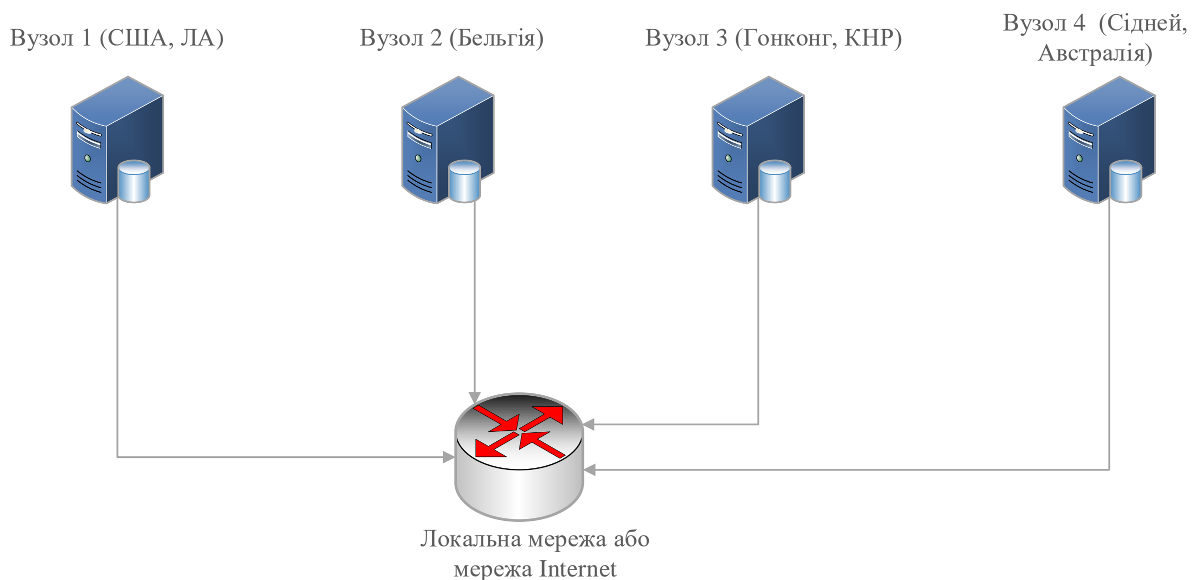


Рисунок 4.1 – Схема розміщення вузлів систем

Враховуючи топологію на рисунку 3.1, в усіх тестових випадках вузлом-ініціатором обчислень вважатимемо вузол 2 (Бельгія). Важливо врахувати, що він не має можливості передавати дані безпосередньо до вузла 4 (Сідней). Таким чином, вся комунікація між вузлами 2 та 4 буде відбуватися через вузол 3 (Гонконг).

4.4 Моделювання роботи КС

Процес моделювання включає багаторазовий почерговий запуск всіх модельних програмних прототипів із замірами їх часу виконання. Середній час виконання кожної програми визначається як середнє арифметичне всіх часових результатів її роботи. Також виконувалися короткі асинхронні паралельні заміри часу початку та кінця пересилок для визначення їх впливу. Здійснювався контроль за результуючими середніми показниками завантаження акселератора (графічного процесору) кожного з вузлів задачами для оцінки обсягу обчислень, який система визначає як доцільний для виконання на акселераторі. Часові показники результатів прямої швидкодії програми були переведені в абсолютний коефіцієнт прискорення паралельної програми. Цей коефіцієнт визначається як відношення часу виконання послідовного варіанту програми до часу виконання паралельної програми на P обчислювачах і вказує як скорочується час виконання програми в паралельній КС [11].

Розраховується коефіцієнт прискорення k_{SU} за формулою (4,1):

$$k_{su} = \frac{T_1}{T_P}\ , \tag{4.1}$$

де T_1 – час роботи послідовної частини, T_P – час роботи паралельої частини програми на P-вузлах.

Під час тестування проводилася перевірка залежності показників коефіцієнтів прискорення різних програм в залежності від обсягу задач. Цей обсяг визначався за певними вхідними даними чи умовами. Зазначте, що при дуже великих обсягах послідовних варіантів задач, де обчислення не можливо було виконати за адекватний час, час роботи послідовного варіанту обчислювався як передбачення значення функції на основі логістичного порівняння.

Коефіцієнт ефективності k_{EF} є важливим показником ефективності роботи паралельних систем та алгоритмів, він обчислюється за формулою (4.2) на основі коефіцієнту прискорення та вказує на ефективну сумарну загрузку всієї системи [15].

$$k_{EF} = \frac{T_1}{T_P * P} * 100\% \, , \tag{4.2}$$

де T_1 – час роботи послідовного варіанту алгоритму, T_P – час роботи паралелього варіанту алгоритму на P- вузлах.

Розглядаючи особливості паралельних розподілених КС з акселераторами, доцільно використовувати коефіцієнт завантаженості акселератору. Цей показник вказує на відсоток обчислень, що виконуються акселераторами в системі в порівнянні з загальним обчисленням.

Розраховуючи показники для розподіленої системи, необхідно врахувати розкид латентності, або відсоткові відхилення тривалості передачі даних між однаковими обчислювачами. Для цього розміщують часові заміри у відповідних частинах програми, й використовувати етапи, які можна вважати аналогічними, для оцінювання часових витрат на передачу даних.

Тестування пропонується провести на двох найпоширених прикладних задачах, кожна з яких має свою цінність, оскільки відповідає поширеному випадку в практичній організації паралельних програм в розподілених КС.

4.4.1 Моделювання обчислення наближення ряда

Формула для обчислення гіперболічного арктангенсу на проміжку від -1 до 1 виглядає наступним чином:

$$arth(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots = \sum_{n=0}^{\infty} \frac{1}{2n+1} x^{2n+1} \quad |x| < 1 \tag{4.3}$$

Вона дозволяє КС проводити вирахування значень гіперболічного арктангенсу з заданою точністю за допомогою розкладу в ряд та наближення цього ряду. У виразі, x представляє значення гіперболічного арктангенсу, та обчислення проводиться за допомогою натурального логарифму та арифметичних операцій.

У паралельному обчисленні даного ряду, де кожен член є незалежним, можна використовувати розподілений підхід для підрахунку частин ряду на різних обчислювачах. Однак, при введенні умови досягнення заданої точності, виникає проблема необхідності постійної комунікації між обчислювачами.

Ініціатор обчислень відповідає за наступні завдання:

- зчитування вхідних даних, значення x та показника точності у форматі дробового числа;
- передача значення x іншим вузлам для обчислення частини ряду;
- надання іншим вузлам інформації про позиції членів ряду, які вони повинні обчислити;
- збір результатів обчислень від усіх вузлів;
- повідомлення інших вузлів про продовження чи завершення виконання обчислень.

Це вимагає ефективного механізму синхронізації та обміну даними між вузлами для забезпечення виконання обчислень й досягнення необхідної точності. Таким чином, ця задача є актуальною для тестування, оскільки зі збільшенням бажаної точності зростає кількість міжвузлових комунікацій. Одиницею навантаження на підзадачу можна розглядати не лише один елемент ряду, але й певну кількість елементів, наприклад, проміжок ряду.

На рисунку 4.2 відображено графік залежності коефіцієнта прискорення програми від значення точності (ε) для кожної системи.

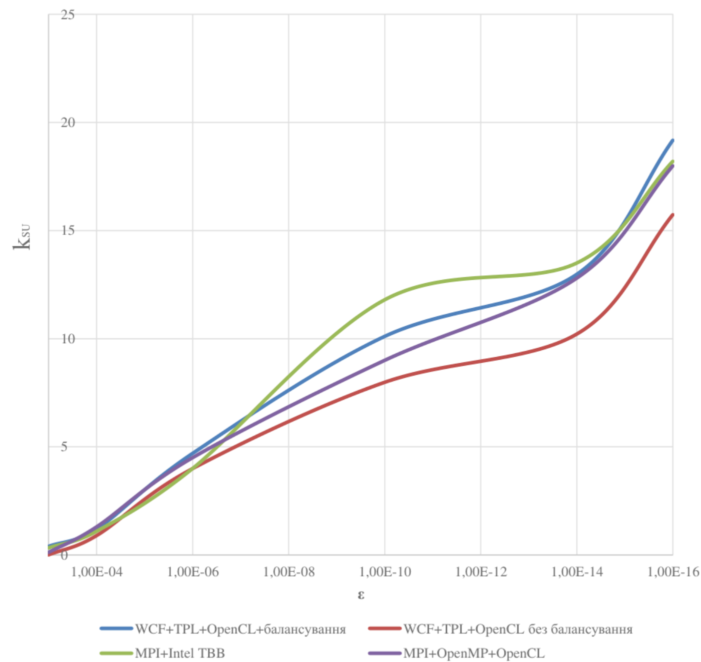
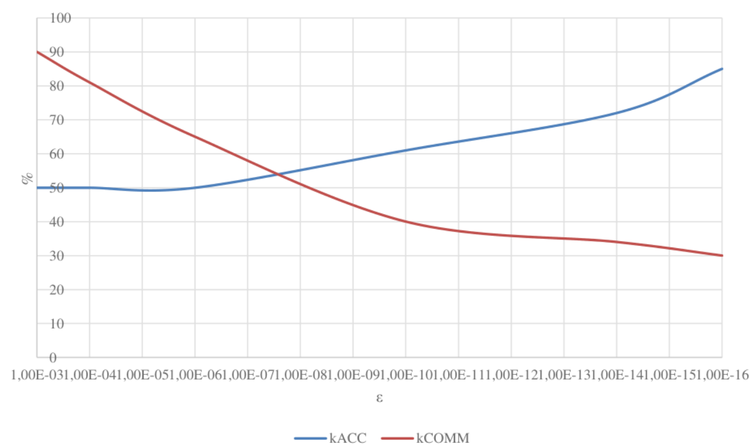


Рисунок 4.2 – Графік моделювання обчислення наближення ряду

Рисунок 4.3 відображає графіки залежності коефіцієнта використання акселераторів k_{ACC} та коефіцієнта використання мережі прискорення k_{COMM} програми від значення точності ε.



Рисунку 4.3 – Графік залежність коефіцієнта використання акселераторів та коефіцієнта використання мережі прискорення від значення точності

4.4.2 Моделювання підрахунку кількості зустрічей слова за допомогою моделі MapReduce

Модель обробки даних MapReduce є ключовою в контексті технологій Big Data й призначена для однотипної паралельної обробки великих обсягів даних, зазвичай для виконання операції згортки. Для алгоритму підрахунку кількості зустрічей кожного слова у документах за цією моделлю послідовність дій буде наступною. На етапі Map вузол-ініціатор розбиває документ на частини та розсилає ці частини до вузлів відповідно до запропонованого методу. Вузли, отримавши частини, відразу застосовують операцію Map, перетворюючи кожне отримане слово в список пар "ключ-значення", де ключ – це слово, а значення – 1. Вузли виконують операцію Reduce, яка полягає в згортці списку "ключ-значення" для підрахунку кількості співпадінь кожного ключа. Якщо є співпадіння між двома ключами, то одна пара "ключ-значення" видаляється, а значення іншої збільшується на 1. Вузли передають ініціаторові результуючі списки "ключ-значення", ініціатор виконує остаточну операцію Reduce по всіх отриманих часткових результатах. Це типа задача для розподілених КС, яка добре піддається розпаралелюванню. У запропонованому методі вона дозволяє перевірити ефективність балансування навантаження й загальної ефективності роботи системи. Відносною одиницею навантаження на підзадачу для цього алгоритму є частина документу, тобто масив слів.

На рисунку 4.4 зображено графік залежності коефіцієнту прискорення даної програми від значення розміру документу для відповідного програмного комплексу.

На рисунку 4.5 зображено графіки залежності коефіцієнту використання акселераторів та коефіцієнту використання мережі прискорення програми від значення розміру документу за допомогою моделі MapReduce для відповідного програмного комплексу.

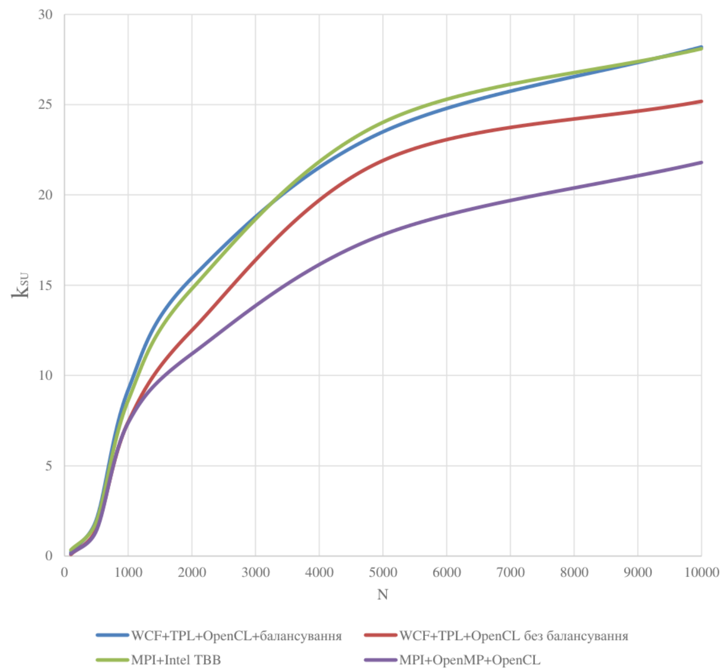


Рисунок 4.4 – Моделювання підрахунку кількості зустрічей слова за допомогою моделі MapReduce

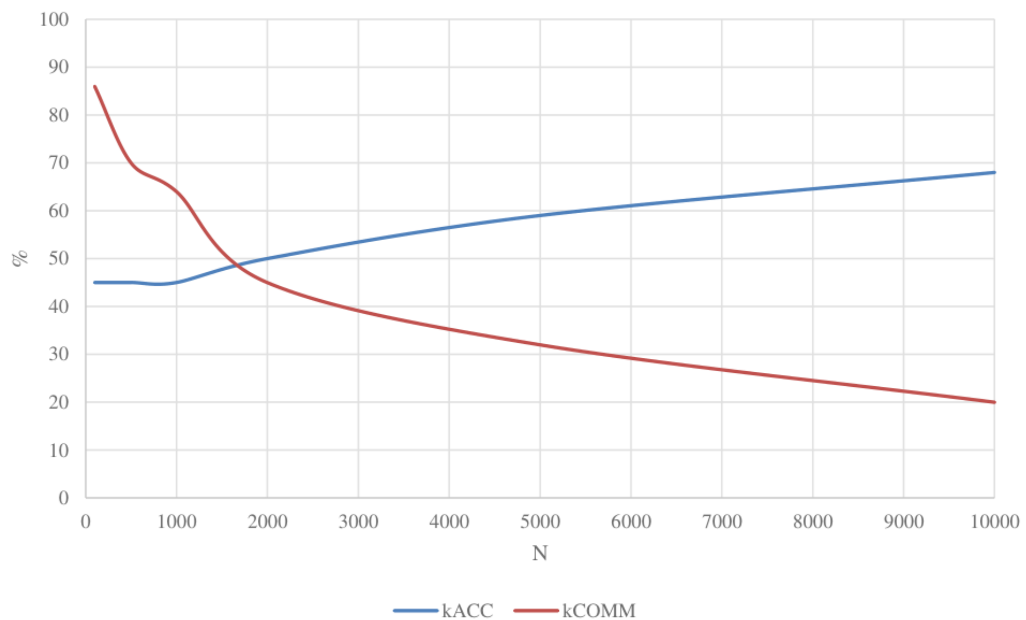


Рисунок 4.5 – Графік залежності коефіцієнту використання прискорювачів та коефіцієнту використання мережі прискорення обрахування кількості зустрічей слова за допомогою моделі MapReduce

На продуктивність розподілених систем, де комунікативним середовищем є глобальна мережа, суттєво впливає розкид латентності. Цей фактор може стати чинником принципових помилок на етапі оцінки навантаження та проведення статичного балансування в таких системах. Наступне тестування для обох розглянутих технологій організації рівня розподіленої системи (WCF та MPI) перевіряє середні показники відхилення за модулем часу передачі відповідних між собою пакетів від найбільшого із отриманих на всіх тестах середнього графіку (рисунок 4.6).

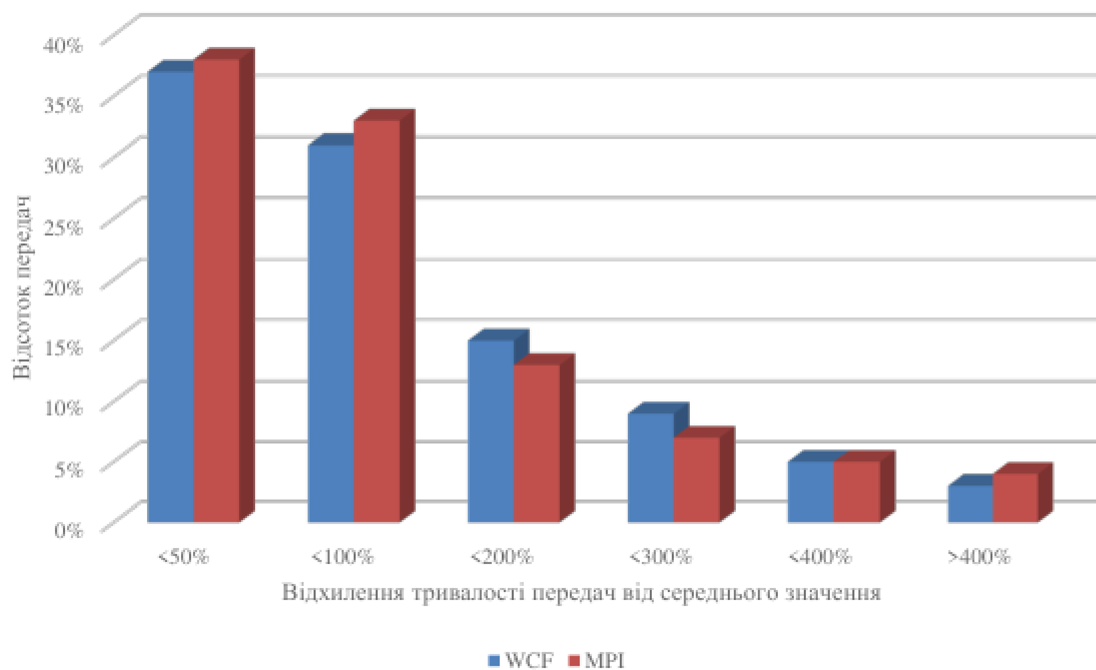


Рисунок 4.6 – Середні показники відхилення за модулем латентності

Такми чином, рамках кожного із програмних комплексів було організовано розв’язання ряду поширених математичних та прикладних задач, а саме: обчислення наближення розкладу функції в ряд та підрахунок кількості входження всіх слів в документ за моделлю MapReduce.

Застосування паралельних обчислень в розподілених КС з акселераторами дозволяє значно підвищити ефективність вирішення задач,

проте враховуючи особливості обраного підходу та технічні обмеження акселераторів. Організація деяких задач в рамках парадигми MapReduce є виправданою і може призвести до високого коефіцієнту прискорення. Важливим чинником є аналіз та вибір оптимальної стратегії паралельного виконання задачі з урахуванням особливостей задачі та характеристик розподіленої системи. Урахування розкиду латентності важливо для оцінки продуктивності та стабільності розподілених систем.

Застосування запропонованого методу порівняно з програмним комплексом, в якому використовувались технології WCF, TPL та OpenCL без балансування, дало середнє зростання коефіцієнту прискорення на 10%. Максимальне зростання на 15% спостерігалося для задач із незначними комунікаціями в процесі роботи, а мінімальне зростання на 5% для задач зі значною інтенсивністю комунікацій.

Отже, запропонований метод багаторівневого балансування навантаження є ефективним для підвищення продуктивності програмних комплексів, в умовах неоднорідних розподілених КС із акселераторами, де розподілені вузли різних класів, використання багаторівневого балансування навантаження дало значний приріст коефіцієнтів прискорення [17].

Застосування технологій WCF та TPL разом з розробленим методом балансування навантаження виявилося більш ефективним порівняно з технологіями MPI, OpenMP та OpenCL, особливо для задач з невеликою інтенсивністю комунікацій між вузлами системи.

Застосування акселераторів дозволило значно збільшити роль цих обчислювальних пристроїв у вирішенні задач, особливо при збільшенні об'ємів обчислень [16].

Такми чином, запропонований метод багаторівневого балансування навантаження є ефективним та може суттево покращити продуктивність КС.

ВИСНОВКИ

В сучасному світі паралельні комп'ютерні системи відіграють важливу роль у різних сферах людської діяльності, таких як наука, промисловість, медицина, фінанси та інші, при цьому виникає необхідність постійного вдосконалення паралельних алгоритмів обчислень, які здатні ефективно розподіляти завдання між численними процесорами або обчислювальними вузлами та забезпечуючи високу продуктивність.

В рамках виконання кваліфікаційної роботи було проведено аналіз поточного стану паралельної обробки даних, здійснено огляд та розвиток паралельних обчислень, розглянута еволюція обчислювальних платформ для паралельної обробки інформації, а також класифікація архітектур комп'ютерних систем. Визначено, що сучасні паралельні КС не завжди можуть ефективно вирішувати складні наукові задачі в прийнятний проміжок часу. Саме тому для обробки високонавантажених обчислень найбільш широко застосовують розподілені комп'ютерні системи. При цьому одні з перспективних є гетерогенні розподілені КС.

Також у роботі розглянуто принципи та методи паралельної обробки інформації, як на різних рівнях паралельності, так й сучасні моделі паралельної обробки інформації, такі як MapReduce, SIMD, Dataflow model. Крім того здійснено огляд технологій програмування паралельних КС: низькорівневої передачі повідомлень, розпаралелювання з застосуванням препроцесору, відвантажених обчислень та відкладених обчислень.

Визначено, що потрібно зупинити вибір на моделі із динамічним створенням потоків, з додаванням особливості автоматичного створення потоків, такі як автоматичний поділ задач та даних за заданим шаблоном та автоматичну регуляцію рівня паралелізму.

Запропоновано метод підвищення ефективності розподіленої паралельної комп’ютерної системи, у якому наведено порядок оцінки

вхідного завдання, відповідну процедуру ініціалізації КС, алгоритми оцінки графу системи та підвищення ефективності КС, а також процес делегування обрахунків між вузлами КС.

Також проведено моделювання роботи розподіленої паралельної комп'ютерної системи у рамках якого здійснено вибір прототипів компонентів системи, засобів програмної реалізації та програмних комплексів. Виконано моделювання роботи КС для на двох найпоширених прикладних задачах, кожна з яких має свою цінність, оскільки відповідає поширеному випадку в практичній організації паралельних програм в розподілених КС: обчислення наближення ряду та підрахунок кількості зустрічей слова за допомогою моделі MapReduce.

Визначено, що запропонований метод багаторівневого балансування навантаження є ефективним для підвищення продуктивності програмних комплексів, в умовах неоднорідних розподілених КС із акселераторами, й використання багаторівневого балансування навантаження дало значний приріст коефіцієнтів прискорення.

За темою кваліфікаційної роботи опубліковано тези доповіді [16] в рамках всеукраїнської науково-практичної конференції здобувачів вищої освіти і молодих учених «Комп'ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві» та тези доповіді [17] в рамках одинадцятої міжнародної науково-технічної конференції "Проблеми інформатизації" (додаток Б).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wolf W. Computers as components. Principles of embedded computing system design. USA : Elsevier Inc., 2012. 500 p.
2. Trobec R., Slivnik B., Bulić P., Robič B. Introduction to parallel computing: from algorithms to programming on state-of-the-art platforms // Springer, 2018. – p. 268
3. Dean J., Ghemawat S. Mapreduce: Simplified data processing on large clusters // OSDI. 2004. Pp. 137–150.
4. Рольщиков В.Б. Технології розподілених систем та паралельних обчислень. Конспект лекцій. Одеса: ОДЕКУ, 2016. – 155 с.
5. Паралельні та розподілені обчислення: навчальний посібник для вищих закладів освіти / К.Т. Кузьма, О.В. Мельник. – Миколаїв: ФОП Швець В.М., 2020. – 172 с.
6. Adamatzky A., Akl S., Sirakoulis G. From parallel to emergent computing // CRC Press, 2019. – 628 p
7. Lorenzon A., Filho A. Parallel computing hits the power wall: principles, challenges, and a survey of solutions // Springer briefs in computer science, 2019 – 88 p.
8. Czarnul P. Parallel Programming for modern high performance computing systems// CRC Press, 2018. – 304 p
9. Аксак Н.Г. Паралельні та розподілені обчислення / Н.Г. Аксак, О.Г. Руденко, А.М. Гуржій. – Х. : Компанія СМІТ, 2009. – 480 с.
10. Сабат Н.В. Паралельні та розподілені обчислення : конспект лекцій / Н.В. Сабат, В.Б. Кропивницька. – Івано-Франківськ: ІФНТУНГ, 2017. – 80 с.
11. Стіренко С. Г. Організації паралельний обчислювальних процесів в кластерних системах [Текст] / С.Г. Стіренко, – К.: «Три К», 2014. – 196 с.
12. Parallel program performance prediction using deterministic task graph analysis [Електронний ресурс] / Vikram S. Adve, Mary K. Vernon // ACM

Transactions on computer systems. – 2004. – Режим доступу до ресурсу: doi:10.1145/966785.966788 (дата звернення 22.12.2023).

13. Aiken, A. Optimal loop parallelization / A. Aiken, A. Nicolau // SIGPLAN Not. –1988. – Vol. 23, no. 7. – P. 308-317.

14. Brainerd, W.S., Landweber, L.H. Theory of Computation / W.S. Brainerd, L.H. Landweber. — Wiley, 1974.

15. 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.

16. Гулак А.С., наук. керівник Піскарьов О.М. Підвищення ефективності комп’ютерної системи засобами паралельної обробки інформації / Комп’ютерно-інтегровані технології автоматизації технологічних процесів на транспорті та у виробництві. Матеріали всеукраїнської науково- практичної конференції здобувачів вищої освіти і молодих учених. – Харків, ХНАДУ, 2023. – С. 109-110.

17. Гулак А.С., наук. керівник Піскарьов О.М. Підвищення ефективності комп'ютерної системи за допомогою паралельної обробки даних / Проблеми інформатизації. Тези доповідей одинадцятої міжнародної науково-технічної конференції 16 – 17 листопада 2023 р. 3 (4) – Харків: ХНУРЕ, 2023 – С.68.