МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломної роботи

магістра
(освітній ступінь)

на тему «Експериментальне дослідження швидкодії методів інтелектуального аналізу даних на основі технології паралельних обчислень»

ХАІ.603.667п1.121.156310.20О

Виконав: студент 6 курсу групи № 667п1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Коваль М.Ю.
(прізвище й ініціали студента)
Керівник Шостак І.В.
(прізвище та ініціали)
Рецензент Ткачов В.М.
(прізвище та ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)

Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Коваль Максиму Юрійовичу
(прізвище, ім’я, по батькові)

1. Тема дипломного проекту *Експериментальне дослідження швидкодії методів інтелектуального аналізу даних на основі технології паралельних обчислень*
керівник дипломного проекту *Шостак Ігор Володимирович, д.т.н, професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “__” ___2020 року № __
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи: *програмний застосунок для проведення експериментальних досліджень кластеризації надвеликих масивів даних*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 критичний огляд методів інтелектуального аналізу даних при роботі з надвеликими масивами інформації;
2 критичний огляд застосування технології паралельних обчислень для рішення задач кластеризації надвеликих масивів даних;
3 розробка удосконалених методів кластеризації надвеликих масивів даних на основі паралельних обчислень;
4 розробика програмного застосунку для проведення експериментальних досліджень кластеризації надвеликих масивів даних.

5. Перелік графічного матеріалу
РПЗ – стор. 82, рисунків – 19 шт., таблиць – 7 шт., презентація – 23 слайди.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Шостак І. В., проф. каф. 603		
2	Шостак І. В., проф. каф. 603		
3	Шостак І. В., проф. каф. 603		

8. Нормоконтроль _____ __В.А. Постернакова__ «____» _____2020 р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Отримання і затвердження теми диплому	03.09.2019	
2	Аналіз предметної області	04.09.2019	
3	Постановка задачі	20.11.2019	
4	Проведення теоретичних досліджень	22.11.2019	
5	Розробка прототипу ПЗ	02.09.2020	
6	Підготовка пояснювальної записки	22.10.2020	
7	Оформлення пояснювальної записки до дипломного проекту	10.11.2020	
8	Передзахист дипломного проекту	24.11.2020	
9	Захист дипломного проекту	04.12.2020	

Студент _____ Коваль М.Ю._____
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шостак І.В._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту містить 82 стор., 19 рис., 15 джерел.

Об’єктом дослідження є процеси інтелектуального аналізу даних.

Предметом дослідження є методи кластеризації надвеликих масивів даних.

Метою досліджень є підвищення ефективності методів кластеризації даних шляхом використання технології паралельних обчислень за рахунок розроблення програмного забезпечення для аналізу надвеликих масивів даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести критичний огляд методів інтелектуального аналізу даних при роботі з надвеликими масивами інформації;

– провести критичний огляд застосування технології паралельних обчислень для рішення задач кластеризації надвеликих масивів даних;

– розробити удосконалені методи кластеризації надвеликих масивів даних на основі паралельних обчислень;

– розробити програмний застосунок для проведення експериментальних досліджень кластеризації надвеликих масивів даних.

Методи досліджень. У роботі було використано методи кластерного аналізу, методи системного аналізу.

Наукова новизна. Удосконалено метод кластеризації даних який на відміну від існуючих використовує механізм паралельних обрахунків для інтелектуального аналізу надвеликих масивів даних, що дає можливість підвищити швидкодію кластеризації даних.

Практичне значення отриманих результатів. В результаті роботи був розроблений програмний продукт для кластерного аналізу даних, який може бути використаний при роботі з надвеликими масивами інформації.

ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ, КЛАСТЕРНИЙ АНАЛІЗ, НАДВЕЛИКІ МАСИВИ ДАНИХ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ

ABSTRACT

Explanatory note to the graduate work contains 82 pp., 19 fig., 15 sources.

The object of research is the processes of data mining.

The subject of research is the methods of large data sets clustering.

The aim of the research is to increase the efficiency of data clustering methods through the use of parallel computing technology by development of software for the analysis of ultra-large data sets.

To achieve this goal it is necessary to solve the following tasks:

– conduct a critical review of data mining methods when working with large arrays of information;

– conduct a critical review of the application of parallel computing technology to solve problems of clustering of large data sets;

– develop improved methods for clustering ultra-large data sets based on parallel computations;

– develop a software application for conducting experimental studies of ultra-large data sets clustering.

Research methods. The methods of cluster analysis, methods of system analysis were used in the work.

Scientific novelty. The method of data clustering has been improved, which, unlike the existing ones, uses the mechanism of parallel calculations for intelligent analysis of ultra-large data sets, which makes it possible to increase the speed of data clustering.

The practical significance of the results. As a result, a software product for cluster data analysis was developed, which can be used when working with very large arrays of information.

INTELLECTUAL DATA MINING, CLUSTER ANALYSIS, LARGE DATA ARRAYS, PARALLEL COMPUTATIONS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

КАД – кластерний аналіз даних.

ПЗ – програмний застосунок.

ПП – програмний продукт.

ВСТУП

Актуальність теми. Глобальна комп’ютеризація, прогрес в сфері мікроелектроніки та інформаційних технологій обумовлюють збільшення об’ємів інформації. Загальний об’єм даних на 2012 рік складав більше 1,8 зеттабайт (1,8 трлн Гб) та за дослідженнями IDC (International Data Corporation) ця цифра подвоюється кожні 2 роки[9]. Дев'яносто відсотків даних у світі сьогодні створено лише за останні два роки. Наш поточний вихід даних складає приблизно 2,5 мільярда Гб на день. Збереження такого об’єму інформації потребує немалих ресурсів та зусиль. Але найбільша проблема полягає не в збереженні даних, а в їх обробці - за прогнозами IDC в 2020 році з усіх об’ємів даних лише 35% будуть корисними. Таким чином потрібно механізм для спрощення даних для подальшого їх аналізу. Одним з таким методів є кластеризація даних.

Кластерний аналіз або кластеризація – це процес об'єднання набору об'єктів в групи – кластери – таким чином, щоб в кожному кластері об’єкти були більш схожими один до одного, ніж до об’єктів із інших кластерів. Це головне завдання пошукового виявлення даних, а також загальна методика аналізу статистичних даних, що використовується у багатьох областях, включаючи машинне навчання, розпізнавання образів, аналіз зображень, пошук інформації, біоінформатика, стиснення даних та комп'ютерна графіка[10].

Кластеризація застосовується для того, щоб зробити так зване стиснення даних, тобто скоротити обсяг використовуваних даних за рахунок того, що всередині кластера об'єкти не розрізняються (розглядаються як один об'єкт).

Тому задача автоматичної та якісної кластеризації даних за прийнятний час, без попереднього структурування та дослідження інформації, є задачею, яка варта уваги та досліджень.

Дана робота присвячена розробці модифікації для існуючих алгоритмів кластерного аналізу шляхом застосування паралельних розрахунків.

Використання розподілених систем є досить складним і вимагає додаткових затрат на устаткування. В той же час користувачі на побутовому рівні мають змогу використовувати графічні процесори для паралельних обчислень. В сучасному світі напрямок обчислень еволюціонує від централізованої обробки даних на центральному процесорі до спільної обробки на CPU і GPU. Для реалізації нової обчислювальної парадигми, наприклад, компанія NVIDIA винайшла архітектуру паралельних обчислень CUDA. Сьогодні усі нові графічні карти Nvidia підтримують технологію CUDA – програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність та має високу продуктивність паралельних обчислень [2].

Об'єктом дослідження є процеси інтелектуального аналізу даних.

Предметом дослідження є методи кластеризації надвеликих масивів даних.

Метою досліджень є підвищення ефективності методів кластеризації даних шляхом використання технології паралельних обчислень за рахунок розроблення програмного забезпечення для аналізу надвеликих масивів даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести критичний огляд методів інтелектуального аналізу даних при роботі з надвеликими масивами інформації;

– провести критичний огляд застосування технології паралельних обчислень для рішення задач кластеризації надвеликих масивів даних;

– розробити удосконалені методи кластеризації надвеликих масивів даних на основі паралельних обчислень;

– розробити програмний застосунок для проведення експериментальних досліджень кластеризації надвеликих масивів даних.

Методи досліджень. У роботі було використано методи кластерного аналізу, методи системного аналізу.

Наукова новизна. Удосконалено метод кластеризації даних який на відміну від існуючих використовує механізм паралельних обрахунків для

інтелектуального аналізу надвеликих масивів даних, що дає можливість підвищити швидкодію кластеризації даних.

Практичне значення отриманих результатів. В результаті роботи був розроблений програмний продукт для кластерного аналізу даних, який може бути використаний при роботі з надвеликими масивами інформації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Постановка мети й завдань дослідження

Дев'яносто відсотків даних у світі сьогодні створено лише за останні два роки. Щодня створюється близько трьох мільярдів гігабайтів нової інформації. Збереження такого об’єму даних потребує немалих ресурсів та зусиль. Але найбільша проблема полягає не в збереженні даних, а в їх обробці – за прогнозами в 2020 році лише 35% даних будуть корисними. Тому інтелектуальний аналіз даний є актуальною проблемою сьогодення. Кластерний аналіз \є потужним механізм для спрощення даних для подальшого їх аналізу. Тому для дослідження було обрано методи кластеризації k-means та c-means.

Обчислення на центральному процесорі є досить повільними для великих об’ємів даних, тому паралельні обчислення активно застосовуються для інтелектуального аналізу даних. Використання розподілених систем для паралельних обчислень є досить складним і вимагає додаткових затрат на устаткування. Через це було вирішено використовувати графічні процесори Nvidia з підтримкою технології CUDA, яка має високу продуктивність паралельних обчислень.

Метою досліджень є підвищення ефективності методів кластеризації даних шляхом використання технології паралельних обчислень за рахунок розроблення програмного забезпечення для аналізу надвеликих масивів даних.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести критичний огляд методів інтелектуального аналізу даних при роботі з надвеликими масивами інформації;

– провести критичний огляд застосування технології паралельних обчислень для рішення задач кластеризації надвеликих масивів даних;

– розробити удосконалені методи кластеризації надвеликих масивів даних на основі паралельних обчислень;

– розробити програмний застосунок для проведення експериментальних досліджень кластеризації надвеликих масивів даних.

1.2 Інтелектуальний аналіз даних

Найбільш часто використовуваними методами інтелектуального аналізу даних є методи кластеризації. Завдання кластеризації на відміну від інших відомих задач машинного навчання, таких, наприклад, як завдання регресійного аналізу та завдання класифікації, передбачає так зване «навчання без учителя» [10] («unsupervised learning») - вона полягає в тому, щоб надати досліднику інструмент для автоматичного поділу наявних об'єктів на класи на підставі подібності тих чи інших характеристик (факторів) цих об'єктів. При цьому ні самі класи, ні їх кількість заздалегідь не відомі. Дана задача вирішується на початкових етапах дослідження, коли про дані є мало інформації.

Кластери будуються таким чином, щоб характеристики об'єктів одного класу були дуже близькі і при цьому сильно відрізнялися б від характеристик об'єктів інших кластер. Зрозуміло, розуміння «подібності» і «відмінності» сильно залежать від предметної області і навіть від конкретного завдання.

При проведенні кластеризації також можуть виявитися нетипові об'єкти, які не можна віднести до жодного з класів - це може дати новий корисний матеріал для дослідження.

Для наукових досліджень вивчення результатів кластеризації, а саме причин, за якими об’єкти об’єднуються в певні групи, можуть відкрити нові аспекти і перспективні напрями досліджень[1]. Найпопулярнішим прикладом такої можливості являється періодична таблиця хімічних елементів. Менделєєв розділив 60 відомих елементів на кластери чи періоди. Елементи, що попали, в одну групу мали схожі властивості. Вивчення причин цього явища, в значній мірі визначало пріоритети наукових досліджень на роки вперед.

Для задачі кластеризації характерна відсутність будь-яких відмінностей між об’єктами та їх характеристиками.

Це універсальний алгоритм, який можна використовувати для будь-якого типу групування. Деякими прикладами використання є:

- поведінкове групування:
  - групування за історією покупки;
  - групування за активністю в додатках чи веб-сайтах;
  - визначення груп осіб на основі інтересів;
  - створення профілів на основі моніторингу активності;
- сортування вимірювань сенсорів:
  - визначення типів активності в датчиках руху;
  - групування зображень;
  - відокремлення аудіо потоків;
  - визначення груп по показниках моніторингу здоров'я;
- відокремлення активності реальних людей від ботів.

Рішення завдання кластеризації є неоднозначним, і тому є кілька причин[3]:

- не існує однозначно найкращого критерію якості кластеризації. Відомий цілий ряд евристичних критеріїв, а також ряд алгоритмів, які не мають чітко вираженого критерію, але здійснюють досить розумну кластеризацію. Всі вони можуть давати різні результати;
- число кластерів, як правило, невідоме заздалегідь і встановлюється відповідно до деякого суб'єктивного критерію;
- результат кластеризації істотно залежить від метрики, вибір якої, як правило, також суб'єктивний і визначається експертом.

1.3 Кластеризація даних

В загальному можна виділити дві основні класифікації методів кластерного аналізу:

– ієрархічні і неієрархічні. Ієрархічні алгоритми (також звані алгоритмами таксономії) будують не одне розбиття вибірки на непересічні кластери, а систему вкладених розбиття. На виході ми отримуємо дерево кластерів, коренем якого є вся вибірка, а листям - найбільш дрібні кластера. неієрархічні алгоритми будують одне розбиття об'єктів на кластери;

– чіткі і нечіткі. Чіткі (або непересічні) алгоритми кожному об'єкту вибірки ставлять у відповідність номер кластера, тобто кожен об'єкт належить тільки одному кластеру. Нечіткі (або пересічні) алгоритми кожному об'єкту ставлять у відповідність набір речових значень, що показують ступінь відносини об'єкта до кластерів. Тобто кожен об'єкт відноситься до кожного кластера з певною ймовірністю.

При використанні ієрархічних алгоритмів можна використовувати одну із наведених метрик[5]:

– метод ближнього сусіда або одиночний зв'язок. Тут відстань між двома кластерами визначається відстанню між двома найбільш близькими об'єктами (найближчими сусідами) в різних кластерах. Цей метод дозволяє виділяти кластери складної форми за умови, що різні частини таких кластерів з'єднані ланцюжками близьких один до одного елементів. В результаті роботи цього методу кластери представляються довгими ланцюжками;

– метод найбільш віддалених сусідів або повний зв'язок. Тут відстані між кластерами визначаються найбільшою відстанню між будь-якими двома об'єктами в різних кластерах (найбільш віддаленими сусідами). Метод добре використовувати, коли об'єкти дійсно відбуваються з різних груп. Якщо ж кластери мають подовжену форму то цей метод не слід використовувати;

– метод Варда. Як відстань між кластерами береться приріст суми квадратів відстаней об'єктів до центрів кластерів, що отримується в результаті їх об'єднання. На відміну від інших методів кластерного аналізу для оцінки відстаней між кластерами, тут використовуються методи дисперсійного аналізу. На кожному кроці алгоритму об'єднуються такі два кластери, які призводять до

мінімального збільшення цільової функції, суми квадратів елементів кожної групи. Цей метод направлений на об'єднання близько розташованих кластерів і використовується для створення кластерів невеликого розміру;

– метод невиваженого попарного середнього. Як відстань між двома кластерами береться середня відстань між усіма парами об'єктів в них. Цей метод слід використовувати, якщо об'єкти дійсно належать до різних груп, коли кластери мають подовжену форму, та якщо є припущення щодо істотних відмінностей в розмірах кластерів;

– метод зваженого попарного середнього. Цей метод схожий на метод невиваженого попарного середнього, різниця полягає лише в тому, що тут в якості вагового коефіцієнта використовується розмір кластера (число об'єктів, що містяться в кластері);

– незважений центроїдний метод. Як відстань між двома кластерами в цьому методі береться відстань між їх центрами тяжкості. Цей метод переважно використовувати у випадках, якщо є припущення щодо істотних відмінностей в розмірах кластерів;

– зважений центроїдний метод. Цей метод схожий на попередній, різниця полягає в тому, що для обліку різниці між розмірами кластерів (числа об'єктів в них), використовуються ваги. Цей метод переважно використовувати у випадках, якщо є припущення щодо істотних відмінностей в розмірах кластерів.

Серед алгоритмів ієрархічної кластеризації виділяються два основних типи: висхідні і низхідні алгоритми. Низхідні алгоритми працюють за принципом «зверху-вниз»: на початку всі об'єкти поміщаються в один кластер, який потім розбивається на всі більш дрібні кластери. Більш поширені висхідні алгоритми, які на початку роботи поміщають кожен об'єкт в окремий кластер, а потім об'єднують кластери в усе більші, поки всі об'єкти вибірки не будуть міститися в одному кластері[1]. Результати таких алгоритмів зазвичай представляють у вигляді дерева - дендрограми.

Для обчислення відстаней між кластерами частіше за все користуються двома відстанями: одиночній зв'язок або повний зв'язок[4]. До недоліку ієрархічних алгоритмів можна віднести систему повного розбиття, яка може бути зайвою в контексті розв'язуваної задачі.

Велику популярність при вирішенні задач кластеризації набули алгоритми, засновані на пошуку оптимального розбиття множини даних на кластери (групи) – неієрархічні методи. В багатьох задачах в силу своїх достоїнств використовуються саме алгоритми розбиття. Дані алгоритми намагаються групувати дані в кластери таким чином, щоб цільова функція алгоритму розбиття досягала екстремуму (мінімуму). В даних алгоритмах використовуються наступні базові поняття:

- вхідна множина даних;
- метрика відстані;
- вектор центрів кластерів;
- матриця розбиття кластерів U;
- цільова функція;
- набір обмежень.

Ще одним класом алгоритмів кластерного аналізу є алгоритми засновані на теорії графів. Суть таких алгоритмів полягає в тому, що вибірка об'єктів представляється у вигляді графа G = (V, E), вершинам якого відповідають об'єкти, а ребра мають вагу, рівній відстані між об'єктами. Перевагою алгоритмів кластеризації на основі графів є наочність, відносна простота реалізації і можливість різних удосконалень, заснованих на геометричних міркуваннях. Основними алгоритмами є алгоритм виділення зв'язкових компонент, алгоритм побудови мінімального кістякового дерева і алгоритм пошарової кластеризації.

Однією з основних проблем кластерного аналізу є визначення кількості кластерів, що для багатьох алгоритмів є вхідним параметром. Розбиття на кластери можуть суттєво відрізнятися в залежності від обраної кількості кластерів.

Існує ряд методів для обчислення оптимальної кількості кластерів[8] заснованих на :

– індексах, які порівнюють ступені "розкиду" даних усередині кластерів і між кластерами;

– розрахунку значень характеристик (функцій стійкості), що показують відповідність призначених кластерів для окремих елементів;

– статистиках, що визначають найбільш ймовірне рішення;

– оцінюванні цільності розподілів.

Розглянемо декілька відомих методів на основі індексів.

– перший підхід (Calinski-Harabasz) вибирає кількість кластерів як значення аргументу, максимізує функцію *CH(K)* [8]*:*

$$CH(K) = \frac{B(K)/(K-1)}{W(K)/(n-K)} \tag{1.1}$$

де *B(K)* і *W(K)*, відповідно, зовнішня і внутрішня суми квадратів елементів даних з K кластерами. Це один з найперших запропонованих методів. Він виявляється ефективним при даних невеликої розмірності;

– підхід Krzanowski-Lai максимізує функцію *KL(K)* [8]*:*

$$KL(K) = \left|\frac{Diff(K)}{Diff(K+1)}\right| \tag{1.2}$$

де

$$Diff(K) = (K-1)^{\frac{2}{p}}W(K-1) - (K)^{\frac{2}{p}}W(K) \tag{1.3}$$

де p – розмірність, *W(K)* – внутрішня сума квадратів елементів даних з K кластерами. Основна ідея полягає в вимірі порядку мінливості внутрішніх дисперсій.

– Hartigan пропонує вибирати таке найменше значення K, що *H(K)* менше або дорівнює 10[8]:

$$H(K) = (n - K - 1)(W(K)W(K + 1) - 1) \quad (1.4)$$

– Kaufman and Rousseeuw пропонують вимірювати, наскільки і-та точка була добре кластерізована. Для цього визначають функцію[8]:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \quad (1.5)$$

де *a(i)* - середня відстань між і-тою точкою і всіма іншими, які потрапили в той же кластер, *b(i)* - середня відстань до точок в найближчому кластері, де під найближчим кластером розуміється той, який мінімізує *b(i)*. Кількість кластерів вважається вірним, якщо воно максимізує середнє значення *s(i)*.

Суть одного з можливих удосконалень полягає в знаходженні більш "хороших" початкових значень центроїдів кластерів. Оригінальні методи не регламентують те, як виконується цей етап, і тому інколи є нестабільними. Метод запропоновано в 2007 році Девідом Артуром та Сергієм Васильвітським.

Точний метод виглядає наступним чином[12]:

– вибрати перший центр ваги випадковим чином (серед усіх точок);

– для кожної точки знайти значення квадрата відстані до найближчого центроїда (з тих, які вже обрані) - dx^2;

– вибрати з цих точок наступний центроїд так, щоб ймовірність вибору точки була пропорційна обчисленому для неї квадрату відстані. Це можна зробити наступним чином. На кроці 2 потрібно паралельно з розрахунком dx^2 підраховувати суму Sum(dx^2). Після накопичення суми знайти значення Rnd = random (0.0,1.0) * Sum(dx^2). Rnd випадковим чином вкаже на число з інтервалу [0; Sum), і нам залишається тільки визначити, якій точці це відповідає. Для цього потрібно знову почати підраховувати суму S (dx^2) до тих пір, поки сума не

перевищить Rnd. Як тільки це станеться, підсумовування зупиняється, і ми можемо взяти поточну точку в якості центроїда. При виборі кожного наступного центроїда спеціально стежити за тим, щоб він не збігся з однією з вже обраних в якості центроїда точок, не потрібно, тому що ймовірність повторного вибору деякої точки дорівнює 0;

– повторювати кроки 2 і 3 до тих пір, поки не будуть знайдені всі необхідні центроїди.

В подальших розділах буде детально розглянуто три ітераційних неієрархічних алгоритмів, два з яких було обрано для модифікації.

1.3.1 Метод k -середніх

Метод k-середніх (англ. k-means) був запропонований ще в 1979 році, проте він не втрачає своєї актуальності і сьогодні. Ідея, що лежить в його основі, досить проста, і метод працює досить ефективно. Однак, оптимальність рішень, отриманих за допомогою Методу k-середніх, не гарантована. Крім того, недоліком методу є необхідність знати число кластерів заздалегідь.

Формально рішення задачі кластеризації полягає в тому, щоб «помітити» кожен з наявних об'єктів - приписати йому номер певного класу. Метод k-середніх передбачає таке розбиття об'єктів на класи, при якому мінімізуються відмінності («відстані») між об'єктами одного і того ж класу та максимізуються рас стояння між об'єктами різних класів.

Даний метод являється прообразом практично всіх методів нечіткої кластеризації, і його розгляд допоможе нам краще зрозуміти принципи, що закладені в більш складні методи.

В загальному вигляді метод представляє собою ітераційну процедуру[1]:

Крок 1. Проініціалізувати початкову матрицю розбиття U випадковим чином і обрати точність δ, що буде використовуватися для завершення методу, встановити номер ітерації $l = 0$;

Крок 2. Визначити центри кластерів за наступною формулою:

$$c_l^{(i)} = \frac{\sum_{j=1}^{d} u_{ij} m_j}{\sum_{j=1}^{d} u_{ij}}, 1 \le i \le c \quad (1.6)$$

де $c_l^{(i)}$– центри кластерів, *d* – вимір об'єкта, що досліджується, *c* – кількість кластерів, *l* – номер поточної ітерації, *u* – матриця розбиття, *m* – об'єкт, що досліджується;

Крок 3. Обновити матрицю розбиття:

$$u_{ij}^{(l)} = \begin{cases} 1, & \text{при } d(m_j, c_i) = \min\limits_{l \le k \le c} d(m_j, c_k) \\ 0, & \text{в інших виппадках} \end{cases} \quad (1.7)$$

де $u_{ij}^{(l)}$– елемент матриці розбиття, *d(x,y)* – обрана метрика, *c* –кластер, *m* – об'єкт, що досліджується;

Крок 4. Перевірити умову $\left\| U^{(l)} - U^{(l-1)} \right\| < \delta$, де *U* –матриця розбиття, і δ – обрана точність. Якщо умова виконується – завершити процес, якщо ні – перейти до кроку 2 з номером ітерації *l = l + 1*.

Також існує альтернативний варіант даного методу:

Крок 1. Випадковим чином обрати центри кластерів із елементів вхідних даних і обрати точність δ, що буде використовуватися для завершення методу, встановити номер ітерації *l* = 0;

Крок 2. Оновити матрицю розбиття(формула 1.14);

Крок 3. Визначити центри кластерів(формула 1.13);

Крок 4. Перевірити наскільки змістилися центри кластерів. Якщо вони змістилися менше ніж на точність δ – завершити процес, якщо ні – перейти до кроку 2 з номером ітерації *l = l + 1*.

Також існує набір обмежень: $u_{ij}^{(l)} \{0,1\}$; $\sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d$, який визначає, що кожен вектор даних може належати тільки одному кластеру і не належати іншим. В кожному кластері повинно бути не менше одного елемента даних і не більше загальної кількості елементів.

Основний недолік даного методу через дискретність елементів матриці розбиття – великий розмір просторового розбиття. Одним із способів усунення даного недоліку є представлення елементів матриці розбиття числами із інтервалу від 0 до 1. Тобто належність елемента даних до кластера визначається функцією належності – елемент даних може належати декільком кластерам з різною степеню належності. Даний підхід реалізований в методі нечіткої кластеризації с-середніх(Fuzzy C-Means).

1.3.2 Метод с-середніх

Алгоритм с-середніх (англ. fuzzy c-means)відрізняється від попереднього тим, що кластери тепер являються нечіткими множинами і кожний елемент даних належить до різних кластерів з різною степеню приналежності[1].

Крок 1. Вибрати кількість кластерів $2 \leq c \leq d$, і обрати точність δ, що буде використовуватися для завершення алгоритму, встановити номер ітерації $l = 0$;

Крок 2. Вибрати коефіцієнт нечіткості $w = 2$;

Крок 3. Проініціалізувати початкову матрицю розбиття U випадковим чином;

Крок 4. Визначити центри кластерів за наступною формулою:

$$c_l^{(i)} = \frac{\sum_{j=1}^{d}(u_{ij})^w m_j}{\sum_{j=1}^{d}(u_{ij})^w}, 1 \leq i \leq c \qquad (1.8)$$

де $c_l^{(i)}$– центри кластерів, d – вимір об’єкта, що досліджується, c – кількість кластерів, l – номер поточної ітерації, u – матриця розбиття, m – об’єкт, що досліджується, w – коефіцієнт нечіткості;

Крок 5. Для всіх елементів даних обчислити квадрати відстаней до всіх центрів кластерів:

$$d_A^2\left(m_j, c_l^{(i)}\right) = \left(c_l^{(i)} - m_j\right)^i A\left(c_l^{(i)} - m_j\right); \quad (1.9)$$

Крок 6. Оновити матрицю розбиття:

$$u_{ij}^{(l)} = \frac{1}{\sum_{k=1}^{c}\left(\frac{d_A^2\left(m_j, c^{(l)}\right)}{d_A^2\left(m_j, c^{(k)}\right)}\right)^{\frac{1}{w-1}}}, \text{для } 1 \le i \le c, 1 \le j \le d \quad (1.10)$$

де $u_{ij}^{(l)}$ – елемент матриці розбиття, *d(x,y)* – обрана метрика, *c* –кластер, *m* – об'єкт, що досліджується, *w* – коефіцієнт нечіткості;

Крок 7. Перевірити умову $\left\|U^{(l)} - U^{(l-1)}\right\| < \delta$. Якщо умова виконується – завершити процес, якщо ні – перейти до кроку 4 з номером ітерації *l = l + 1*.

Також існує набір обмежень: $u_{ij}^{(l)}$ $[0,1]$; $\sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d$, який визначає, що кожен вектор даних може належати різним кластерам з різною степеню приналежності, сума степенів приналежності елементів даних до різних кластерів повинна бути рівною одиниці. В кожному кластері повинно бути не менше одного елемента даних і не більше загальної кількості елементів.

1.3.3 Метод Гюставсона-Кеселя

Даний метод маю одну велику відмінність від попередніх, а саме він шукає кластери в формі еліпсоїдів, а не круглі чи циліндричні. Це робить даний метод більш гнучким у використанні. Також кластери характеризується не тільки центром, а і коваріаційною матрицею[1]:

$$F^{(i)} = \frac{\sum_{j=1}^{d}\left(u_{ij}\right)^w (m_j - c^{(i)})(m_j - c^{(i)})^T}{\sum_{j=1}^{d}\left(u_{ij}\right)^w}. \quad (1.11)$$

Даний метод використовує власну нормуючу матрицю для обчислення відстаней:

$$d^2 = \left(c^{(i)} - m_j\right)^T A^{(i)} \left(c^{(i)} - m_j\right), \tag{1.12}$$

де

$$A^{(i)} = \left(\left|F^{(i)}\right|\right)^{\frac{1}{r+1}} \left(F^{(i)}\right)^{-1}. \tag{1.13}$$

Власне метод наступний[1]:

Крок 1. Вибрати кількість кластерів $2 \leq c \leq d$, і обрати точність δ, що буде використовуватися для завершення методу, встановити номер ітерації $l = 0$;

Крок 2. Вибрати коефіцієнт нечіткості $w = 2$;

Крок 3. Проініціалізувати початкову матрицю розбиття U випадковим чином;

Крок 4. Визначити центри кластерів за наступною формулою:

$$c_l^{(i)} = \frac{\sum_{j=1}^{d} (u_{ij})^w m_j}{\sum_{j=1}^{d} (u_{ij})^w}, 1 \leq i \leq c \tag{1.14}$$

де $c_l^{(i)}$– центри кластерів, d – вимір об’єкта, що досліджується, c – кількість кластерів, l – номер поточної ітерації, u – матриця розбиття, m – об’єкт, що досліджується, w – коефіцієнт нечіткості;

Крок 5. Розрахувати коваріаційні матриці кластерів по формулі

Крок 6. Для всіх елементів даних обчислити квадрати відстаней до всіх центрів кластерів по формулі

Крок 7. Оновити матрицю розбиття:

$$u_{ij}^{(l)} = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_A^2\left(m_j, c^{(l)}\right)}{d_A^2\left(m_j, c^{(k)}\right)} \right)^{\frac{1}{w-1}}}, \text{для } 1 \leq i \leq c, 1 \leq j \leq d \tag{1.15}$$

де $u_{ij}^{(l)}$ – елемент матриці розбиття, *d(x,y)* – обрана метрика, *c* –кластер, *m* – об'єкт, що досліджується, *w* – коефіцієнт нечіткості;

Крок 8. Перевірити умову $\left\|U^{(l)} - U^{(l-1)}\right\| < \delta$. Якщо умова виконується – завершити процес, якщо ні – перейти до кроку 4 з номером ітерації *l = l + 1*.

Також існує набір обмежень: $u_{ij}^{(l)}\ [0,1]$; $\sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d$, який визначає, що кожен вектор даних може належати різним кластерам з різною степеню приналежності, сума степенів приналежності елементів даних до різних кластерів повинна бути рівною одиниці. В кожному кластері повинно бути не менше одного елемента даних і не більше загальної кількості елементів.

З огляду на розглянуту вище інформацію можна зробити такі припущення про дані три методи:

– в загальному кластери мають форму еліпсоїда;

– у кожного кластера завжди є центр;

– віднесення елементів до кластери базується на відстані точок до центра кластера на основі обраної метрики.

Також з наведених припущень випливають основі недоліки даних методів[1]:

– припущення, що кластери мають певну сталу форму, не завжди правдиве, тому результат кластеризації для даних які мають складне взаємне розташування можу бути взагалі не точним і його не можливо буде інтерпретувати;

– припущення, що кластери мають центр – певну точку, яка має рівень приналежності до певного кластера рівним одиниці, а інші точки не можуть належати до кластера з таким же самим рівнем приналежності, веде до того, що результат кластеризації для даних які мають складне взаємне розташування може бути не прийнятним;

– дані методи засновані не на основі взаємного розміщення точок один від одного, а тільки на розміщенні точок відносно центра кластерів.

1.4 Технологія паралельних обчислень

Основною складовою роботи є удосконалення даних методів кластеризації. Для прискорення доцільно розпаралелювати обрахунки. Використання розподілених систем є досить складним і вимагає додаткових затрат на устаткування. В той же час користувачі на побутовому рівні мають змогу використовувати графічні процесори Nvidia з підтримкою технології CUDA, яка має високу продуктивність паралельних обчислень.

CUDA (англ. Compute Unified Device Architecture) – програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів фірми Nvidia [2]. CUDA SDK дозволяє програмістам реалізовувати на спеціальному спрощеному діалекті мови програмування C алгоритми, здійснювані на графічних процесорах Nvidia, і включати спеціальні функції в текст програми на C. Архітектура CUDA дає розробнику можливість на свій розсуд організовувати доступ до набору інструкцій графічного прискорювача і управляти його пам'яттю. Первинна версія CUDA SDK була представлена 15 лютого 2007 року. У основі CUDA API лежить розширена мова C. Для успішної трансляції коду цією мовою, до складу CUDA SDK входить власний C-компілятор командного рядка nvcc компанії Nvidia. Компілятор nvcc створений на основі відкритого компілятора.

Напрямок обчислень еволюціонує від «централізованої обробки даних» на центральному процесорі до «спільної обробки» на CPU і GPU. Для реалізації нової обчислювальної парадигми компанія NVIDIA винайшла архітектуру паралельних обчислень CUDA, на даний момент представлену в графічних процесорах GeForce, ION, Quadro і Tesla і забезпечує необхідну базу розробникам ПЗ. Говорячи про споживчому ринку, варто зазначити, що майже всі основні програми для роботи з відео вже обладнані, або будуть оснащені підтримкою CUDA-прискорення. Область наукових досліджень з великим ентузіазмом зустріла технологію CUDA. Приміром, зараз CUDA прискорює

AMBER, програму для моделювання молекулярної динаміки, використовувану більше 60000 дослідниками в академічному середовищі і фармацевтичними компаніями по всьому світу для скорочення термінів створення лікарських препаратів.

Термін GPU вперше був застосований корпорацією Nvidia для виділення того факту, що графічний прискорювач, став потужним засобом, що може програмуватися, для вирішення великого класу обчислювальних задач, а не тільки для прискорення тривимірної графіки. Сучасні графічні процесори представляють собою масово паралельні обчислювальні пристрої з продуктивністю порядку терафлопса. Флопс – одиниця виміру кількості операцій над даними за секунду(FLOPS – floating-point operations per second). В загальному графічні прискорювачі базуються на архітектурі SIMD[2].

SIMD (Single Instruction Multiple Data) – одиночний потік команд, множинний потік даних) — це елемент класифікації згідно з таксономією Флінна для паралельних процесорів, де до багатьох елементів даних виконується одна або однакові команди[15]. Тобто на вхід отримується потік однорідних незалежних даних, паралельно обробляється і формується вихідний потік (рисунок 1.1). Програмний модуль що відповідає за дане перетворення називається обчислювальним ядром(kernel).

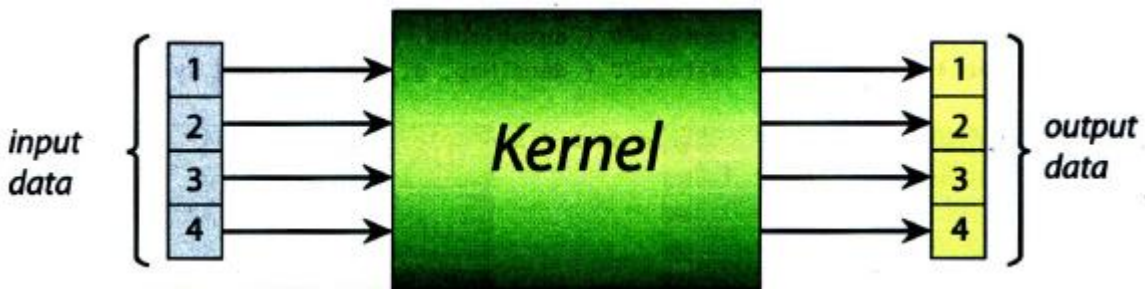

Рисунок 1.1 – SIMD архітектура

Як вже було підмічено технологія CUDA використовує CPU для послідовної частини коду і підготовкою даних для обчислень на GPU.

Паралельні частини коду будуть виконуватися на GPU одночасно на великій кількості ниток(threads) (рисунок 1.2).

Рисунок 1.2 – Виконання CUDA програми

Важливо підмітити дві основні різниці між потоками центрального процесора і нитками на графічному процесорі [2]:

– нитка GPU надзвичайно легка, її контекст мінімальний, регістри розподілені заздалегідь;

– для ефективного використання ресурсів графічного процесора програмі необхідно задіяти тисячі окремих ниток, в той час як на багатоядерному процесорі максимальна ефективність досягається при числі потоків рівній фізичні чи логічній кількості ядр.

В загальному робота ниток в архітектурі CUDA відповідає архітектурі SIMD. Проте тільки нитки в межах одної групи – варпа – виконуєтсья фізично одночасно. Нитки різних варпів можуть знаходитися на різних стадіях виконання. Така архітектура називається SIMT(Single Instruction – Multiple Threads) [2]. Управління роботою варпів заложено на апаратному рівні.

Графічний процесор відрізняється від центрального і за принципами доступу до пам'яті [7]. В GPU він пов'язаний і легко передбачуваний - якщо з

пам'яті зчитується блок інформації, то через деякий час прийде час і для сусідніх блоків. Тому організація пам'яті відрізняється від тієї, що використовується в CPU. У графічному процесорі, на відміну від універсальних процесорів, просто не потрібна кеш-пам'ять великого розміру, а для текстур потрібні лише кілька (128-256 в сучасних GPU) кілобайт. Та й сама по собі робота з пам'яттю у GPU і CPU дещо відрізняється. Так, не всі центральні процесори мають вбудовані контролери пам’яті, а у всіх GPU зазвичай є по кілька таких контролерів. Крім того, на графічних процесорах більш швидка пам'ять, і в результаті отримуємо велику пропускну здатність пам'яті, що також дуже важливо для паралельних розрахунків, що оперують величезними потоками даних. Універсальні центральні процесори використовують кеш-пам'ять для збільшення продуктивності за рахунок зниження затримок доступу до пам'яті, а GPU використовують кеш пам'ять для збільшення смуги пропускання. Графічні процесори обходять проблему доступу до пам'яті за допомогою одночасного виконання тисяч потоків: в той час, коли один з потоків очікує даних з пам'яті, інші потоки можуть виконувати обчислення без очікування і затримок, якщо це не вказано явно за допомого спеціальних деректив. Також варто підмітити що на центральному процесорі кожне ядро підтримує по 1-2 потоку, при цьому перемикання з одного на інше процесору коштує сотні тактів. В той же час графічні процесори підтримують до 1024(в найновіших версіях графічних прискорювачів 2048) потоків на кожен мультипроцесор, яких десятки. І перемикання між потоками відбувається всього лише за один такт.

Кожен потік має доступ до наведених нижче типів пам'яті [13].

– Глобальна пам'ять - найбільший обсяг пам'яті, доступний для всіх мультипроцесорів на графічному процесорі, розмір становить від 256 мегабайт до 4 гігабайт. Володіє високою пропускною здатністю, більш 100 гігабайт / с, але дуже великими затримками в декілька сотень тактів.

– Локальна пам'ять - це невеликий обсяг пам'яті, до якого має доступ тільки один потоковий процесор. Вона досить повільна - така ж, як і глобальна.

– Роздільна пам'ять - це 16-кілобайтний блок пам'яті із загальним доступом для всіх потокових процесорів в мультипроцесорі. Вона забезпечує взаємодію потоків, управляється розробником безпосередньо і має низькі затримки. Переваги роздільної пам'яті: використання у вигляді керованого програмістом кеша першого рівня, зниження затримок при доступі виконавчих блоків до даних, скорочення кількості звернень до глобальної пам'яті.

– Пам'ять констант - область пам'яті об'ємом 64 кілобайт, доступна тільки для читання мультипроцесорам. Вона кешується по 8 кілобайт на кожен мультипроцесор. Досить повільна - затримка в кілька сотень тактів при відсутності потрібних даних в кеші.

В результаті всіх описаних вище відмінностей, теоретична продуктивність графічних процесорів значно перевершує продуктивність CPU. Компанія Nvidia наводить графік зростання продуктивності CPU і GPU за останні кілька років [14], наведений на рисунку 1.3.

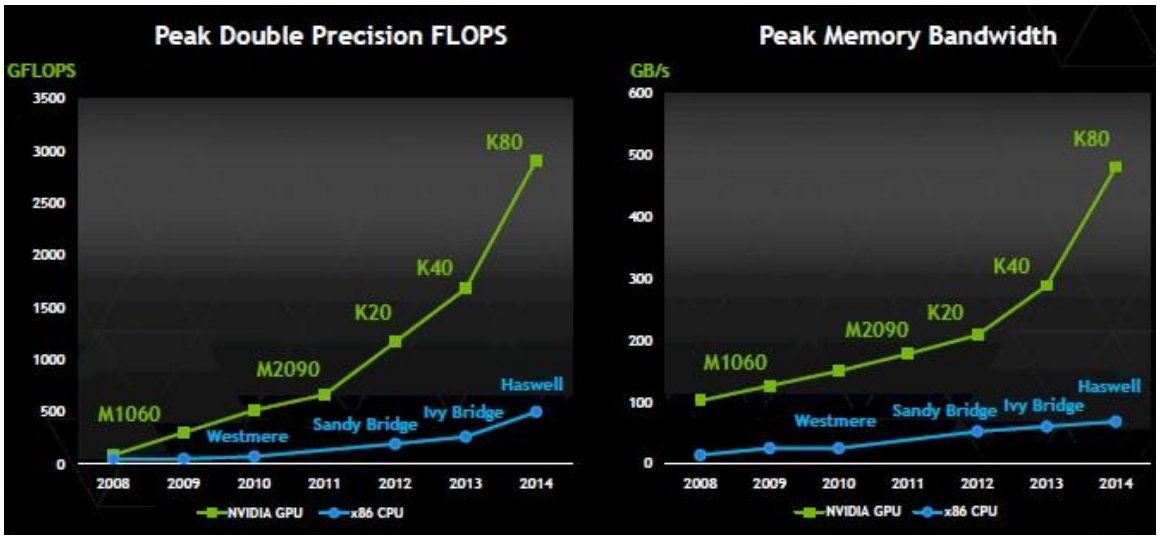

Рисунок 1.3 – Графік зростання продуктивності CPU і Nvidia GPU

На фінансовому ринку компанії Numerix і CompatibL анонсували підтримку CUDA в новому додатку аналізу ризику контрагентів і досягли прискорення роботи в 18 разів[6]. Numerix використовується майже 400

фінансовими інститутами. Показником зростання застосування CUDA є також зростання використання графічних процесорів Tesla в GPU обчисленнях. На даний момент більше 700 GPU кластерів встановлені по всьому світу в компаніях зі списку Fortune 500, таких як Schlumberger і Chevron в енергетичному секторі, а також BNP Paribas в секторі банківських послуг. Завдяки нещодавно випущеним системам Microsoft Windows 10 і Apple Snow Leopard, обчислення на GPU займуть свої позиції в секторі масових рішень. У цих нових операційних системах GPU постане не тільки графічним процесором, але також і універсальним процесором для паралельних обчислень, що працюють з будь-яким додатком.

Платформа паралельних обчислень CUDA забезпечує набір розширень для мов C і C++, що дозволяють висловлювати як паралелізм даних, так і паралелізм завдань на рівні дрібних і великих структурних одиниць. Програміст може вибрати засоби розробки: мови високого рівня, такі як C, C++, Fortran або ж відкриті стандарти, такі як директиви OpenACC. Платформа паралельних обчислень CUDA використовується на сьогоднішній день в тисячах GPU-прискорених додатків і тисячах опублікованих наукових статтях. Для мови високого рівня C# був розроблений фреймворк CUDAfy .Net, який дає змогу розробляти програмне забезпечення з великим показником продуктивності і використанням технології Nvidia CUDA для Microsoft .Net framework[11].

1.5 Висновки до розділу 1

В першому розділі обґрунтовано актуальність обраної теми, зроблено постановку мети й завдань досліджень. Було проаналізовано існуючі методи та підходи до кластеризації даних. Описано переваги обчислень на графічних процесорах в порівняні з центральним процесором.

На основі розглянутої літератури було вирішено обрати для подальшої розробки неієрархічні методи k-means та c-means. Даний вибір обґрунтовується ітераційною структурою даних методів, і обчислення для кожного об’єкта

можна проводити незалежно від інших, що є основою масово паралельних обчислень на графічних процесорах.

2 ПЛАНУВАННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ ШВИДКОДІЇ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

2.1 Формальна постановка задачі кластеризації

Дано набір даних з наступними властивостями [1]:

– кожний екземпляр даних виражається однаковою кількістю параметрів з чітким числовим значення;

– клас для кожного конкретного екземпляру даних до початку дослідження невідомий.

Потрібно знайти:

– спосіб порівняння даних між собою(міру схожості);

– спосіб кластеризації;

– розбиття даних на кластери.

Формально задача кластеризації описується наступним чином. Дано множину об’єктів даних I, кожен з яких представлений набором атрибутів. Потрібно побудувати множину кластерів C і відображення F множини I на множину C, F: I -> C. Відображення F задає модель даних, що являється власне вирішенням задачі[1].

Множина I визначається наступним чином

$$I = \{i_1, i_2, \ldots, i_j, \ldots, i_n\}, \tag{2.1}$$

де i_j – об’єкт, що досліджується.

Кожен із об’єктів характеризується набором параметрів:

$$i_j = \{x_1, x_2, \ldots, x_h, \ldots, x_m\} \tag{2.2}$$

Задача кластеризації полягає в побудові множини:

$$C = \{c_1, c_2, \ldots, c_k, \ldots, x_g\}, \tag{2.3}$$

де c_k – кластер, який має в собі похожі один на одного об’єкти з множини I:

$$c_k = \{i_l, i_p \mid i_l \in I, i_p \in I, d(i_l, i_p) < \sigma\}, \tag{2.4}$$

де σ – величина, яка визначає міру близькості для включення об’єктів в один кластер, $d(i_l, i_p)$ – відповідно міра близькості.

Метрики, за допомогою яких можна рахувати міру близькості [1]:

– евклідова метрика, де відстань обчислюється наступним чином:

$$d(x_i, x_j) = \sqrt{\sum_{t=1}^{m} (x_{it} - x_{jt})^2} \tag{2.5}$$

– метрика Хеммінга – це відстань є просто середнім різниці за координатами. У більшості випадків дана міра відстані приводить до таких же результатів, як і для звичайного відстані Евкліда, проте для неї вплив окремих великих різниць (викидів) зменшується (так як вони не зводяться в квадрат). Відстань за Хеммінгом обчислюється по формулі:

$$d(x_i, x_j) = \sum_{t=1}^{m} |x_{it} - x_{jt}| \tag{2.6}$$

– метрика Чебишева. Дана метрика може виявитися корисною, коли бажають визначити наскільки два об'єкти є "різними", якщо вони розрізняються

за якоюсь однією координатою (будь-яким одним виміром). Відстань Чебишева обчислюється за формулою:

$$d_{\infty}(x_i, x_j) = \max_{l<i<m} |x_{it} - x_{jt}| \quad (2.7)$$

2.2 Застосування паралельних обрахунків для обраних методів кластерного аналізу даних

2.2.1 Метод k-середніх

Сьогодні використовуються різні модифікації даного алгоритму, але кожен із них має два основних недоліки: необхідність знати кількість кластерів перед виконанням алгоритму і оптимальність отриманих рішення не є гарантованою. Тому пропонована модифікація буде направлена на прискорення роботи алгоритму за допомогою використання паралельних обчислень.

Розроблений алгоритм представляє собою ітераційну процедуру:

Крок 1 (CPU). Випадковим чином обрати центри кластерів із елементів вхідних даних, встановити номер ітерації $l = 0$.

Крок 2 (CPU). Підключити GPU модуль, обрахувати конфігурацію запуску для функції-ядра:

– кількість ниток(так прийнято називати один потік виконання обрахунків) в блоці:

$$numT = \frac{maxT}{2}, \quad (2.8)$$

де $maxT$ – максимальна можлива кількість ниток в блоці для підключеного графічного процесора;

– кількість блоків:

$$numB = \left\lceil \frac{num0}{numT} \right\rceil, \tag{2.9}$$

де $num0$ – загальна кількість об’єктів, що досліджується, $numT$ – визначена кількість ниток в блоці.

В оригінальному методі наступні два кроки полягають у обчисленні матриці розбиття за формулою 1.6,та у визначенні нових центрів кластерів за формулою 1.7. В нашій модифікації суть даних кроків зберігається, але присутня специфічна реалізація в зв’язку з застосуванням паралельних обчислень.

Крок 3 (GPU). Викликати кернел-функцію з конфігурацією запуску, обчисленою на кроці 2, для опрацювання даних на графічному процесорі. Кернел-функція виконується паралельно на великій кількості ниток, при чому кожна нитка опрацьовує один об’єкт із вхідної послідовності, що досліджується:

1) в спільний пам’яті виділити масиви пам’яті для збереження накопичувальної суми характеристик об’єктів(*SharedSums*), що належать для певного кластера та відповідно кількості об’єктів в кожному кластері(*SharedCounts*);

2) синхронізувати нитки за допомогою бар’єрної синхронізації;

3) знайти оптимальний центроїд (кластер) для об’єкта, що досліджується;

4) додати характеристики об’єкта до масиву *SharedSums* відповідно до оптимального центроїда;

5) інкрементувати кількість об’єктів(*SharedCounts*); в оптимальному кластері на 1;

6) синхронізувати нитки за допомогою бар’єрної синхронізації;

7) копіювати дані масивів з спільної пам’яті в результуючі масиви в глобальній пам’яті;

Таким чином на графічному процесорі ми паралельно обчислили одразу потрібні нам: $SharedSums_i = \sum_{j=1}^{d} u_{ij} m_j$ та $SharedCounts_i = \sum_{j=1}^{d} u_{ij} m_j$ з формули 1.7 для кожного кластера, де $1 \leq i \leq c$

Крок 4 (CPU). Визначити нові центри кластерів:

$$c_l^{(i)} = \frac{SharedSums_i}{SharedCounts_i}, 1 \le i \le c, \quad (2.10)$$

де c – кількість кластерів;

Крок 5. Перевірити наскільки змістилися центри кластерів. Якщо вони змістилися менше ніж на обрану точність δ – завершити процес, якщо ні – перейти до кроку 2 з номером ітерації *l = l + 1*.

Набір обмежень залишається таким же як і в оригінальному алгоритмі: $u_{ij}^{(l)} \{0,1\}; \sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d.$

Розглянем складність альтернативного варіанту оригінального алгоритму описаного в розділі 1.4, на базі якого був розроблений модифікований алгоритм. Для кожного з *N* вхідних об'єктів потрібно обчислити відстань до кожного центроїда *K* і вибрати найкоротшу відстань. Це відома проблема пошуку найближчого сусіда і лінійний пошук дає нам складність *O(KN*). Оскільки в алгоритмі використовується метрика Евкліда і множина вхідних точок R^D, де *D* – це кількість вимірів(характеристик) об'єкта, що досліджується, то складність уже становить *O(DKN)*. Припускаючи, що збіжність даного алгоритму становить I ітерацій, отримуємо складність *O(DKNI)*.

В цьому розділі описано паралельний алгоритм для графічного процесор, але якщо припустити, що він буде реалізований для паралельної роботи на центральному процесорі, то ми отримаємо наступну складність. Для початку розділяємо N вхідних об'єктів на P рівних частин, що буде відповідати кількості паралельних потоків. Складність присвоєння точок кластерам становить *O(D K)*, але додається затрата на обчислення часткової суми та кількості об'єктів в кластері *O(D + 1)*. Зважаючи, що в кожному потоці P, опрацьовуються N/P об'єктів, загальна складність для одної паралельної ітерації становить *O(N/P (DK + D + 1))* до бар'єру, та ще *O(K D)* для перерахунку центроїдів. Припускаючи, що збіжність даного алгоритму становить I ітерацій, отримуємо:

$$O\left(I\left(\frac{N}{P}(DK+D+1)+DK\right)\right)=$$

$$O\left(\frac{I(N(DK+D+1)+DKP)}{P}\right)=$$

$$O\left(\frac{I(NDK+ND+N+DKP)}{P}\right)=$$

$$O\left(\frac{DI\left(NK+N+\frac{N}{D}+KP\right)}{P}\right)=$$

$$O\left(\frac{DI\left(N(K+1+\frac{1}{D})+KP\right)}{P}\right)$$

Прискорення даного алгоритму, обчислюємо розділивши складність оригінального алгоритму на складність паралельного:

$$\frac{O(DKNI)}{O\left(\frac{DI\left(N\left(K+1+\frac{1}{D}\right)+KP\right)}{P}\right)}=$$

$$\frac{KNP}{N\left(K+1+\frac{1}{D}\right)+KP}=$$

$$\frac{KNP}{N\left(K+1+\frac{1}{D}\right)+KP}=$$

$$\frac{KP}{\left(K+1+\frac{1}{D}\right)+\frac{KP}{N}}$$

Тоді для великих значень N припускаємо, що KP / N прямує до 0 і для таких значень N, ми отримуємо прискорення:

$$\frac{KP}{\left(K+1+\frac{1}{D}\right)}$$

Тепер розглянемо варіант з власне представленим методом для масово паралельних обчислень. Кожна паралельна нитка T в блоці B буде мати обчислювальну складність *O(K(D+1) + D + 1 + KD + K)*:

– *O(K(D+1))* – на знаходження оптимального центроїда, Крок 3.3;

– *O(D)* – збільшення накопичувальної суми $SharedSums_i$, Крок 3.4;

– *O(1)* –інкремент $SharedCounts_i$, Крок 3.5;

– *O(KD + K)* – на копіювання отриманого результату, Крок 3.7. Враховуючи те, що кількість блоків $B = N / T$ і припускаючи, що збіжність даного алгоритму становить I ітерацій, складність алгоритму становить:

$$O\left(\frac{IN(2K+\ (2K+1)D+1)}{T}\right)$$

Відповідно прискорення алгоритму буде становити:

$$\frac{O(DKNI)}{O\left(\frac{IN(2K+\ (2K+1)D+1)}{T}\right)} = \frac{DKT}{2K+\ (2K+1)D+1}$$

Припустимо, що P = 8, K = 10, D = 2, T = 512 і N досить велике, то отримане теоретичне прискорення паралельного алгоритму:

$$\frac{KP}{\left(K+1+\frac{1}{D}\right)} = \frac{10*8}{\left(10+1+\frac{1}{2}\right)} = 6{,}96$$

І алгоритму для графічного процесору:

$$\frac{DKT}{2K + (2K+1)D + 1} = \frac{2 * 10 * 512}{2 * 10 + (2 * 10 + 1)2 + 1} = 162$$

2.2.2 Метод с-середніх

Модифікація направлена на прискорення роботи алгоритму за допомогою використання паралельних обчислень.

Розроблений алгоритм представляє собою ітераційну процедуру:

Крок 1 (CPU). Випадковим чином обрати центри кластерів із елементів вхідних даних, встановити номер ітерації $l = 0$.

Крок 2 (CPU). Підключити GPU модуль, обрахувати конфігурацію запуску для функції-ядра:

– Кількість ниток(так прийнято називати один потік виконання обрахунків) в блоці:

$$numT = \frac{maxT}{2}, \tag{2.11}$$

де $maxT$ – максимальна можлива кількість ниток в блоці для підключеного графічного процесора;

– Кількість блоків:

$$numB = \left\lceil \frac{num0}{numT} \right\rceil, \tag{2.12}$$

де $num0$ – загальна кількість об’єктів, що досліджується, $numT$ – визначена кількість ниток в блоці.

Крок 3 (GPU). Викликати кернел-функцію з конфігурацією запуску, обчисленою на кроці 2, для опрацювання даних на графічному процесорі. Кернел-функція виконується паралельно на великій кількості ниток, при чому кожна нитка опрацьовує один об’єкт із вхідної послідовності, що досліджується:

1) в спільний пам’яті виділити масиви пам’яті для збереження накопичувальної суми для чисельника(*SharedNumerator*) та знаменника (*SharedDenominator*), які в подальшому будуть застосовуватися до формули 1.15;

2) синхронізувати нитки за допомогою бар’єрної синхронізації;

3) обчислити квадрати відстаней від центрів усіх кластерів до об’єкта, що досліджується;

4) синхронізувати нитки за допомогою бар’єрної синхронізації;

5) оновити частину матрицю розбиття, що відповідає об’єкту, що досліджується, на основі відстаней обчислених в пункті 3 по формулі 1.10;

6) синхронізувати нитки за допомогою бар’єрної синхронізації;

7) збільшити чисельник(*SharedNumerator*) та знаменник (*SharedDenominator*), відповідно до обчисленої матриці розбиття;

8) синхронізувати нитки за допомогою бар’єрної синхронізації;

9) копіювати дані масивів з спільної пам’яті в результуючі масиви в глобальній пам’яті;

Коментар: таким чином на графічному процесорі ми паралельно обчислили одразу потрібні нам: $SharedNumerator_i = \sum_{j=1}^{d}(u_{ij})^w m_j$ та $SharedDenominator_i = \sum_{j=1}^{d}(u_{ij})^w$ з формули 1.15 для кожного кластера, де $1 \leq i \leq c$

Крок 4 (CPU). Визначити нові центри кластерів:

$$c_l^{(i)} = \frac{SharedNumerator_i}{SharedDenominator_i}, 1 \leq i \leq c, \quad (2.13)$$

де c – кількість кластерів;

Крок 5. Перевірити наскільки змістилися центри кластерів. Якщо вони змістилися менше ніж на обрану точність δ – завершити процес, якщо ні – перейти до кроку 2 з номером ітерації $l = l + 1$.

Набір обмежень залишається таким же як і в оригінальному алгоритмі: $u_{ij}^{(l)} \{0,1\}; \sum_{j=1}^{c} u_{ij} = 1; 0 < \sum_{j=1}^{d} u_{ij} < d.$

Розглянем складність оригінального алгоритму описаного в розділі 1.5, на базі якого був розроблений модифікований алгоритм. Для кожного з N вхідних об'єктів потрібно обчислити відстань до кожного центроїда K – $O(K\ N)$. Після цього нам потрібно обчислити матрицю розбиття для всіх кластерів, що дасть нам складність $O(K^2\ N)$. Оскільки в алгоритмі використовується метрика Евкліда і множина вхідних точок R^D, де D – це кількість вимірів(характеристик) об'єкта, що досліджується, то складність уже становить $O(DK^2N)$. Припускаючи, що збіжність даного алгоритму становить I ітерацій, отримуємо складність $O(DK^2NI)$.

Тепер обчислимо складність представленого алгоритму для масово паралельних обчислень. Кожна паралельна нитка T в блоці B буде мати обчислювальну складність $O(K(D + 1) + K(K + 1) + 2KD + 2KD)$:

– $O(K(D + 1))$ – на обчислення відстаней, Крок 3.3;
– $O(K(K + 2))$ – на оновлення матриці розбиття, Крок 3.5;
– $O(2KD)$ – на збільшення знаменника і чисельника, Крок 3.7;
– $O(2KD)$ – на копіювання отриманого результату, Крок 3.9.

Враховуючи те, що кількість блоків $B = N / T$ і припускаючи, що збіжність даного алгоритму становить I ітерацій, складність алгоритму становить:

$$O\left(\frac{INK(3 + 5D + K)}{T}\right)$$

Відповідно прискорення методу буде становити:

$$\frac{O(DK^2NI)}{O\left(\frac{INK(3+5D+K)}{T}\right)} = \frac{DTK}{3+5D+K}$$

Припустимо, що K = 10, D = 2, T = 512 і N досить велике, то отримане теоретичне прискорення модифікованого алгоритму:

$$\frac{DTK}{3+5D+K} = \frac{2*10*512}{3+10+10} = 445$$

2.3 Висновки до розділу 2

В даному розділі проведено аналіз методів k-середніх та c- середніх на предмет можливості розпаралелювання обчислень. Виявлено, що в цих методах операції обчислення матриці приналежності та центрів кластерів можна виконати паралельно. Наведено модифіковані алгоритми кластеризації для масово паралельних обчислень на графічних процесорах Nvidia.

Математичний аналіз складності алгоритмів показує велике теоретичне прискорення для розроблених модифікацій методів в порівнянні з виконанням на центральному процесорі, що може досягати більше ста для модифікованого методу k-середніх, та сотень для модифікованого методу c- середніх.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ШВИДКОДІЇ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ ДАНИХ НА ОСНОВІ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

3.1 Розроблення прототипу програмного продукту для проведення експериментальних досліджень

3.1.1 Архітектура програмного забезпечення

Вимога ефективності процесу розробки призвела до вибору платформи .NET на базі операційної системи Windows, і відповідно високорівневої мови програмування C#. Такий вибір дозволяє використовувати розвинене середовище розробки Microsoft Visual Studio та потужний функціонал уже існуючих бібліотек та фреймворків.

Для створення програмного продукту було використано:

– для графічного інтерфейсу користувача було обрано WindowsForms через простоту реалізації і зручність отриманого програмного продукту;

– для роботи з графічним процесором було обрано технологію CUDA і фреймворк CUDAfy.Net, який реалізує зручні інтерфейси для реалізації паралельних обчислень на графічних процесорах Nvidia;

– для побудови графіків, було обрано бібліотеку OxyPlot, що надає можливість інтерактивної роботи для користувача: масштабування, зміни кольорів елементів, опис легенди та інше.

3.1.2 Структура класів програмного забезпечення

Діаграма класів розроблена в ході конструювання програмного забезпечення для демонстрації роботи розроблених модифікованих методів кластерного аналізу представлена на рисунку 3.1.

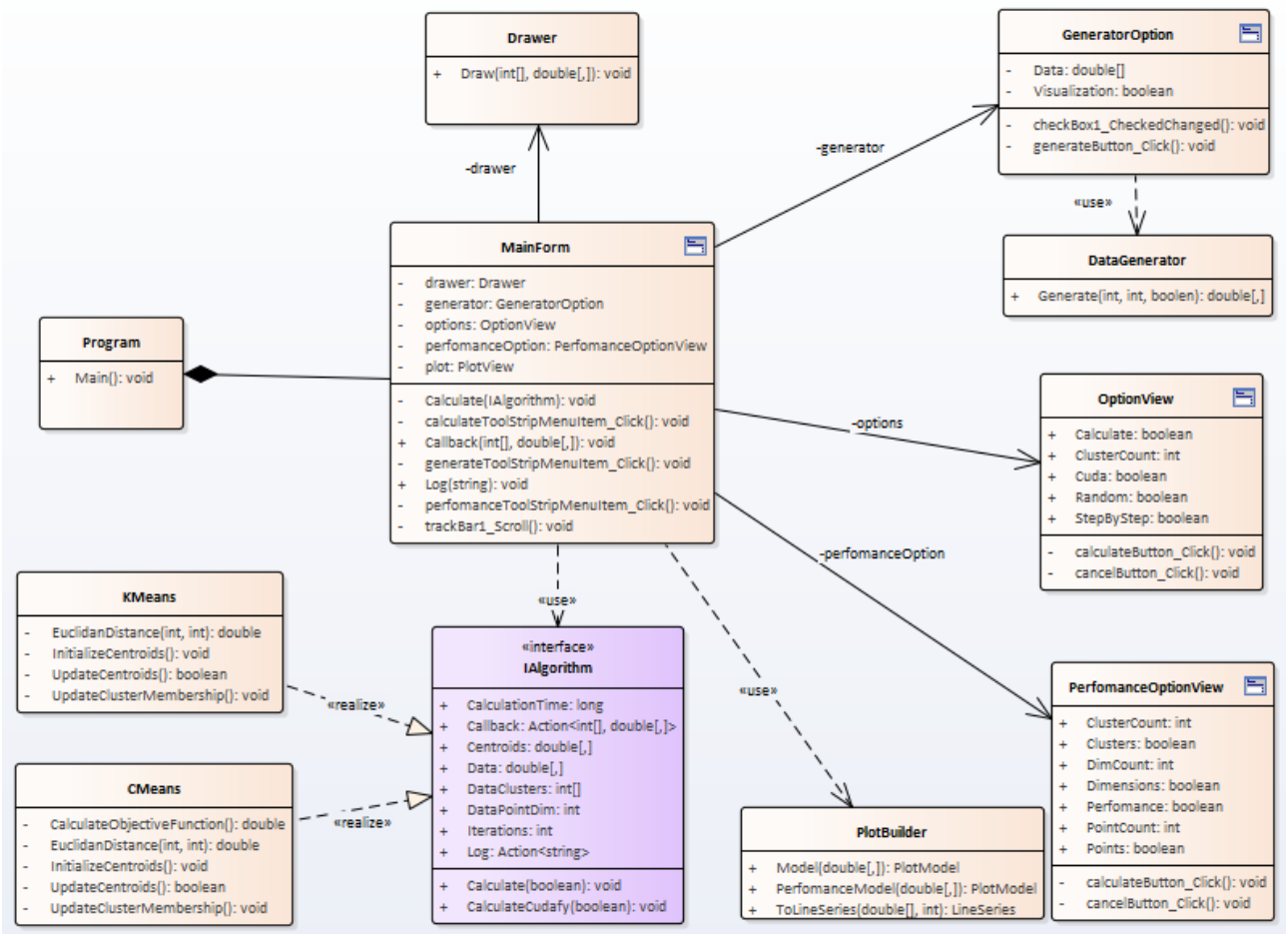

Рисунок 3.1 – Діаграма класів

Основним інтерфейсом є IAlgorithm, який повинні реалізовувати класи методів, таблиця 3.1.

Таблиця 3.1 – Опис інтерфейсу «IAlgorithm.cs»

Властивості	
Double[,] Data	Вхідні дані
Int32[] DataClusters	Матриця розбиття
Double[,] Centroids	Центри обчислених кластерів
Int32 DataPointDim	Кількість вимірів вхідних об’єктів
Int32 Iterations	Кількість ітерації, затрачених на обчислення

Закінчення таблиці 3.1

Int64 CalculationTime	Кількість часу, в мілісекундах, затрачених на обчислення
Action<Int32[], Double[,]> Callback	Делегат для повернення покрокового результату матриці розбиття і центрів кластерів
Action<String> Log	Делегат для логування
Методи	
void Calculate(bool perfomance)	Метод обчислення методу на CPU. Коли параметр істинний відбувається заміри швидкодії.
void CalculateCudafy(bool perfomance)	Метод обчислення методу на GPU. Коли параметр істинний відбувається заміри швидкодії.

Для побудови графіків було розроблений статичний клас PlotBuilder, таблиця 3.2.

Таблиця 3.2 – Опис методів статичного класу «PlotBuilder.cs»

Метод	Модифікатор	Параметри	Опис
Model	static public	(double[]): PlotModel	Повертає графік для початкового екрану програми
PerfomanceModel	static public	(pd: double[]): PlotModel	Повертає графік результатів прогнозу
ToLineSeries	static private	(series: double[], startX: int): LineSeries	Перетворює масив даних в структуру даних для моделі графіка

Для генерації випадкових даних створено клас «DataGenerator.cs» який на основі переданих параметрів генерує масив з n-вимірних об’єктів, характеристиками якого є числові значення. Якщо згенеровано двовимірні дані, то вони відображаються на екрані, за допомогою класу «Drawer.cs».

Основними класами, які реалізують обрані методи кластеризації, є «KMeans.cs» та «CMeans.cs», що реалізують інтерфейс «IAlgorithm.cs».

Дві допоміжні форми, «OptionView.cs» та «PerfomanceOptionView.cs» відповідають за вибір параметрів для кластеризації та вимірювання швидкодії розроблених методів. Детально опис параметрів наведено в таблиці 3.3 і 3.4 відповідно.

Таблиця 3.3 – Опис властивостей класу «OptionView.cs»

Властивості	
bool Calculate	Параметр, який відповідає чи були обрані параметри кластеризації
Int32 ClusterCount	Параметр, який відповідає за кількість кластерів для кластеризації
bool Cuda	Параметр, який відповідає чи використовувати графічний процесор для розрахунків
bool Random	Параметр, який відповідає чи використовувати випадкове значення seed для генерації псевдовипадкових чисел.
bool StepByStep	Параметр, який відповідає чи зберігати результати виконання методу після кожної його ітерації

Таблиця 3.4 – Опис властивостей класу «PerfomanceOptionView.cs»

Властивості	
bool Perfomance	Параметр, який відповідає чи були обрані параметри для вимірювання швидкодії

Продовження таблиці 3.4

Int32 ClusterCount	Параметр, який відповідає за кількість кластерів для кластеризації
bool Clusters	Параметр, який відповідає чи буде змінюватися кількість кластерів для вимірювання швидкодії
Int32 PointsCount	Параметр, який відповідає за кількість об’єктів для кластеризації
bool Points	Параметр, який відповідає чи буде змінюватися кількість об’єктів для заміру швидкодії
Int32 DimCount	Параметр, який відповідає за кількість вимірів об’єкту, що досліджується, для кластеризації
bool Dimensions	Параметр, який відповідає чи буде змінюватися кількість вимірів об’єкту, що досліджується, для вимірювання швидкодії

3.1.3 Керівництво користувача

Розпочати роботу з програмним продуктом можливо запустивши на виконання файл «CudaClasterization.exe», подвійним кліком мишки на ярлик з іконкою програми.

Після успішного запуску програми на екрані відкривається головне вікно – «Cuda Clasterization» (Рисунок 3.2).

Головне вікно програми містить 3 основних елемента:

- «Меню»;
- вкладки «Visualization» і «Perfomance»
- інформаційна панель.

Структура меню:

– «Data» – відповідає за роботу з вхідними та вихідними даними процесу кластеризації:

  – «Generate» – викликає нове діалогове вікно(Рисунок 3.3), для генерації даних;

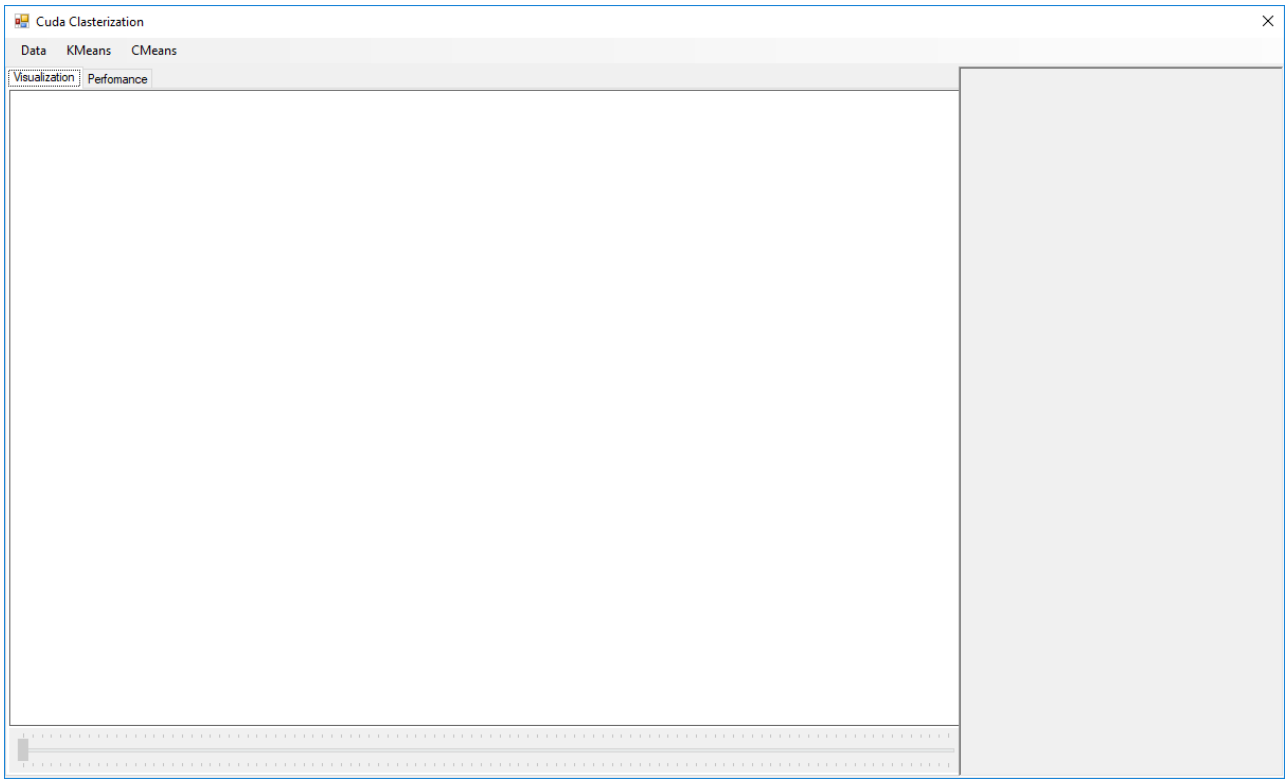

Рисунок 3.2 – Головне вікно програмного продукту

  – «Read From File» – викликає діалогове вікно, для зчитування даних з CSV файлу;
  – «Save To File» – викликає діалогове вікно, для збереження файлу з результатами кластеризації.

– «KMeans» – відповідає за кластеризацію з використання методу k-means:

  – «Calculate» – викликає діалогове вікно для налаштування параметрів кластеризації методом k-means (Рисунок 3.5);

  - «Perfomance» – викликає діалогове вікно для налаштування параметрів для виміру швидкодії методу k-means (Рисунок 3.5).
- «CMeans» – відповідає за кластеризацію з використання методу c-means:
  - «Calculate» – викликає діалогове вікно для налаштування параметрів кластеризації методом c-means (Рисунок 3.5);
  - «Perfomance» – викликає діалогове вікно для налаштування параметрів для виміру швидкодії методу c-means (Рисунок 3.5).

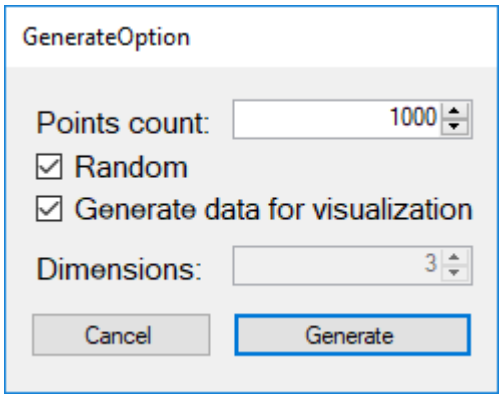

Рисунок 3.3 – Діалогове вікно для генерування даних

Як видно з рисунку 3.3 користувач має змогу вибрати яку кількість об’єктів він хоче згенерувати. При виборі прапорця «Random» користувач отримувати кожен раз нові випадкові дані при їх генерації. В іншому випадку, користувач отримуватиме одні й ті ж дані. Це зручно для тестування різних методів і для порівняння отриманих результатів. Прапорець «Generate data for visualization» відповідає за кількість вимірів згенерованих об’єктів. Якщо він вибраний, то будуть генеруватися двовимірні об’єкти. В іншому випадку користувачу надається змогу самому встановити бажану кількість вимірів, але тоді не має змоги переглянути результат візуально. При кліку на кнопку «Generate» відбувається генерація даних, і повернення до головного вікна, і якщо було згенеровано двовимірні дані вони відобразяться в вкладці «Visualization»

(Рисунок 3.4). При кліку на кнопку «Cancel» генерація даних не відбудеться. В будь якому із двох випадків обрані параметри будуть збережені.

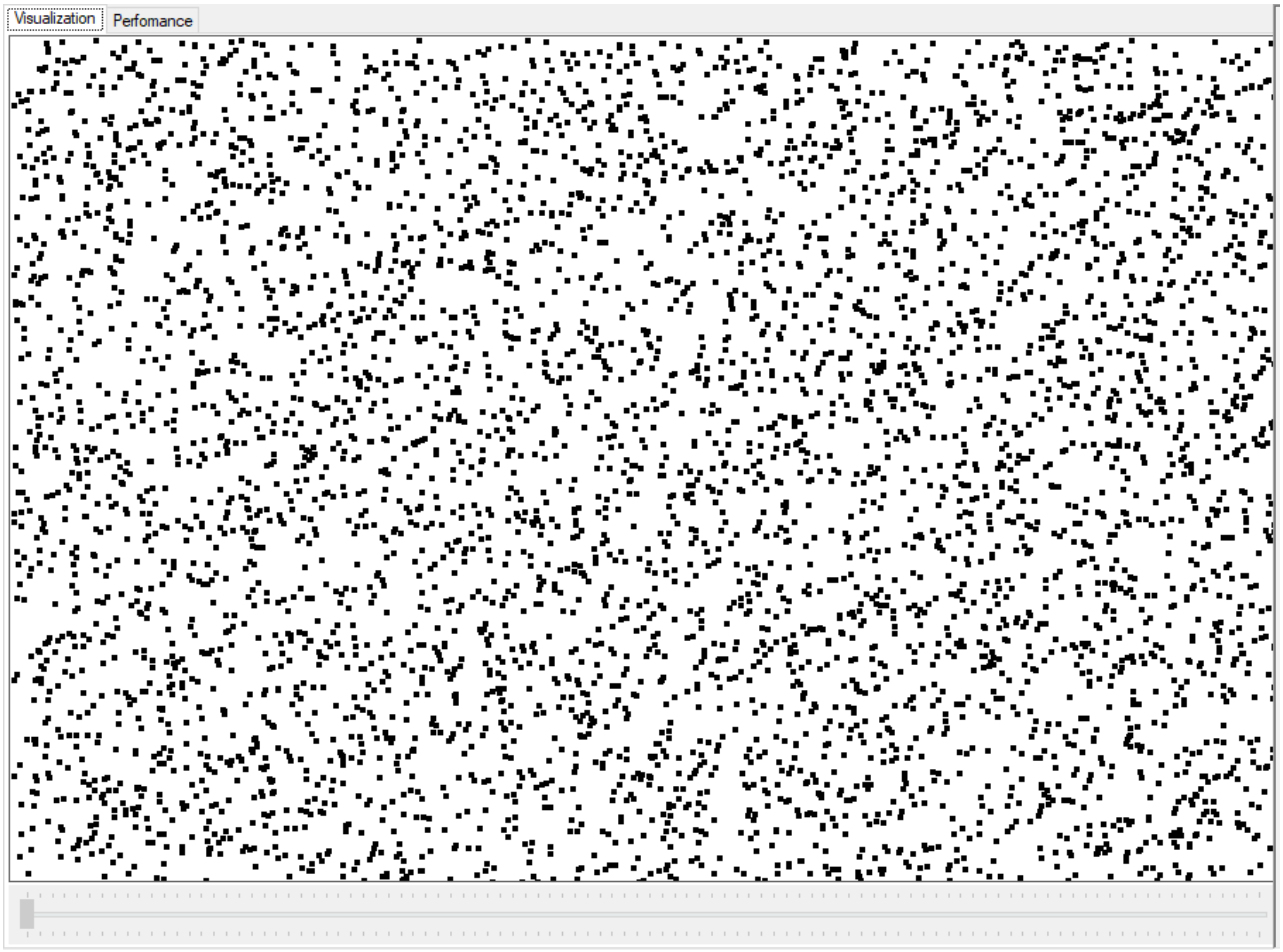

Рисунок 3.4 – Відображення 5000 згенерованих точок в вкладці «Visualization»

Для вибору параметрів кластеризації для обох методів відображається діалогове вікно «Option» (Рисунок 3.5).

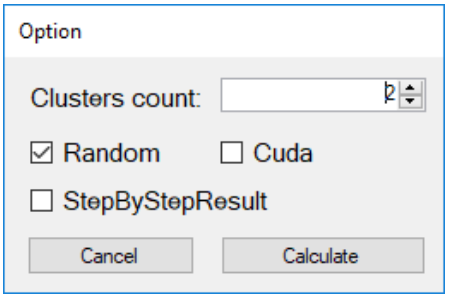

Рисунок 3.5 – Діалогове вікно для вибору параметрів методу

Як видно з рисунку 3.4 користувач повинен вибрати на яку кількість кластерів повинен розбити метод вхідні дані. При виборі прапорця «Random» користувач отримувати кожен раз нові випадкові числа, які потрібні в ході виконання методу кластеризації. В іншому випадку, користувач отримуватиме одні й ті ж числа. Це зроблено для того, щоб мати змогу перевірити і порівняти результати методу, обчисленого на центральному та графічному процесорах. Прапорець «Cuda» відповідає за використання графічного процесора в процесі розрахунків методу. При виборі прапорця «StepByStepResult» після кожної ітерації методу проміжні дані будуть збережені, і користувач буде мати змогу переглянути їх за допомогою повзунка знизу на вкладці «Visualization» (рисунок 3.7). При кліку на кнопку «Calculate» відбувається виконання методу, і повернення до головного вікна. Результат відобразиться в вкладці «Visualization» та в інформаційне вікно виведеться інформація про кількість ітерації і затрачений час (рисунок 3.6). При кліку на кнопку «Cancel» генерація даних не відбудеться. В будь якому із двох випадків обрані параметри будуть збережені.

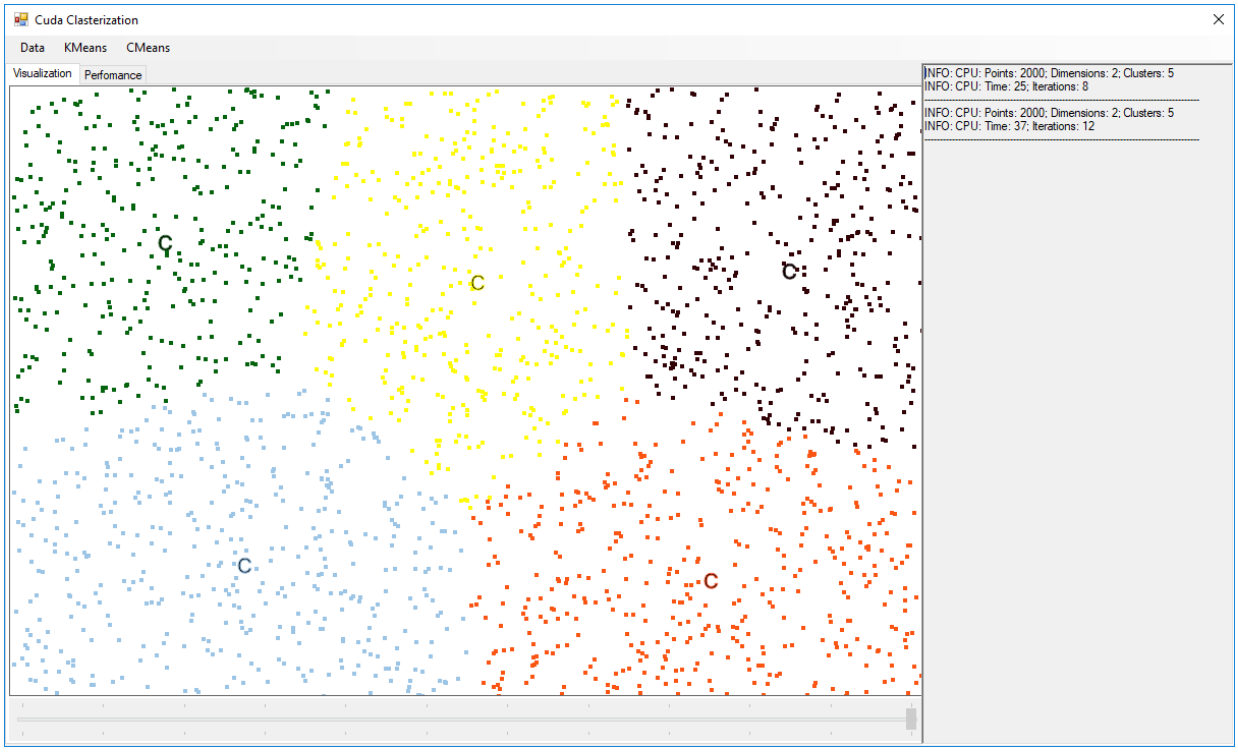

Рисунок 3.6 – Результат виконання методу для 2000 точок і 5 кластерів

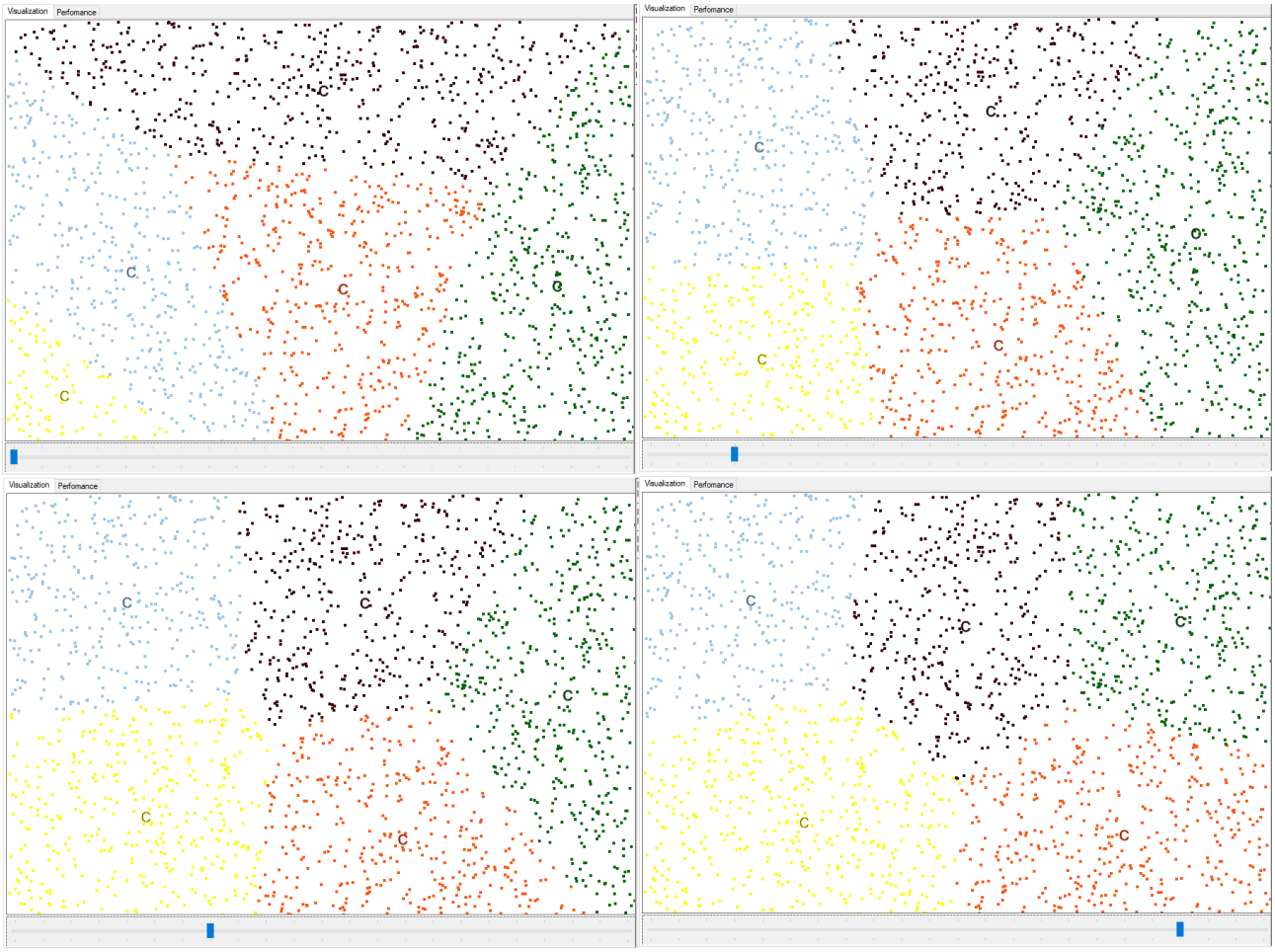

Рисунок 3.7 – Покрокові результати виконання методу

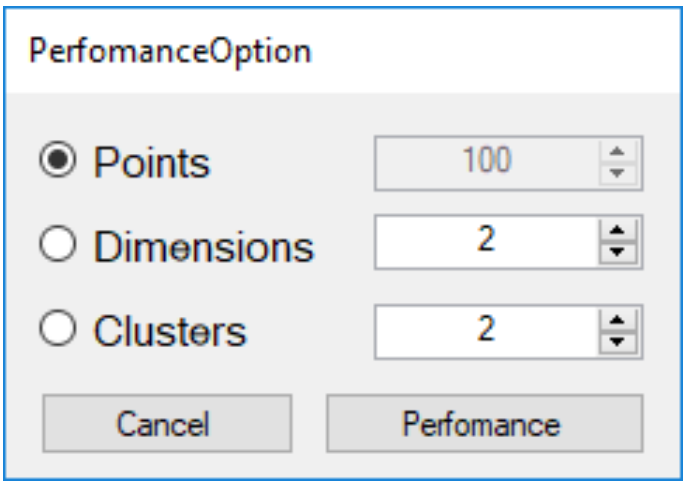

Рисунок 3.8 – Діалогове вікно для вибору параметрів заміру швидкодії

Як видно з рисунку 3.8 користувач має змогу вибрати який із параметрів буде змінюватися в процесі заміру швидкодії. Тобто для прикладу користувач вибрав полу «Points». Це означає що метод буде розраховуватися для двовимірних точок та для пошуку двох кластерів, а кількість точок з кожним запуском методу буде збільшуватися. Результат відобразиться в вкладці «Perfomance» у вигляді графіку та в інформаційне вікно виведеться інформація про кількість ітерації і затрачений час на кожен запуск методу.(Рисунок 3.9).

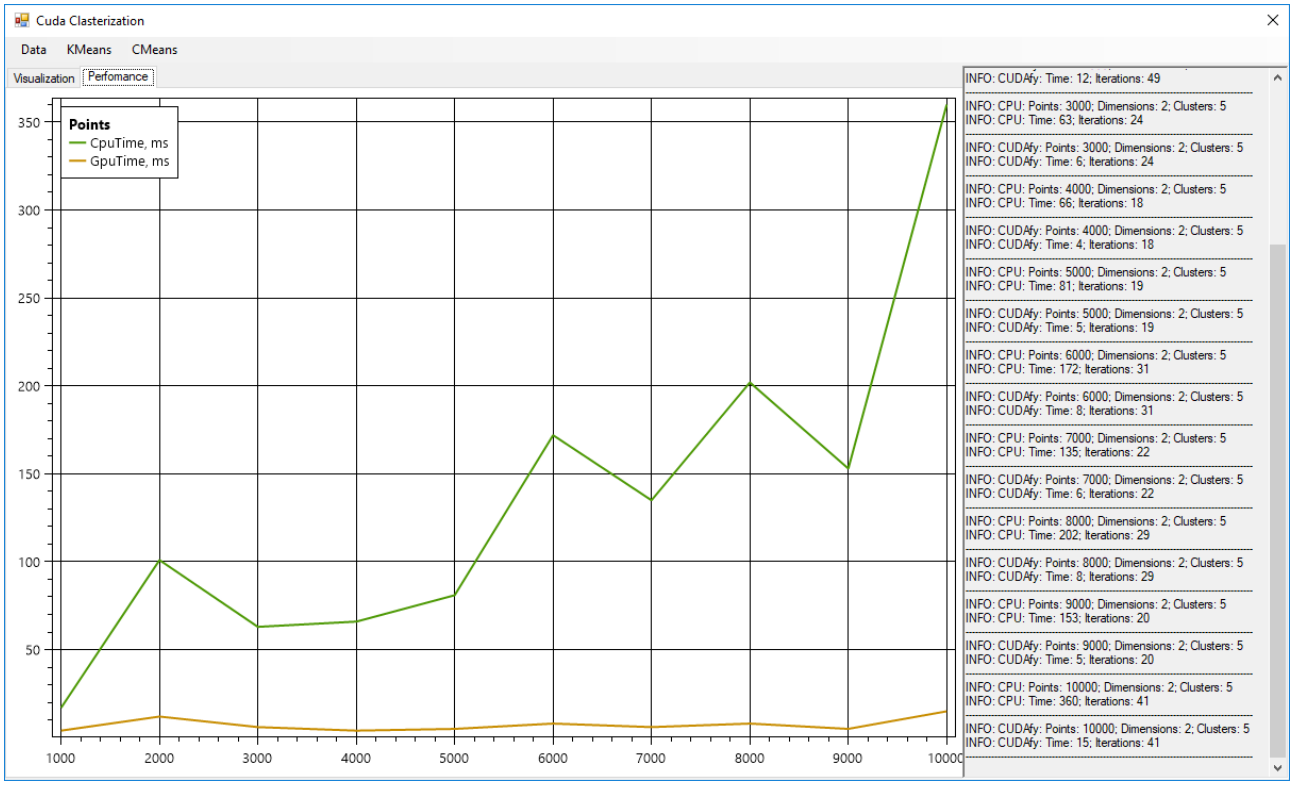

Рисунок 3.9 – Результат заміру швидкодії методу k-means

3.2 Результати експериментальних досліджень

Для тестування використовувався персональний комп’ютер:

- 4 ядерний CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz;
- GPU: Nvidia GeForce GTX 960 2Gb;
- RAM: 8Gb Kingston DDR3 SDRAM.

3.2.1 Порівняння швидкодії для методу k-means

Для порівняння виконання методів було згенеровано 10000 двовимірних точок. Для того щоб уникнути генерації різних псевдовипадкових чисел в оригінальному і модифікованому методах був встановлений однаковий seed для генератора випадкових чисел.

Так на рисунку 3.10 спостерігаємо розбиття вхідних точок на 10 кластерів, що відображаються різними кольорами, а також в інформаційній в правій частині вікна, що обидва методи виконалися за однакову кількість ітерації – 93, але виконання на CPU зайняло 1460 мс, а виконання на GPU – 27 мс. Це показує нам, що навіть на невеликому наборі простих даних модифікований метод дає нам чималий виграш в його швидкодії.

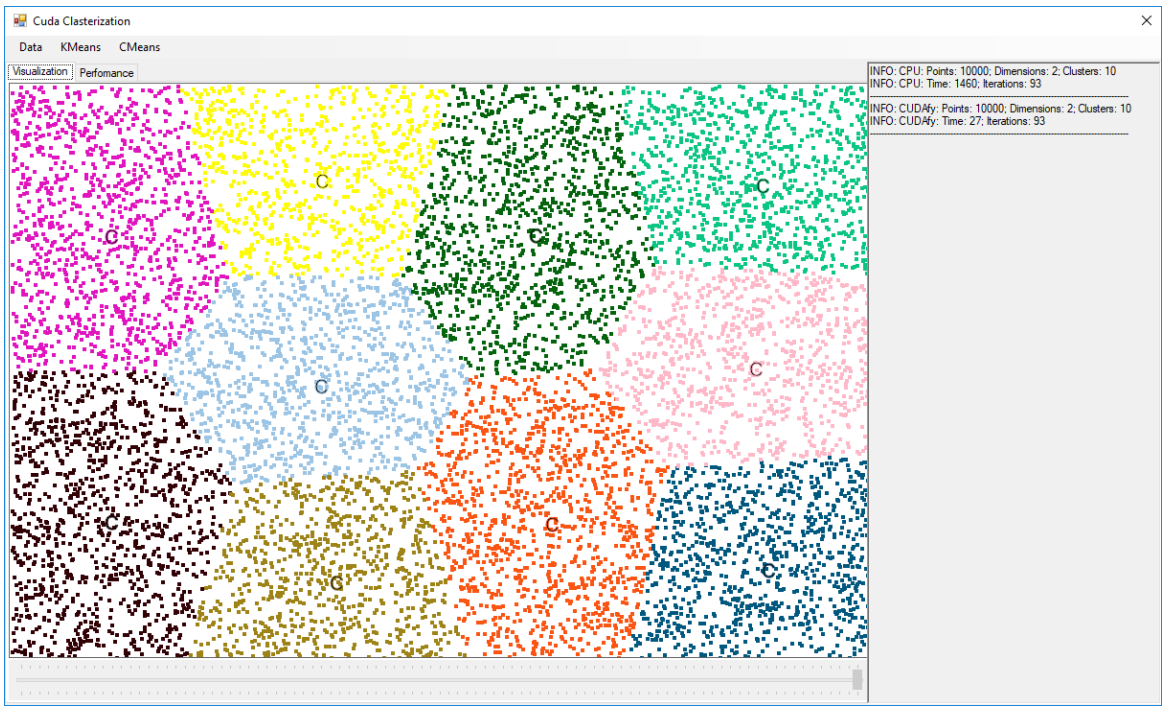

Рисунок 3.10 – Результати роботи методу k-means для 10 000 точок

В таблиці 3.5 та на рисунку 3.11 наведено порівняльну характеристику часу кластеризації для 16 кластерів оригінального і модифікованого методів для двовимірних точок кількістю від 3000 до 10000 з кроком в 1000.

Таблиця 3.5 – Порівняння швидкодії роботи методу на CPU і CPU+GPU

Кількість точок	К-сть ітерацій	CPU, мс	CPU + GPU, мс	Коефіцієнт прискорення
3000	42	306	11	27,8
4000	61	593	15	39,5
5000	57	568	15	37,9
6000	61	890	17	52,4
7000	60	1014	16	63,4
8000	81	1593	25	63,7
9000	65	1414	22	64,3
10 000	79	1910	29	65,9

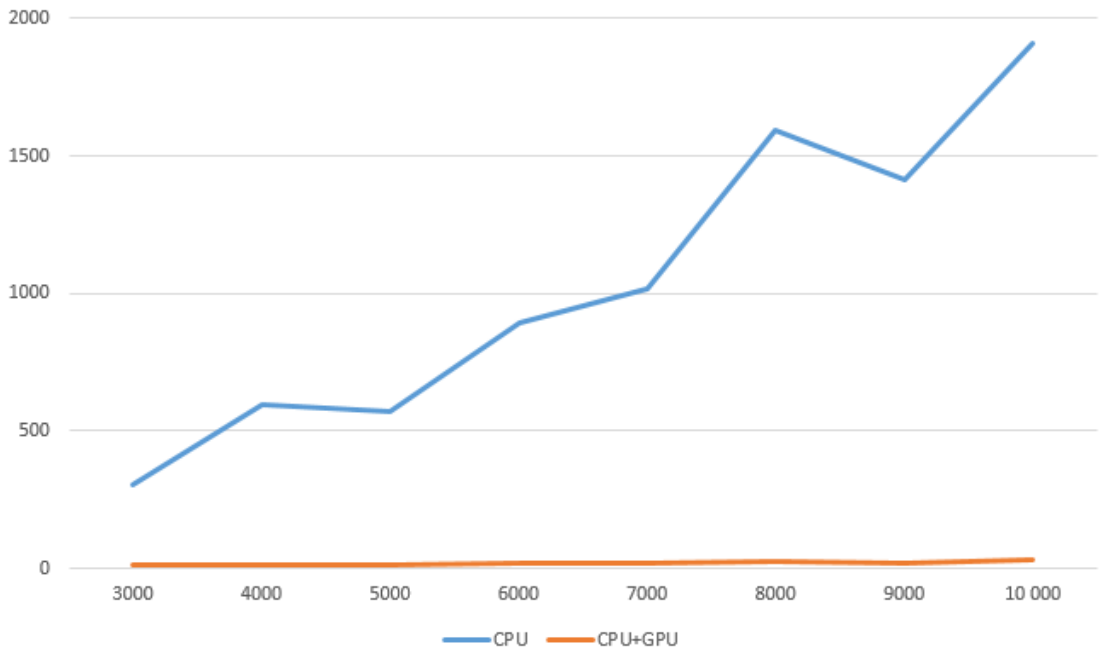

Рисунок 3.11 – Порівняння швидкодії роботи методу на CPU і CPU+GPU

Таким чином бачимо, що при зростанні кількості вхідних об’єктів зростає коефіцієнт ефективності, і для надвеликого масиву вхідних даних можливість

досягти близького до теоретичного прискорення, обчисленого в другому розділі, цілком реальна.

3.2.2 Порівняння швидкодії для методу c-means

В таблиці 3.6 та на рисунку 3.12 наведено порівняльну характеристику часу кластеризації для 10 кластерів оригінального і модифікованого методів для двовимірних точок кількістю від 3000 до 10000 з кроком в 1000.

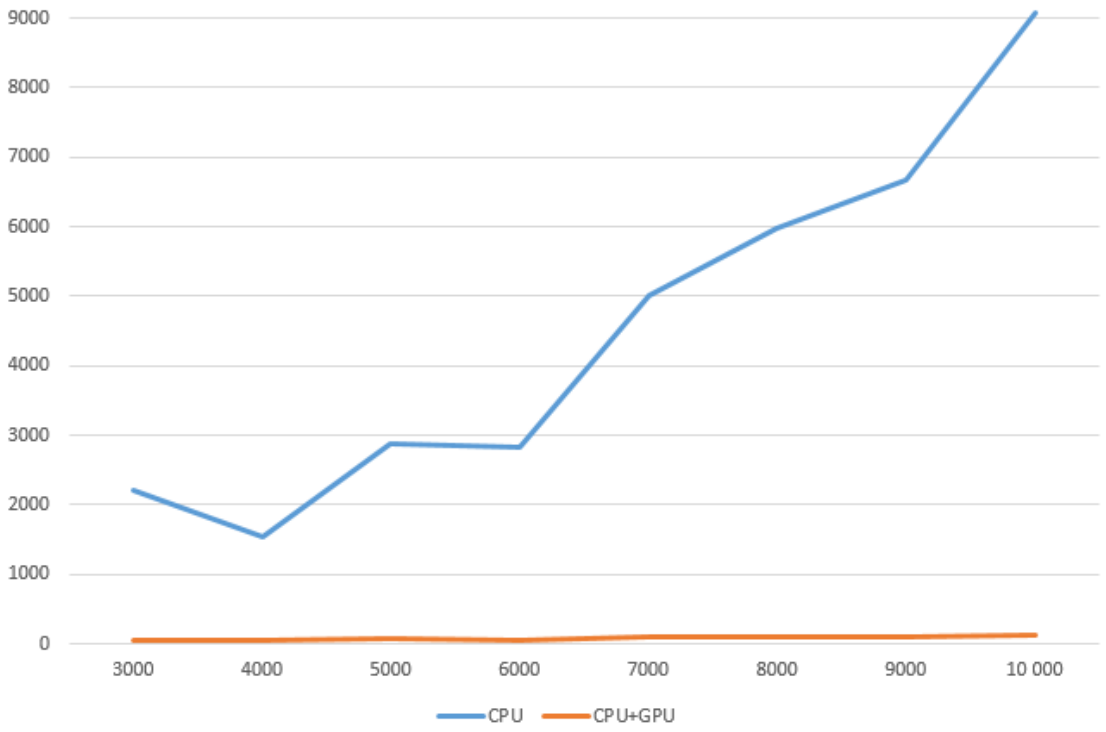

Рисунок 3.12 – Порівняння швидкодії роботи методу на CPU і CPU+GPU

Таблиця 3.6 – Порівняння швидкодії роботи методу на CPU і CPU+GPU

Кількість точок	К-сть ітерації	CPU, мс	CPU + GPU, мс	Коефіцієнт прискорення
3000	183	2204	60	36.7,8
4000	96	1540	45	32,2
5000	142	2868	67	42,8

Продовження таблиці 3.6

Кількість точок	К-сть ітерації	CPU, мс	CPU + GPU, мс	Коефіцієнт прискорення
6000	116	2829	61	46,4
7000	178	5005	93	53,8
8000	187	5986	99	60,5
9000	184	6682	96	69,6
10 000	226	9085	122	74,5

3.3 Застосування методу для кластеризації кольорів

Одним із популярних застосувань кластерного аналізу є сегментація зображень - це класифікація зображення в різні групи. Для цього вхідне зображення (Рисунок 3.13), використовується як вхідні дані.

Для цього кожен піксель зображення перетворюється в масив із трьох елементів, що відповідають за колір даного пікселя. Далі проводиться кластерний аналіз для заданого числа кластерів – кількість кольорів для вихідного зображення. Таким чином отримуємо центри кластерів, що будуть відповідати кольорам, які ми будемо використовувати для сегментації. Після цього замінюємо усі пікселі, що попали в певний кластер, на колір центроїда. В результаті ми отримуємо нове зображення, яке буде представлене заданою кількістю кольорів.

Для виділення значущого зображення від фону виконується кластерний аналіз для двох кластерів. Результат зображено на рисунку 3.14.

На рисунках 3.14, 3.15, 3.16 показано візуальні результати для виконання кластеризації на 2, 5 і 10 кластерів відповідно. В таблиці 3.7 наведено числові результати виконання кластеризації.

Рисунок 3.13 – Зображення для якого буде проводитися сегментація

Рисунок 3.14 – Результат кластерного аналізу для двох кластерів

Рисунок 3.15 – Результат кластерного аналізу для п`яти кластерів

Рисунок 3.16 – Результат кластерного аналізу для десяти кластерів

Таблиця 3.7 – Числові результати кластеризації зображення

Кількість вхідних точок	614 400
Кількість вимірів	3
2 кластери	
Кількість ітерацій	9
Час виконання на CPU	2955 мс
Час виконання на CPU+GPU	147 мс
Коефіцієнт прискорення	20,1
5 кластерів	
Кількість ітерацій	20
Час виконання на CPU	14 227 мс
Час виконання на CPU+GPU	248 мс
Коефіцієнт прискорення	57,4
10 кластерів	
Кількість ітерацій	49
Час виконання на CPU	67 190 мс
Час виконання на CPU+GPU	676 мс
Коефіцієнт прискорення	99,4

3.5 Висновки до розділу 3

Для програмної реалізації було обрано платформу .NET на базі операційної системи Windows, та мову програмування C#. Такий вибір дозволяє використовувати розвинене середовище розробки Microsoft Visual Studio та потужний функціонал уже існуючих бібліотек та фреймворків: CUDA та CUDAfy.Net.

В даному розділі описано архітектуру програмного забезпечення, а саме детально описано структурну схему класів програмного забезпечення. Подано

детальну інструкцію користувача з описом та ілюстраціями усіх вікон і елементів програмного забезпечення.

Наведено експериментальне порівняння швидкодії модифікованих та оригінальних методів. З таблиць 3.5 та 3.6 бачимо досить велике практичне прискорення модифікацій, а саме на 10 тисячах вхідних двовимірних точок модифікований метод k-means дає прискорення в 65,9 раз і модифікований метод c-means – в 74,5. При застосуванні модифікованого методу k-means для сегментації зображень отримуємо навіть більший виграш у швидкості виконання через збільшення кількості вхідної вибірки та кількості вимірів вхідних елементів. В загальному варто підмітити, що коефіцієнт прискорення зростає, при збільшенні кількості вхідних об’єктів. Це свідчить про те, що завдяки масово паралельним обчисленням, усі ітерації методу відбуваються паралельно на великій кількості ниток, а решта часу займає передача масивів даних між центральним та графічним процесором.

ВИСНОВКИ

У даній дипломній роботі було розроблено та реалізовано модифіковані методи кластеризації за допомогою паралельних обрахунків на графічному процесорі. Програмна реалізація даних методів написана на високорівневій мові програмування C# на платформі .NET на базі операційної системи Windows. Такий вибір дозволяє використовувати розвинене середовище розробки Microsoft Visual Studio та потужний функціонал уже існуючих бібліотек та фреймворків, що дозволяють задовольнити вимогу створення зручного графічного інтерфейсу користувача.

Першим етапом був аналіз предметної області. Був досліджений теоретичний матеріал у сфері інтелектуального аналізу даних і однієї із найважливіших його задач – кластерного аналізу. Також детально проаналізовано технологію паралельних обчислень на графічних процесорах.

На основі проведених досліджень стало можливим зробити обґрунтований вибір найбільш ефективних для поставленої задачі методів: неієрархічні алгоритми k-means та c-means. Даний вибір обґрунтовується ітераційною структурою методів, і обчислення для кожного об’єкта можна проводити незалежно від інших, що є основою масово паралельних обчислень на графічних процесорах.

В ході теоретичного дослідження обраних методів кластеризації було наведено їх модифікації для масово паралельних обчислень на графічних процесорах. Математичний аналіз складності методів показує велике теоретичне прискорення для розроблених модифікацій методів в порівнянні з виконанням на центральному процесорі, що може досягати більше ста для модифікованого методу k-means, та сотень для модифікованого методу c-means. В рамках опису архітектури побудованої системи розглянуто перелік усіх її класів та наведено їх схему структурну. У якості документації дана інструкція користувача з ілюстраціями.

Розроблений програмний продукт було використано для проведення числових експериментів при порівнянні швидкодії модифікованих та оригінальних методів. Результати числових експериментів показують досить велике практичне прискорення модифікацій, а саме на 10 тисячах вхідних двовимірних точок модифікований метод k-means дає прискорення в 65,9 раз і модифікований метод c-means – в 74,5. Також варто підмітити, що коефіцієнт прискорення зростає, при збільшенні кількості вхідних об’єктів. Це свідчить про те, що завдяки масово паралельним обчисленням, усі ітерації методу відбуваються паралельно на великій кількості ниток, а решта часу займає передача масивів даних між центральним та графічним процесором. З цього можна зробити висновок, що для надвеликих масивів вхідних даних можливо отримати практичне прискорення модифікованих методів, що буде прямувати до розрахованого теоретичного значення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Барсегян А.А. Методи і моделі аналізу даних: OLAP і Data Mining : учебное пособие. – Санкт-Петербург: БХВ-Петербург, 2004. – 336 с.

2. Боресков А.В. Параллельные вычисления на GPU. Архитектура и программная модель Cuda.: учебное пособие. – М: Издаьельство Московского университета, 2012. – 336 с.

3. Кластеризация [Електрон. ресурс]. – Режим доступу: http://www.machinelearning.ru/wiki/index.php?title=Кластеризация

4. Котов А., Красильников Н. Кластеризация данных. 2006. [Електрон. ресурс]. – Режим доступу: https://logic.pdmi.ras.ru/~yura/internet/02ia-seminar-note.pdf

5. Методы кластерного анализа. Иерархические методы [Електрон. ресурс]. – Режим доступу: https://www.intuit.ru/studies/courses/6/6/lecture/182

6. Параллельные вычисления CUDA [Електрон. ресурс]. – Режим доступу: http://www.nvidia.ru/object/cuda-parallel-computing-ru.html

7. Полетаев С. А. Параллельные вычисления на графических процессорах [Електрон. ресурс]. – Режим доступу:
http://www.iis.nsk.su/files/articles/sbor_kas_16_poletaev.pdf

8. Шалимов. Д.С. Алгоритмы устойчивой кластеризации на основе индексных функций и функций устойчивости [Електрон. ресурс]. – Режим доступу: http://www.meta.math.spbu.ru/user/gran/soi4/shalymov4.pdf

9. Big Data – Are You In Control? [Електрон. ресурс]. – Режим доступу: https://www.waterfordtechnologies.com/big-data-interesting-facts/

10. Cluster Analysis [Електрон. ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Cluster_analysis

11. CUDAfy.NET [Електрон. ресурс]. – Режим доступу: https://cudafy.codeplex.com/

12. David Arthur, Sergei Vassilvitskii How Slow is the k-Means Method? [Електрон. ресурс]. – Режим доступу: https://www2.cs.duke.edu/courses/spring07/cps296.2/papers/kMeans-socg.pdf

13. Nvidia CUDA: неграфические вычисления на графических процессорах [Електрон. ресурс]. – Режим доступу: https://www.ixbt.com/video3/cuda-1.shtml

14. Nvidia Doubles Up Tesla GPU Accelerators [Електрон. ресурс]. – Режим доступу: https://www.enterprisetech.com/2014/11/17/nvidia-doubles-tesla-gpu-accelerators

15. SIMD [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/SIMD