ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ

КУРСОВА РОБОТА

на тему:

“Розпаралелювання обчислення тріангуляції Делоне ”

Студента III курсу, групи ПМП-32,
напряму підготовки спеціальності
113 Прикладна математика
Рибачик Олександр Валерійович

Керівник: доцент кафедри
прикладної математики Макар І.Г.

Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Львів – 2024

ЗМІСТ

ВСТУП

В IT-індустрії дуже широкого застосування набула тріангуляція Делоне, а саме у графічному дизайні, комп'ютерній графіці, обробці зображень, обчислювальній геометрії та інших областях. Тріангуляція Делоне є ефективним інструментом для розділення простору на трикутники, забезпечуючи оптимальну геометричну структуру для подальших обчислень і візуалізації. Однак зростання, в сучасній IT-індустрії, об'єму даних та складності задач призводить до необхідності розробки ефективних та маштабованих методів обчислення тріангуляції Делоне. Одним із найефективніших способів вирішення цієї проблеми є метод розпаралелювання алгоритмів.

Об'єкт дослідження цієї роботи - це процес тріангуляції Делоне, який є важливим етапом у багатьох областях обчислювальної геометрії, комп'ютерної графіки та графічного дизайну.

Предмет дослідження цієї роботи є процес розпаралелювання обчислень тріангуляції Делоне . А саме дослідження методів та технік розпаралелювання , які можна застосувати для оптимізації алгоритмів тріангуляції Делоне .

Мета дослідження — огляд і аналіз існуючих алгоритмів розпаралелювання тріангуляції Делоне за для того щоб порівняти їх ефективність та швидкодію.

РОЗДІЛ 1. ТРІАНГУЛЯЦІЯ ДЕЛОНЕ

1.1. Визначення тріангуляції Делоне

Тріангуляція Делоне для множини точок P на площині - це така тріангуляція DT(P) , що жодна точка множини P не знаходиться всередині описаних довкола трикутників кіл в множині DT(P). Тріангуляція Делоне дозволяє якомога зменшити кількість малих кутів . Цей спосіб тріангуляції був винайдений Борисом Делоне в 1934 році [1].

Описане коло навколо трикутника називається пустим у тому випадку коли воно не містить інших вершини трикутників , окрім трьох вершин трикутника якого воно описує.

Умова Делоне – це умова , яка стверджує , що мережа трикутників є тріангуляцією Делоне тоді і лише тоді коли всі кола які описані навколо тих трикутників пусті. Щоб проілюструвати цю ідею, рис 1.1 показує приклад того, що відбувається, коли набір точок з’єднується довільним чином проти набору, який відповідає умові Делоне[2].

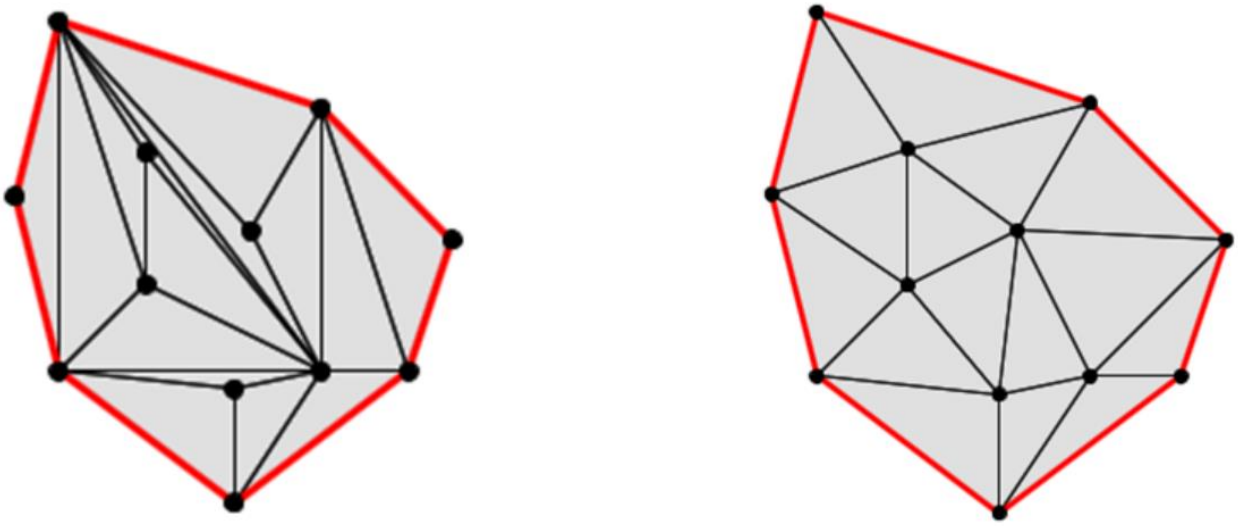

Рис 1.1 Приклад тріангуляції без та з дотриманням умови Делоне

Варто зауважити що тріангуляція Делоне має певний зв'язок з діаграмою Вороного. З’єднавши центри описаних кіл трикутників які мають спільне ребро ми отримаєм діаграму Вороного.

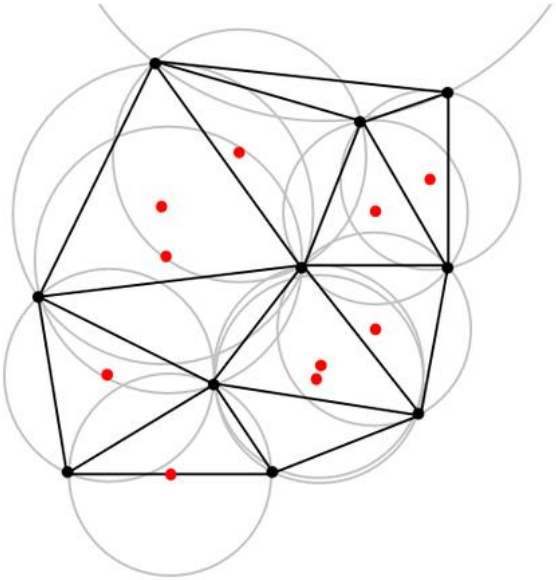

Рис 1.2 Тріангуляція Делоне зі всіма колами та їхніми центрами (червоні)

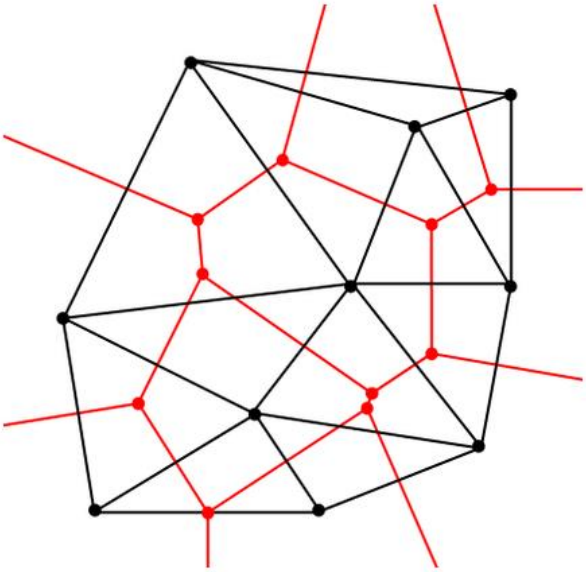

Рис 1.3 Утворення діаграми Вороного із центрів описаних кіл

Тріангуляція Делоне застосовна не лише до точок , які розташовані у двовимірному просторі, але й до тих які розташовані у просторі вищих рівнів. У багатовимірному випадку критерії тріангуляції Делоне визначаються за допомогою гіперсфер, що описується навколо кожного симплексу (високовимірного аналога трикутника). Багатовимірні випадки складніші у реалізації, проте деякі алгоритми побудови базуються на узагальненнях алгоритмів для двовимірного випадку.

1.2. Огляд відомих алгоритмів тріангуляції

Для реалізації тріангуляції Делоне застосовують багато різних алгоритмі. Кожен з них має певні свої переваги, недоліки та швидкодію. Ось кілька з них : інкрементальний алгоритм, алгоритм заміни ребра, алгоритм розділяй і володарюй, алгоритм оберненої шкали, алгоритм Форчуна. Розглянемо кожен з них.

1.2.1 Інкрементальний алгоритм

Інкрементальний алгоритм є одним із найпопулярніших і найпростіших алгоритмів. Він базується на поступовому додаванні нових точки до вже існуючої тріангуляції, та її оновлення при потребі.

Розглянемо покроково наш алгоритм:

1. Розпочинаємо з порожньої тріангуляції, яка не містить жодної точки і жодного трикутника
2. Створюємо так званий “початковий трикутник”, який охоплює всі точки.
3. Додаємо послідовно, до нашої тріангуляції, наступні точки
4. Після того як додали нову точку , нам необхідно оновити нашу тріангуляцію. Потрібно щоб вона задовольняла умову Делоне.
5. У випадку, якщо умова Делоне порушується, нам потрібно відновити правильну тріангуляцію. Цього ми можемо досягти видаляючи ребра та трикутники та додаючи нові.
6. Якщо ми додали всі точки і кожен трикутник задовольняє умову Делоне, то ми в результаті отримуємо повну тріангуляцію Делоне.

Саме цей алгоритм є простий у розумінні та реалізації та має часову складність $O(n^2)$, де n – кількість точок у нашому просторі. Проте даний алгоритм є неефективним для великих наборів точок. Це спричинено тим, що нам потрібно постійно оновлювати тріангуляцію

1.2.2 Алгоритм заміни ребра

Алгоритм заміни ребра є методом оптимізації тріангуляції Делоне за допомогою заміни одного ребра трикутника на інше. Завдяки цьому процесу ми маємо можливість покращити якість тріангуляції, зменшити кількість

трикутників або вирішити інші проблеми, що виникають при побудові тріангуляції.

Основні кроки алгоритму заміни ребра:

1. Обираємо ребро, яке плануємо замінити. Це повинне бути ребро, яке утворюється з двох сусідніх трикутників, які не задовольняють умову Делоне.
2. Перевіряємо чи нове ребро, яким ми плануємо замінити старе ребро, задовольняє умову Делоне.
3. Якщо у нас виконується умова Делоне , то замінюємо старе ребро на нове, яке утворюється шляхом додавання двох нових трикутників.
4. Старі трикутники, які містили наше старе ребро, видаляємо із тріангуляції.
5. Повторяємо попередні процеси для інших ребер, які потрібно замінити.
6. В результаті всіх операцій ми отримуємо оптимізовану тріангуляцію Делоне.

У загальному випадку цей алгоритм має часову складність $O(n)$, де n - кількість точок у нашому просторі. Однак часова складність нашого алгоритму залежить від реалізації та конкретних деталей алгоритму. Головним недоліком нашого алгоритму це є складність реалізації, оскільки потребує ретельної перевірки умови Делоне, та оновлення тріангуляції після кожної заміни старого ребра на нове.

1.2.3 Алгоритм розділяй і володарюй

Алгоритм розділяй і володарюй ґрунтується на ідеї рекурсивного розділення простору на менші частини, де на кожній з них обчислюємо тріангуляцію. Об’єднавши всі частини ми отримуємо повну тріангуляцію. Завдяки цьому підходу ми можемо покращити ефективність обчислення шляхом розділення нашої задачі на менші підзадачі.

Основні кроки нашого алгоритму:

1. Розподіляємо наш простір із точками на менші частини
2. Після того , як ми розділили наш простір, обчислюємо тріангуляцію Делоне для кожного нашого підпростору.
3. Обчисливши тріангуляцію Делоне для кожного підпростору, об’єднуємо їх разом.
4. Під час злиття тріангуляції ми вирішуємо проблеми злиття, такі як наявність ребер або трикутників, які перетинаються між частинами.
5. Після всіх операцій ми в результаті отримуємо повну тріангуляцію Делоне для вихідного набору точок.

Алгоритм розділяй і володарюй зазвичай має часову складність $O(n \log n)$, де n - кількість точок у нашому просторі. Завдяки цьому алгоритмі ми маємо можливість обробляти великі набори точок . Також цей алгоритм є корисним для просторів вищих порядків. Основним недоліком цього алгоритму є складність реалізації.

1.2.4 Алгоритм оберненої шкали

Алгоритм оберненої шкали є одним із найефективніших алгоритмів для побудови тріангуляції Делоне. Він базується на динамічному підході та є реалізацією методу заміни точки, що дозволяє швидко та ефективно оновлювати тріангуляцію при додаванні нових точок.

Основні кроки нашого алгоритму:

1. Обираємо початкову тріангуляції. Це може бути трикутник, який буде містити усі точки.
2. Додаємо нову точку до тріангуляції
3. Видаляємо всі трикутники, які перетинаються з новою точкою
4. Знаходимо оболонку – найменший опуклий чотирикутник, що містить нову точку та не містить внутрішніх точок.

5. Додаємо нові трикутники, які утворюються шляхом з’єднання нової точки та не містить внутрішніх точок.
6. Після додавання нових трикутників ми отримуємо оновлену тріангуляцію Делоне.

Алгоритм оберненої шкали зазвичай має часову складність $O(n \log n)$, де n - кількість точок у нашому просторі. Наш алгоритм особливий тим , що він працює у прямому напрямку від простіших трикутників до складніших структур, що дозволяє ефективно оновлювати тріангуляцію при додаванні нових точок. Цей підхід є корисним для динамічних застосувань, де необхідно швидко змінювати тріангуляцію відповідно до динамічних змін вихідних даних. Проте цей алгоритм може виявитись неефективним при великих наборах даних. Також він є складним у реалізації.

1.2.5 Алгоритм Форчуна

Алгоритм Форчуна є ефективним алгоритмом побудови тріангуляції Делоне, який використовує прийом замітальної оболонки для ефективного обчислення. Ідея нашого алгоритму базується на тому , що нам потрібно побудувати горизонтальну пряму через простір точок та поступово рухатись зліва направо, обробляючи кожну точку та кожне перетинання з поточним положенням прямої. Під цього процесу відбуваються такі події, як додавання нових точок, видалення трикутників та додавання нових ребер.

Основні кроки алгоритму Форчуна:

1. Починаємо з порожньої тріангуляції , та створюємо замітальну пряму.
2. Рухаємось із замітальною прямою зліва на право та обробляємо додавання нових точок, видалення трикутників та додавання нових ребер, на кожному перетині замітальної прямої з ребром тріангуляції чи точкою з нашого простору.

3. Цей процес продовжуємо до завершення обробки всіх точок та перетинів замітальної прямої.
4. В результаті ми отримуємо повну тріангуляцію Делоне.

Алгоритм Форчуна зазвичай має часову складність $O(n \log n)$, де n - кількість точок у нашому просторі. Цей алгоритм відомий своєю швидкодією та ефективністю навіть на великих наборах точок. Але нажаль цей алгоритм є складний у реалізації. Також він може показувати непередбачувані результати при недосконалих або змінених вхідних даних. Алгоритм Форчуна може стати непрактичним для просторів вищих порядків, через свою квадратичну складність.

1.3. Властивості і застосування

Властивості тріангуляції Делоне мають дуже важливий аспект у цьому методі. Вони мають значення як у практичному так і теоретичному контексті. Розглянемо деякі основні властивості тріангуляції Делоне:

- Тріангуляція Делоне мінімізує суму довжин ребер трикутників, які ми отримуємо в результаті тріангуляції. Це означає, що вона надає оптимальне розбиття геометричної області на трикутники з точки зору довжини ребер.
- При зміні позиції однієї точки або при додаванні(видаленні) точки у тріангуляції Делоне , зазвичай впливає лише на сусідні трикутник , а не на всю тріангуляцію. Завдяки цій властивості тріангуляція Делоне є ефективною для динамічних змін у наборі точок.
- У тріангуляції Делоне жодні два трикутника не перетинаються, окрім країв.

Тріангуляція Делоне має дуже широкий спектр застосувань у різних галузях. Це спричинено тим, що вона є потужним інструментом для аналізу

та обробки геометричних даних. Можна розглядати багато сфер застосування тріангуляції Делоне, проте розглянемо деякі основні сфери застосування тріангуляції Делоне.

1.3.1 Графічне відображення

Тріангуляція Делоне активно використовується у комп’ютерній графіці, а саме для гладкого відображення поверхонь та мереж. За допомогою тріангуляції можна створювати сітки, що представляють поверхню об’єкта. Це особливо важливо у 3D-моделюванні. Наприклад, у тривимірних моделях персонажів для відеоігор або архітектурних моделей тріангуляція дозволяє точно відобразити форму та деталі об'єкта (рис 1.4).

Рис 1.4 Застосування тріангуляції Делоне у 3D-моделях

Тріангуляція використовується у динамічних моделях для обробки змін поверхні в реальному часі, таких як деформації або рухи. Наприклад у симуляції рідин, тканин та інших матеріалів, що змінюють свою форму у процесі взаємодії з іншими об'єктами.

1.3.2 Обробка зображень

Тріангуляція Делоне широко використовується у комп'ютерному зоровому аналізі для розбиття зображень на трикутники та виконання різноманітних операцій обробки зображень, таких як інтерполяція, вирівнювання та згладжування. Завдяки цьому методу маємо можливість відновлювати втрачені або пошкоджені частини зображень. Трикутні сітки надають можливість більш точної інтерполяції у порівнянні з регулярними сітками. Тріангуляція Делоне також використовується у комп’ютерному зоровому аналізі. Завдяки цьому ми можемо визначати ключові точки на зображенні та вирівнювати їх для подальшої обробки. Це особливо використовується у розпізнаванні обличь, де необхідно точно вирівняти риси обличчя рис(1.5). Також варто зауважити що завдяки тріангуляції Делоне ми можемо зображення зробити більш реалістичним.

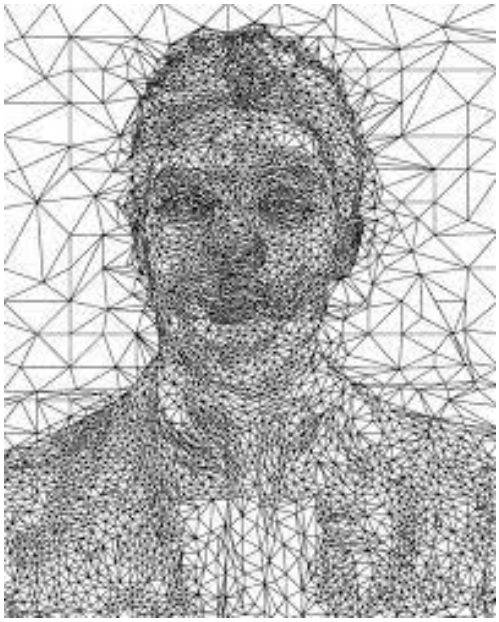

Рис 1.5 Тріангуляція у зображеннях

1.3.3 Геоінформатика

Тріангуляція Делоне відіграє важливу роль у геоінформатиці. Вона дає можливість створювати точні та детальні моделі рельєфу шляхом побудови

трикутної сітки з точок висотного вимірювання. Ці моделі можуть відображати складні рельєфні структури з високою точністю (рис 1.6). У геоінформатиці часто необхідно розбити територію на дрібніші частини для детального аналізу різних параметрів, таких як кліматичні умови, типи ґрунтів або розподіл рослинності. Тріангуляція Делоне дозволяє розбивати простір на трикутники, що полегшує аналіз та візуалізацію просторових даних (рис 1.7).

Рис 1.6 Цифрова модель рельєфу

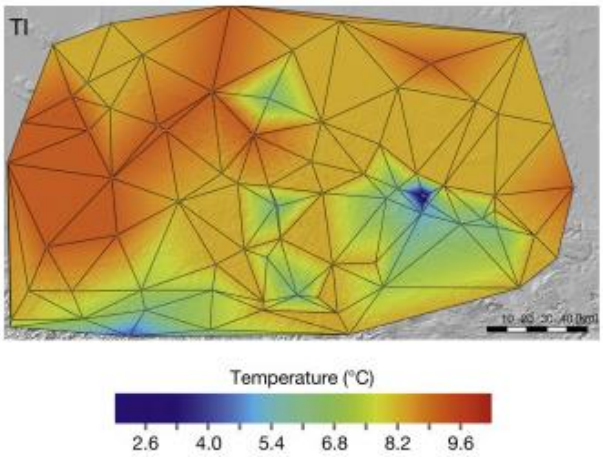

Рис 1.7 Використання тріангуляція Делоне в кліматичній карті

1.3.4 Моделювання та аналіз геометричних структур

Тріангуляція Делоне застосовується у різних галузях для моделювання та аналізу геометричних структур. Цей метод дозволяє створювати ефективні та точні моделі, які використовуються для аналізу та візуалізації складних систем. Наприклад у біології та хімії він використовується для моделювання молекулярних структур. Тріангуляція Делоне є корисною для моделювання дорожніх мереж та аналізу транспортних потоків. Вона дає нам можливість планувати нові дороги, оптимізувати маршрути та аналізувати затори. Варто зауважити що тріангуляція Делоне допомагає у створенні 3D-моделей

будівель рис (1.8). Вона забезпечує точне представлення геометрії, що є важливим для проектування та візуалізації архітектурних проектів.

Рис 1.8 Тріангуляція в архітектурі

Існує ще багато інших сфер діяльності де застосовується тріангуляція Делоне про які не згадано, тому що вона має дуже великий спектр застосування. Також варто зауважити що вона знаходить нові застосування у багатьох галузях науки та технологій.

РОЗДІЛ 2. МЕТОДИ РОЗПАРАЛЕЛЮВАННЯ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ

2.1. Розпаралелювання алгоритмів

Розпаралелювання алгоритмів – це процес розбиття обчислювальної задачі на менші підзадачі та одночасне виконання цих підзадач на різних обчислювальних ресурсах, таких як процесорні ядра, обчислювальні вузли або графічні прискорювачі. Завдяки цьому підходу ми отримуємо можливість прискорити виконання обчислень, підвищити продуктивність та ефективність програм, особливо для завдань, які вимагають великої кількість обчислень.

Важливо нам знати наскільки ми пришвидшимо наш алгоритм в залежності від кількості паралельних процесів. Потенційне прискорення паралельної програми визначається тим, наскільки швидше алгоритм може виконуватися на паралельній архітектурі порівняно з послідовним виконанням. Це залежить від кількох факторів, зокрема від частки програми, яка може бути розпаралелена, і від архітектурних обмежень системи. Існують закони які описують потенційне прискорення розпаралеленого алгоритму. Розглянемо два з ним, це закон Амдала та закон Густафсона.

Закон Амдала є фундаментальною теоремою, що описує потенційне прискорення розпаралелізованого алгоритму. Вона стверджує, що максимальне прискорення програми обмежене послідовною частиною алгоритму. Формула закону Амдала має вигляд :

$$S = \frac{1}{(1 - P) + \frac{P}{N}}$$

де S – прискорення, P – частка алгоритму, яка може бути розпаралелена, N – кількість процесорів. Закон Адамса нам показує, що навіть при дуже великій кількості процесорів прискорення обмежене часткою послідовної роботи. Наприклад, якщо 90% алгоритму ми можемо розпаралелити

(P=0.9), теоретичне максимальне прискорення при необмеженій кількості процесорів буде

$$S(\infty) = \frac{1}{(1-0.9) + \frac{0.9}{\infty}} = \frac{1}{0.1} = 10$$

З цього випливає, що навіть при безмежному числі процесорів, прискорення не перевищить 10 разів.

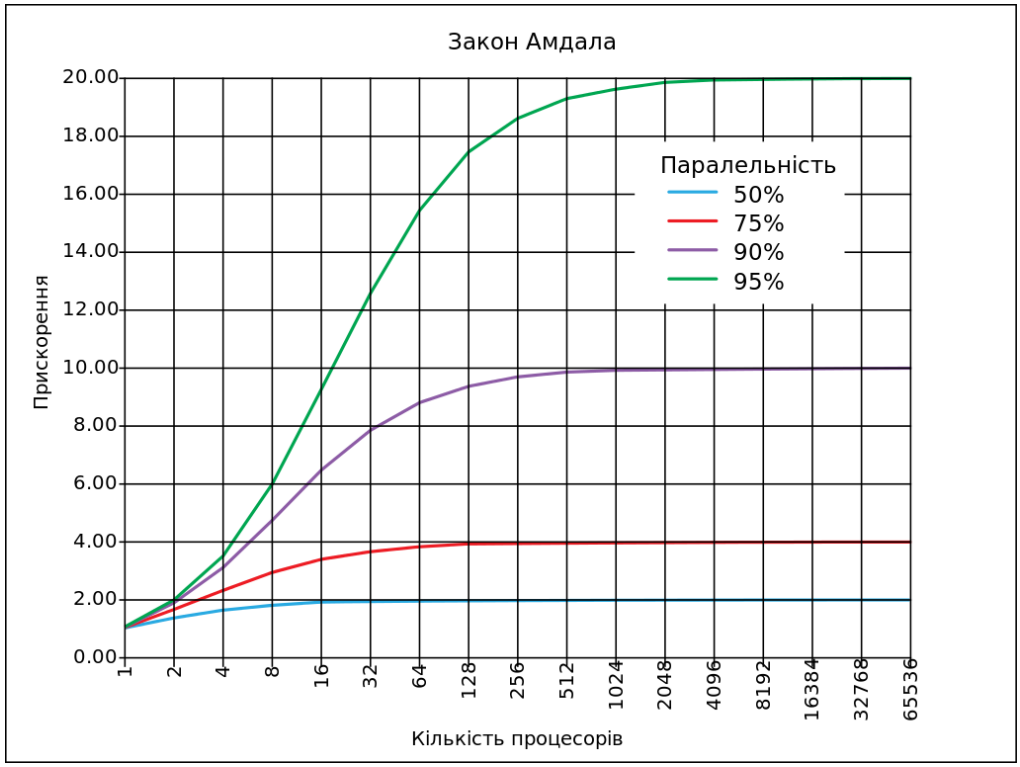

Рис 2.1 Графік залежності прискорення виконання алгоритму залежно від числа процесорів при різних рівнях розпаралелювання алгоритму за законом Амдала

Закон Густафсона пропонує інший підхід до оцінки ефективності паралельних обчислень, зосереджуючи увагу на масштабованості задачі. Він стверджує, що загальний час виконання алгоритму може бути зменшено навіть при великій частці послідовної роботи, якщо зростає розмір задачі. Формула закону Густафсона має вигляд:

$$S = P + N \cdot (1 - P)$$

де S – прискорення, P – частка послідовної роботи алгоритму, N – кількість процесорів. Закон Густафсона показує, що при збільшенні кількості процесорів можна досягти значного прискорення, якщо паралельні частини алгоритму можуть масштабуватися. Наприклад, якщо 90% програми може бути розпаралелено (P=0.1), теоретичне прискорення при 100 процесорах буде:

$$S(100) = 0.1 + 100 \cdot (1 - 0.1) = 0.1 + 100 \cdot 0.9 = 0.1 + 90 = 90.1$$

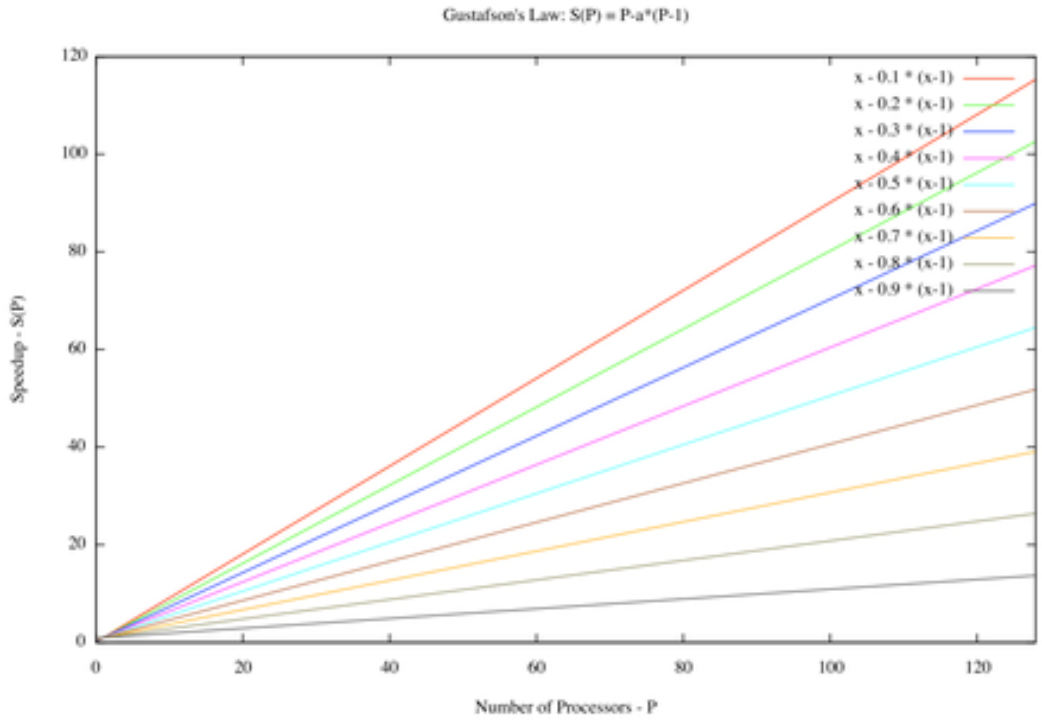

Рис 2.2 Графік залежності прискорення виконання алгоритму залежно від числа процесорів при різних рівнях розпаралелювання алгоритму за законом Густафсона

Основні аспекти розпаралелювання алгоритмів :

- Обчислювальна задача розбивається на менші частини, які виконуються незалежно одна від одної.
- Кожен обчислювальний вузол або потік повинен взаємодіяти з іншими частинами системи, наприклад, шляхом обміну даних або координації дій.

- Для того щоб коректно виконувалась задача необхідно щоб обчислювальні вузли або потоки були синхронізовані. Це для того щоб уникнути конфліктів даних та забезпечити правильний порядок виконання операцій.
- Система повинна вміти ефективно розподіляти та керувати завданнями між різними обчислювальними вузлами або потоками для забезпечення оптимального використання ресурсів.
- Розпаралелювання повинно бути ефективним навіть при збільшенні обсягу обчислень або кількості обчислювальних ресурсів.

Існують різні технології та стратегії розпаралелювання, такі як розподілена обробка даних, розподілена обробка задач, паралельне програмування, використання мультипроцесорних архітектур та багато інших. Вибір конкретної стратегії залежить від характеру задачі, обчислювальних ресурсів та вимог до продуктивності та ефективності.

2.2. Основні підходи до розпаралелення тріангуляції Делоне

Розпаралелювання тріангуляції Делоне дозволяє ефективно обробляти великі набори точок і зменшувати час обчислення. Проте це складне завдання, яке потребує врахування численних факторів для забезпечення ефективної та коректної реалізації алгоритму. Складність цього завдання зумовленна тим що залежить від геометричних структур і обмеженнями які, пов’язані з умовою Делоне. Розглянемо деякі методи та підходи, які використовують для розпаралелювання тріангуляції Делоне.

2.2.1.Розділяй і володарюй

Алгоритм розділяй і володарюй для тріангуляції Делоне ми вже розглядали у пункті 1.2.3. . Цей метод особливо підходить для розпаралелювання, оскільки його природна структура дозволяє розділити

задачу на підзадачі, які ми можемо паралельно вирішувати. Після обробки підзадач результати об’єднуються для отримання остаточної тріангуляції. Об’єднання може бути виконане в кілька етапів, де результати від окремих потоків об’єднуються паралельно а потім результати від цих об’єднань об’єднуються знову. Даний підхід розпаралелювання кидає нам певні виклики. Процес об’єднання тріангуляції може бути складним і вимагати додаткових обчислень для виправлення порушень умови Делоне. Також нам необхідно забезпечити рівномірний розподіл точок між потоками, щоб уникнути ситуації, коли один потік обробляє значно більше точок, ніж інші.

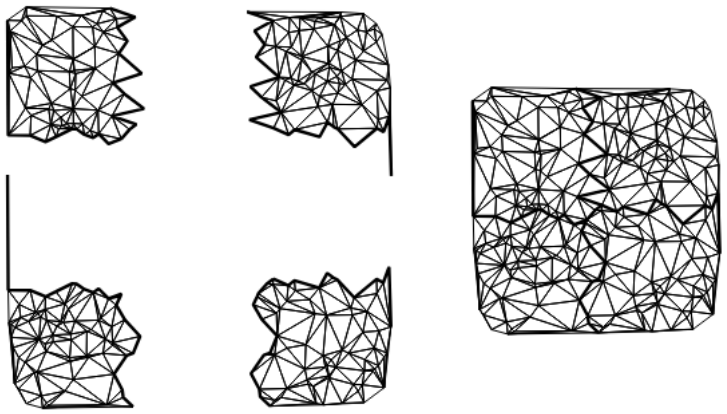

Рис 2.3 Приклад алгоритму розділяй і володарюй для чотирьох процесорів

2.2.2. Алгоритм Форчуна

Алгоритм Форчуна для тріангуляції Делоне ми вже розглядали у пункті 1.2.5. . Паралельна обробка цього алгоритму може бути досягнута за допомогою розподілення прямої між різними обчислювальними потоками або вузлами. Кожен потік або вузол може обробляти свою власну частину прямої незалежно від інших, що дозволяє ефективно використовувати паралельні обчислювальні ресурси. Це відбувається так: замітальна пряма рухається через набір точок, і кожна точка обробляється незалежно.

Паралельна обробка алгоритму Форчуна може значно збільшити продуктивність обчислень. Завдяки цьому ми можемо швидко та ефективно

будувати тріангуляцію Делоне навіть для великих наборів точок. Незважаючи на переваги цього методу ми має і деякі потенційні недоліки. Даний метод не дає нам можливості рівномірно розподілити навантаження між різними обчислювальними вузлами або потоками. Деякі області даних можуть бути більш витратними на обробку, що може призвести до нерівномірного розподілу роботи та затримок в обчисленнях. Також варто зауважити, що під час паралельного виконання можуть виникати проблеми зі збоєм вузлів або потоків. Це може вплинути на стабільність та надійність алгоритму. Необхідно мати механізми відновлення та управління помилками. Не можна оминути і те що паралельна реалізація алгоритму Форчуна є набагато складнішою ніж проста реалізація.

2.2.3. Паралельна інкрементальна тріангуляція

Інкрементальний алгоритм, який ми розглядали у пункті 1.2.1. , теж підходить для розпаралелювання. Паралельна обробка цього алгоритму полягає у паралельному додаванні точок до тріангуляції. Кожен потік або процес відповідає за обробку певної підмножини точок і проводить оновлення тріангуляції шляхом додавання нових точок та перевірки та оновлення сусідніх трикутників. Паралельна інкрементальна тріангуляція добре маштабується для великих наборів даних, оскільки можна розділити обробку точок між багатьма потоками або процесами, що дозволяє прискорити обчислення. Цей метод дозволяє використовувати ресурси обчислювальної системи більш ефективно, через те що деякі обчислення можуть відбуватися паралельно. Проте без недоліків не обійтися. Між потоками або процесами, для уникнення конфліктів даних під час оновлення нашої тріангуляції, потрібна складна синхронізація. Внаслідок чого реалізація паралельної інкрементальної тріангуляції може бути складною.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ЕКСПЕРЕМЕНТИ

Для реалізації та проводження експерементів над тріангуляцією Делоне було обрано мову програмування Python та використанно ряд бібліотек. Основні бібліотеки, що були використанні, включають:

- NumPy: використовується для роботи з числовими даними та операціями лінійної алгебри. NumPy надає швидкість обробки масивів даних, що робить його ідеальним вибором для обчислень, пов'язаних з геометрією.
- Matplotlib: це бібліотека для візуалізації даних у Python. Вона дозволяє створювати різноманітні графіки, включаючи точкові діаграми та графіки, що відображають результати алгоритмів тріангуляції.
- Time: ця бібліотека в мові програмування Python надає функції для роботи з часом. Вона дозволяє вам вимірювати час виконання коду, затримувати виконання програми на певний проміжок часу, отримувати поточний час та інше.
- Triangle: це бібліотека, яка надає нам можливість за допомогою готових функцій реалізовувати тріангуляцію Делоне.
- SciPy: Використовується для реалізації алгоритмів чисельних методів та оптимізації, що може бути корисним для обробки геометричних даних та розрахунків. Також ця бібліотека містить клас Delaunay, за допомогою якого можна реалізувати тріангуляцію Делоне.
- Joblib: надає інструменти для паралельного виконання задач у Python. Основна функціональність полягає у виконанні обчислень паралельно за допомогою розподілення задач між доступними ядрами процесора.

Свою роботу я розпочав з реалізації інкрементального алгоритму. Цей алгоритм описано у пункті 1.2.1 . Для реалізації інкрементального алгоритму

тріангуляції Делоне я використовував мову програмування Python разом з бібліотеками numpy, matplotlib та time.

Для початку я створив простір з точками, саме на якому ми будемо виконувати наш інкрементальний алгоритм для тріангуляції Делоне.

```
np.random.seed(42)
points = np.random.rand(10, 2)
```

З допомогою функції def delaunay(points) ми обчислюємо тріангуляцію Делоне. Саме в ній описаний інкрементальний алгоритм. Розпочинаємо наш алгоритм із створення “початково трикутника”, який буде охоплювати всі точки.

```
    xmin = np.min(points[:, 0]) - 1
    ymin = np.min(points[:, 1]) - 1
    xmax = np.max(points[:, 0]) + 1
    ymax = np.max(points[:, 1]) + 1
    a = [xmin, ymin]
    b = [xmax, ymin]
    c = [xmin, ymax]
    super_triangle = [a, b, c]
    triangles = [super_triangle]
```

Дальше ми проходимо по кожній точці і по черзі додаємо кожну з них до нашої тріангуляції. Створюємо два списки: edges для зберігання ребер, які потрібно буде перевірити, та bad_triangles для зберігання трикутників, що порушують властивість Делоне.

```
for point in points:
        edges = []
        bad_triangles = []
```

Для кожного трикутника з поточної тріангуляції перевіряється, чи потрапляє нова точка всередину описаного кола цього трикутника. Це перевірити нам

допомагає функція in_circle . Якщо трикутник порушує умову Делоне, він додається до списку bad_triangles, а його ребра до списку edges .

```
 for triangle in triangles:
        if in_circle(triangle[0], triangle[1], triangle[2], point):
            bad_triangles.append(triangle)
            edges.append([triangle[0], triangle[1]])
            edges.append([triangle[1], triangle[2]])
            edges.append([triangle[2], triangle[0]])
```

Трикутники із списку bad_triangles видаляються з тріангуляції.

```
for triangle in bad_triangles:
        for triangle_index in range(len(triangles)):
            if np.array_equal(triangles[triangle_index], triangle):
                del triangles[triangle_index]
                break
```

Дальше ми сортуємо edges , щоб дублікати ребер розташовувались поруч. Відсортувавши список ми видаляємо дублікати ребер із списку.

```
edges.sort(key=lambda x: (x[0][0], x[0][1], x[1][0], x[1][1]))
    i = 0
    while i < len(edges) - 1:
        if np.array_equal(edges[i], edges[i + 1]):
            del edges[i]
        else:
            i += 1
```

Дальше для кожного ребра створюємо новий трикутник з новою точкою, який додається до тріангуляції.

```
for edge in edges:
        new_triangle = [edge[0], edge[1], point]
        triangles.append(new_triangle)
```

Додавши всі точки до тріангуляції, ми з тріангуляції видаляємо трикутники які мають спільні вершини з початковим трикутником. Повинні лишитися тільки ті трикутники, які побудовані з початкових точок.

```
result_triangles = []
    for triangle in triangles:
        if not any(np.all(point == vertex) for point in super_triangle for vertex in triangle):
            result_triangles.append(triangle)
    return result_triangles
```

Реалізувавши інкрементаьний алгоритм я вирішив розпаралелити його. Основна ідея його розпаралелення описана у пункті 2.2.3 . Я використав функціональність Parallel з бібліотеку joblib, яка дозволяє розпаралелити виконання нашого алгоритму. До нашої delaunay додано новий параметр n_jobs=-1, який передає кількість паралельних процесів. За замовчуванням цей параметр буде приймати -1, що означає використання всіх доступних ядер процесора. Це дозволяє використовувати максимальну кількість ресурсів для розпаралелення.

```
def delaunay(points,n_jobs=-1)
```

Функціонал, який ми використовували для додавання нової точки до тріангуляції та створення нових трикутників ми помістили у функцію def build_triangles(point). Саме ця функція буде виконуватись паралельно. Паралельне виконання цієї функції ми запускаємо за допомогою Parallel.

```
Parallel(n_jobs=n_jobs)(delayed(build_triangles)(point) for point in points)
```

Щоб визначити ефективність нашого розпаралеленя проведено дослідження. Я виміряв час виконання алгоритму з різною кількістю точок та з різною кількістю паралельних процесів. Результати вимірювань наведені у таблиці :

Кількість точок	Кількість паралельних процесів		
	1	2	3
10	0,00201	0,74799	0,81912
100	0,01599	0,83609	0,88428
1000	0,14408	0,86675	0,97196
10000	1,48538	1,6043	1,59309
100000	14,89002	11,32901	7,06545
500000	74,6136	50,01958	41,00163
1000000	148,6647	105,406	77,04587
1500000	249,3798	164,0008	139,7351

Нижче на рис 3.1 та рис 3.2 відображено результат дослідження у виді графіків. На цих графіках ми можемо побачити що до невеликої кількості точок не варто застосовувати декілька паралельних процесів , оскільки один процес краще справляється . Але вже в діапазоні 100000 ми можемо побачити що розпаралелення стає ефективним , тому що при більшій кількості паралельних процесів, виконання програми стає швидшим порівняно з виконанням цього алгоритму одним процесом.

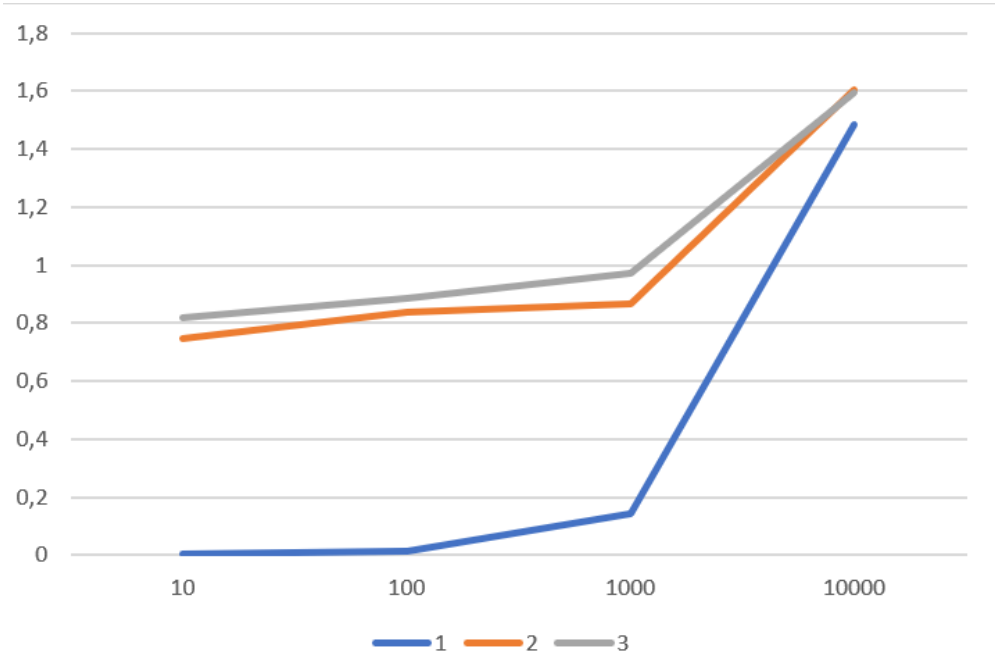

Рис 3.1 Графік з часом виконання алгоритму на діапазоні кількості точок від 10 до 10000

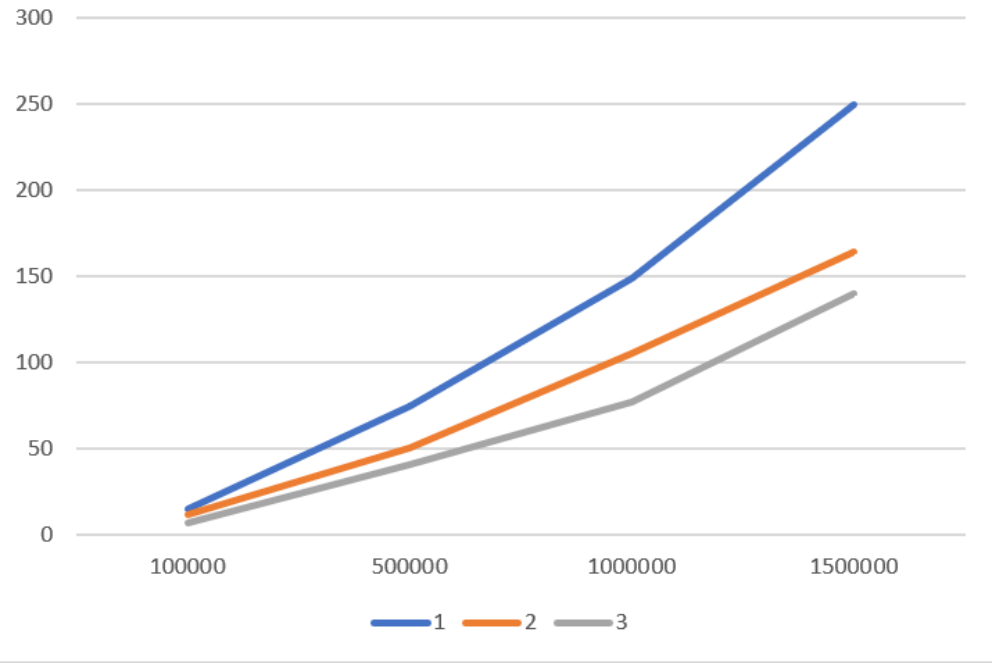

Рис 3.2 Графік з часом виконання алгоритму на діапазоні кількості точок від 100000 до 1500000

Для реалізації тріангуляції Делоне існують готові функції у Python бібліотеках. Одними із таких готових функцій є triangulate з бібліотеки triangle та функція Delaunay з бібліотеки scipy.spatial. Кожна з них під капотом містить свої алгоритми для реалізації тріангуляції Делоне. І відповідно кожна з них має свою швидкодію. Розгляне приклади як застосувати наші функції до такого випадкового набору точок:

```
np.random.seed(42)
points = np.random.rand(100, 2)
```

Функція triangulate:

```
triangulation = tr.triangulate({'vertices': points})
```

Функція Delaunay:

```
triangulation = Delaunay(points)
```

Я провірив їхню швидкодію на різній кількості точок у нашому просторі. Результати наведені у таблиці:

10 точок						Середнє	Різниця
triangulate	0,00016	0,00017	0,00016	0,00018	0,00017	0,000168	
Delaunay	0,00135	0,0015	0,0014	0,00136	0,00146	0,001414	0,001246
100 точок							
triangulate	0,00019	0,00022	0,00022	0,0002	0,00022	0,00021	
Delaunay	0,00178	0,00182	0,00174	0,00163	0,00175	0,001744	0,001534
1000 точок							
triangulate	0,00065	0,00062	0,00058	0,00063	0,00067	0,00063	
Delaunay	0,00382	0,00388	0,0039	0,00394	0,00422	0,003952	0,003322
10000 точок							
triangulate	0,00538	0,00548	0,00568	0,00562	0,00586	0,005604	
Delaunay	0,0292	0,03353	0,03124	0,033	0,03666	0,032726	0,027122
100000 точок							
triangulate	0,06511	0,05823	0,05837	0,05836	0,06322	0,060658	
Delaunay	0,52747	0,46889	0,48653	0,46106	0,50516	0,489822	0,429164
1000000 точок							
triangulate	0,91326	0,89079	0,88963	0,95849	0,89493	0,90942	
Delaunay	8,73809	9,10681	8,82208	9,04529	8,77671	8,897796	7,988376

Як ми можемо побачити, за результатами дослідження, функція triangulate є швидшою ніж Delaunay і чим більша кількість точок тим більша різниця між ним у швидкості виконання.

Обидві функції мають свої переваги та недоліки, і вибір між ними залежить від конкретних вимог проекту. Якщо вам потрібна висока продуктивність, гнучкість та можливості обробки складних геометрій, triangulate може бути кращим вибором. Якщо ж вам потрібна проста у використанні, добре документована функція для загальних випадків тріангуляції, Delaunay є відмінним варіантом.

Варто нам порівняти наскільки швидшою є готова функція triangulate від самостійно реалізованого інкрементального алгоритму. Я теж провів дослідження і порівняв їх швидкодію на різних просторах із різною кількістю точок. Як і було очікувано, готова функція triangulate виявилась набагато швидшою ніж самостійно реалізований алгоритм. Напевно головною перевагою самостійно реалізованого алгоритму є те, що ми маємо можливість ефективніше застосовувати розпаралелювання для нашого коду. Це припущення випливає з того, що ми маємо можливість розпаралелити

саме алгоритм, а не розпаралелити його виконнаня для різних наборів точок. Результати дослідження наведені у таблиці:

						Середнє	Різниця
10 точок							
triangulate	0,00016	0,00017	0,00016	0,00018	0,00017	0,000168	
власний	0,00499	0,00499	0,00312	0,00501	0,002	0,004022	0,003854
100 точок							
triangulate	0,00019	0,00022	0,00022	0,0002	0,00022	0,00021	
власний	0,01498	0,01509	0,01494	0,015	0,01499	0,015	0,01479
1000 точок							
triangulate	0,00065	0,00062	0,00058	0,00063	0,00067	0,00063	
власний	0,135	0,145	0,13765	0,14374	0,13506	0,13929	0,13866
10000 точок							
triangulate	0,00538	0,00548	0,00568	0,00562	0,00586	0,005604	
власний	1,40974	1,36997	1,43534	1,43083	1,38538	1,406252	1,400648
100000 точок							
triangulate	0,06511	0,05823	0,05837	0,05836	0,06322	0,060658	
власний	13,76169	13,60782	13,61846	14,49634	13,94085	13,88503	13,82437

Варто попробувати застосувати розпаралелення до готової функціїї Delaunay. Я використав функціональність Parallel з бібліотеку joblib, яка дозволяє розпаралелити виконання нашого алгоритму. У змінну n_jobs передав кількість паралельних процесів. Дальше наш простір з точками ділимо на підпростори і кожен з них передаємо для обробки паралельним процесам points_chunks = np.array_split(points, n_jobs)

triangulations = Parallel(n_jobs=n_jobs)(delayed(compute_delaunay)(chunk) for chunk in points_chunks)

Функція compute_delaunay , яку ми передаємо для Parallel, обчислює тріангуляцію Делоне для кожного підпростора.

```
def compute_delaunay(points_chunk):
    return Delaunay(points_chunk)
```

Об’єднання тріангуляцій відбувається за допомогою функції merge_triangulations. У ній реалізований найпростіший спосіб об’єднання тріангуляцій, проте існують і інші способи, які є складніші проте ефективніші.

```
def merge_triangulations(triangulations, points_chunks):
    all_points = np.vstack(points_chunks)
    global_triangulation = Delaunay(all_points)
    return global_triangulation
```

Щоб визначити ефективність нашого розпаралеленя проведено дослідження. Я виміряв час виконання алгоритму з різною кількістю точок та з різною кількістю паралельних процесів. Результати вимірювань наведені у таблиці:

Кільсть точок	Кількість паралельних процесів			
		1	2	3
	1000	0,00202	0,86929	0,95207
	10000	0,00418	0,89965	1,00451
	100000	0,52747	1,63877	1,7366
	500000	4,21527	7,05741	6,74022
		1	2	3
	1000000	8,84536	14,93547	13,6123
	2000000	18,8384	30,34742	28,36957
	3000000	20,49823	38,15258	33,35589
	4000000	40,33414	62,64337	57,87883
	5000000	49,73258	73,45045	70,96239

Нижче на рис 3.3 відображено результат дослідження у виді графіка. На цьому графіку ми можемо побачити що реалізоване розпаралелення не є ефективним до тієї кількості точок якої ми перевіряли, оскільки одне послідовне виконання краще справляється. Це може бути пов’язано з тим що не ефективно реалізоване об’єднання тріангуляцій. Нам значно збільшує час виконання те, що ми в кінці проводимо перевірку всього простору точок, а не на межах наших підпросторів. Реалізація коректного та ефективного алгоритму об'єднання локальних тріангуляцій є складною задачею, що вимагає значних знань та часу на імплементацію і тестування. Просте об'єднання всіх точок в глобальну тріангуляцію не вирішує цієї проблеми та може створити додаткові складнощі у подальшій обробці та візуалізації.

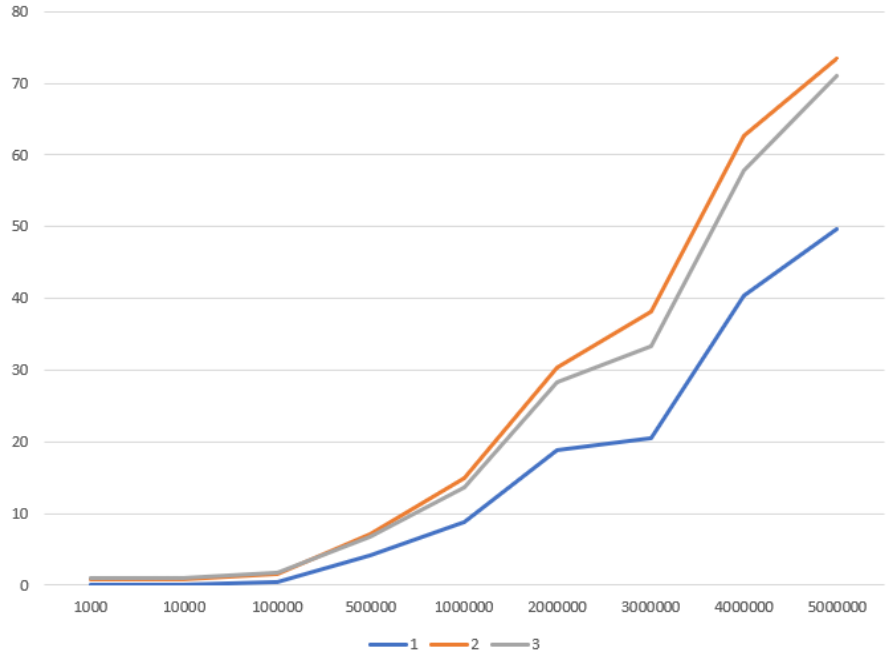

Рис 3.3 Графік з часом виконання розпаралеленого алгоритму та послідовного алгоритму

ВИСНОВКИ

У ході виконання курсової роботи було ознайомлено з основними теоретичними засадами тріангуляції Делоне, розглянуто різні алгоритми її побудови, такі як інкрементальний алгоритм, алгоритм заміни ребра, алгоритм розділяй і володарюй, алгоритм оберненої шкали та алгоритм Форчуна. Кожен із цих алгоритмів має свої особливості, переваги та недоліки, що було проаналізовано. Також були проаналізовані властивості та застосування тріангуляції Делоне у графічному відображенні, обробці зображень, геоінформатиці та моделюванні геометричних структур. З цього було визначено, що тріангуляція Делоне активно застосовується і є дуже необхідною у різних сферах діяльності.

Окрім вище згаданого у ході роботи було також розглянуто основні підходи розпаралелювання алгоритмів тріангуляції Делоне. Визначені основні переваги та недоліки цих підходів. Важливим аспектом є те, що розпаралелювання дозволяє значно прискорити процес обчислення тріангуляції для великих наборів даних, що має велике практичне значення.

З допомогою реалізації одного із основних алгоритмів а саме інкрементального алгоритму, було більше детально ознайомлено з практичною стороною тріангуляції Делоне. Також за допомогою розпаралелення цього алгоритму було досліджено ефективність розпаралелювання. В результаті чого ми побачили що справді розпаралелювання допомагає нам прискорити виконання алгоритму на великих наборах точок. У ході експерементів було порівняно дві готові функції, які виконують тріангуляцію Делоне, та визначено яка з них має кращу швидкодію. Також було цікаво дізнатись на скільки є кращою готова функція від самостійно реалізованого алгоритму. Результат був очевидний, готова функція значно краща.

Не можна було і оминути спроби реалізувати розпаралелювання готової функції. Був використаний такий підхід, розбивається набір точок на декілька менших частин, для паралельного обчислення тріангуляції Делоне для кожної частини. Після обчислення тріангуляції для кожної частини, об’єднуємо їх, створюючи глобальну тріангуляцію для всього набору точок. Проте просте об'єднання всіх точок в глобальну тріангуляцію не було ефективним методом для розпаралелювання. Для досягнення кращих результатів потрібна складніша реалізація.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. B. Delaunay: Sur La sph*ère vide, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh I Estestvennykh Nauk, 7:793-800, 1934*
2. https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html
3. https://www.sciencedirect.com/topics/earth-and-planetary-sciences/delaunay-triangulation
4. https://uk.wikipedia.org/wiki/Закон_Амдала
5. https://uk.wikipedia.org/wiki/Закон_Густавсона_—_Барсіса
6. https://www.researchgate.net/publication/3683562_An_improved_parallel_algorithm_for_Delaunay_triangulation_ondistributed_memory_parallel_computers