Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

Кваліфікаційна робота

Освітній ступінь – магістр

На тему **«ВДОСКОНАЛЕННЯ АЛГОРИТМУ КАНТОРА-ЗАССЕНГАУЗА ДЛЯ ФАКТОРИЗАЦІЇ ПОЛІНОМІВ»**

Виконав: студент 2-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Тіхонов Андрій Миколайович

Керівник: Малашонок Геннадій Іванович
доктор фіз-мат наук, професор.

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____

«____» _____ 2024 р.

Київ – 2025

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

____ (підпис) _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту **2 р.н. магістерської програми Прикладна математика**
Тіхонову Андрію Миколайовичу

Розробити
Програмну реалізацію послідовного та вдосконаленого (паралельного) алгоритму Кантора-Зассенгауза для факторизації поліномів

Зміст текстової частини кваліфікаційної роботи:

Дата видачі "____" _____ 2024 р.

Керівник
Г.І. Малашонок, доктор фіз.-мат. наук, професор _____ (підпис)

Завдання отримав _____ (підпис)
А.М. Тіхонов

Тема: ВДОСКОНАЛЕННЯ АЛГОРИТМУ КАНТОРА-ЗАССЕНГАУЗА ДЛЯ ФАКТОРИЗАЦІЇ ПОЛІНОМІВ

Календарний план виконання роботи:

№ п/п	**Назва етапу дипломної роботи**	**Термін виконання етапу**	**Примітка**
1.	Отримання завдання на дипломну роботу	01.11.2024	
2.	Огляд технічної літератури за темою роботи	30.12.2024	
3.	Реалізація послідовного алгоритму Кантора-Зассенгауза	30.01.2025	
4.	Реалізація паралельного алгоритму Кантора-Зассенгауза	30.03.2025	
5.	Тестування алгоритмів	30.04.2025	
6.	Написання тез для конференції	28.04.2025	
7.	Участь у XIII Всеукраїнській науковій конференції молодих математиків	09.05.2025	
8.	Написання текстової частини роботи	25.05.2025	
10.	Попередній захист роботи	28.05.2025	
11.	Коригування роботи за результатами попереднього захисту	02.06.2025	
12.	Остаточне оформлення текстової частини роботи та презентація	06.06.2025	
13.	Захист магістерської роботи	13.06.2025	

Тіхонов Андрій Миколайович

Керівник Малашонок Геннадій Іванович «____» _____ 2025 р.

Зміст

Анотація

У роботі запропоновано метод прискорення алгоритму Кантора-Зассенгауза для факторизації поліномів над скінченними полями. Розроблено паралельну модель алгоритму, що використовує стратегію динамічного балансування навантаження «work stealing» для ефективної роботи на багатоядерних системах. Програмна реалізація та експериментальне порівняння з послідовною версією підтвердили прискорення обчислень, що є актуальним для криптографії та комп'ютерної алгебри.

Ключові слова: Факторизація поліномів, скінченне поле, поле Галуа, алгоритм Кантора-Зассенгауза, алгоритм Берлекампа, паралельні обчислення, багатоядерні системи, work stealing, обчислювальна складність.

Anotation

The paper proposes a method for speeding up the Cantor-Sassenhaus algorithm for factorizing polynomials over finite fields. A parallel model of the algorithm has been developed that uses a dynamic load balancing strategy called "work stealing" for efficient operation on multi-core systems. The software implementation and experimental comparison with the sequential version confirmed the computational speedup, which is relevant for cryptography and computer algebra.

Keywords: Factorization of polynomials, finite field, Galois field, Cantor-Zassenhaus algorithm, Berlekamp algorithm, parallel computing, multicore systems, work stealing, computational complexity.

Вступ

Актуальність теми. Факторизація поліномів над скінченними полями — це центральна задача комп'ютерної алгебри. Її застосування охоплюють криптографію з відкритим ключем, теорію кодування для корекції помилок, зокрема в кодах Боуза-Чоудхурі-Хоквінгема та Ріда-Соломона, а також системи символьних обчислень. Для розв'язання цієї задачі існують класичні методи, серед яких виділяються алгоритми Берлекампа [1] та Кантора-Зассенгауза [2]. Продуктивність цих алгоритмів, особливо в умовах паралельних обчислень на багатоядерних архітектурах, є предметом активних досліджень.

Алгоритм Кантора-Зассенгауза, представлений у 1981 році, є ймовірнісним методом для знаходження незвідних множників полінома. Оскільки зростання продуктивності одноядерних процесорів уповільнилося, фокус досліджень змістився на паралелізацію обчислень для ефективного використання сучасних багатоядерних систем. Тому розробка паралельної версії цього алгоритму є перспективним напрямом для прискорення обчислень та повнішого залучення ресурсів теперішніх комп'ютерних архітектур.

Мета і задачі дослідження. *Метою* даної дипломної роботи є підвищення обчислювальної ефективності алгоритму Кантора-Зассенгауза шляхом розробки та аналізу його паралельної моделі, орієнтованої на багатоядерні системи з використанням динамічного балансування навантаження.

Для досягнення поставленої мети необхідно вирішити такі *задачі*:

1. Проаналізувати теоретичні основи задачі факторизації поліномів над скінченними полями.
2. Дослідити наявні детерміністичні та ймовірнісні алгоритми факторизації, зокрема алгоритми Берлекампа та Кантора-Зассенгауза.
3. Визначити обчислювальну складність за часом алгоритмів.
4. Визначте етапи алгоритму Кантора-Зассенгауза, які є ресурсомісткими та придатними для розпаралелювання.
5. Розробити паралельну модель алгоритму на основі стратегії динамічного балансування навантаження "work stealing".
6. Здійснити програмну реалізацію послідовної та запропонованої паралельної версії алгоритму.
7. Експериментально дослідити ефективність запропонованого підходу, порівнявши час виконання та показники прискорення для різних вхідних даних.

Об'єкт і предмет дослідження. *Об'єктом дослідження* є процес факторизації поліномів над скінченними полями. *Предметом дослідження* є алгоритм

Кантора-Зассенгауза, його обчислювальна складність та методи його паралелізації з використанням динамічного балансування навантаження.

Наукова новизна отриманих результатів. Наукова новизна роботи полягає в тому, що вперше запропоновано та досліджено модель паралельної реалізації алгоритму Кантора-Зассенгауза, яка базується на стратегії "work stealing"для ефективного розподілу незалежних задач між обчислювальними потоками. Отримано експериментальні дані щодо практичного прискорення та ефективності паралельної реалізації порівняно з послідовною версією на сучасних багатоядерних процесорах.

Практичне значення отриманих результатів. Результати роботи можуть бути використані для прискорення обчислень у прикладних задачах криптографії та теорії кодування. Розроблена програмна реалізація може слугувати основою для створення високоефективних модулів у системах комп'ютерної алгебри (наприклад, SageMath, PARI/GP) або спеціалізованих криптографічних бібліотеках.

Апробація результатів роботи. Основні положення та результати кваліфікаційної роботи пройшли апробацію на XIII Всеукраїнській науковій конференції молодих математиків (Київ, 2025), де були представлені у вигляді доповіді. Матеріали доповіді опубліковані у збірнику тез конференції [3].

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох основних розділів, висновків, списку використаних джерел та додатків.

- *У першому розділі* наведено теоретичні відомості, необхідні для розуміння роботи: розглянуто поняття полінома, операції над ними, властивості скінченних полів та формально поставлено задачу факторизації.
- *У другому розділі* проведено огляд та аналіз існуючих методів факторизації. Класифіковано підходи на детерміністичні та імовірнісні, а також детально розглянуто алгоритми Берлекампа та Кантора-Зассенгауза, проаналізовано їхню обчислювальну складність.
- *У третьому розділі* представлено основну авторську розробку — паралельну версію алгоритму Кантора-Зассенгауза. Проаналізовано алгоритм на предмет можливостей паралелізації, обґрунтовано вибір стратегії "work stealing"та представлено псевдокод модифікованого алгоритму.
- *У четвертому розділі* описано експериментальне дослідження. Наведено опис використаних технологій (`Rust`, `Rayon`, `Criterion`), середовища тестування, представлено та проаналізовано отримані дані щодо продуктивності послідовної та паралельної версій.
- *У висновках* підсумовано основні результати роботи, підтверджено досягнення поставленої мети та окреслено можливі напрямки для подальших досліджень.

1 РОЗДІЛ. ТЕОРЕТИЧНІ ОСНОВИ ТА ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Алгебраїчні структури та поліноми

Поліном (або многочлен) є одним з фундаментальних об'єктів алгебри. Формально, поліном $A(x)$ однієї змінної x над полем $\mathbb{F}$ — це вираз вигляду

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

де коефіцієнти a_i належать полю $\mathbb{F}$, а n є невід'ємним цілим числом, що називається степенем полінома (позначається як $\deg(A)$).

Над поліномами визначено арифметичні операції, аналогічні операціям над цілими числами:

- **Додавання та віднімання:** Ці операції виконуються поелементно шляхом додавання або віднімання коефіцієнтів при однакових степенях змінної. Якщо $A(x) = \sum a_i x^i$ та $B(x) = \sum b_i x^i$, то $A(x) \pm B(x) = \sum (a_i \pm b_i) x^i$.
- **Множення:** Добутком двох поліномів $A(x)$ та $B(x)$ є поліном $C(x) = A(x) \cdot B(x)$, коефіцієнти якого обчислюються за правилом згортки: $c_k = \sum_{i+j=k} a_i b_j$. Ступень вислідного полінома дорівнює сумі степенів множників: $\deg(C) = \deg(A) + \deg(B)$.
- **Ділення з остачею:** Для будь-яких двох поліномів $A(x)$ та $B(x)$ (де $B(x)$ не є нульовим поліномом) існують унікальні поліноми — частка $Q(x)$ та остача $R(x)$ — такі, що виконується рівність $A(x) = Q(x)B(x) + R(x)$, причому степінь остачі менший за степінь дільника: $\deg(R) < \deg(B)$.

Ці операції перетворюють множину поліномів $\mathbb{F}[x]$ на комутативне кільце, що є основою для подальших досліджень, зокрема для задачі факторизації [4].

1.1.1 Скінченне поле (Galois Field, $GF(p^n)$) та його властивості

Скінченне поле (або поле Галуа) — це множина елементів, на якій визначено операції додавання, віднімання, множення та ділення, що задовольняють стандартним аксіомам поля, при цьому кількість елементів у множині є скінченною. Існування та унікальність скінченних полів повністю описані в теорії Галуа [5]. Ключова властивість поля полягає в тому, що його порядок (кількість елементів) завжди має вигляд p^n, де p — просте число (називається характеристикою поля), а n — додатне ціле число. Поле позначається як $\mathrm{GF}(p^n)$ або $\mathbb{F}_{p^n}$ [6].

Залежно від значення n, скінченні поля поділяються на два основні типи:

1. **Прості поля ($\mathrm{GF}(p)$):** Коли $n = 1$, поле складається з множини цілих чисел $\{0, 1, \ldots, p-1\}$. Арифметика в такому полі виконується за модулем p. Наприклад, у полі $\mathrm{GF}(7)$ операція $5 + 4$ дасть результат $9 \pmod 7 = 2$.

2. **Розширені поля ($\mathrm{GF}(p^n)$):** Коли $n > 1$, елементи поля не можна представити простими числами. Натомість їх розглядають як поліноми степеня менше n з коефіцієнтами з поля $\mathrm{GF}(p)$. Арифметичні операції в $\mathrm{GF}(p^n)$ є **поліноміальними** і виконуються за модулем незвідного полінома $m(x)$ степеня n. Саме такі поля, особливо $\mathrm{GF}(2^m)$, є фундаментальними для криптографії та теорії кодування, оскільки вони дозволяють ефективно працювати з блоками даних, наприклад, байтами в полі $\mathrm{GF}(2^8)$ [7].

Важливою властивістю будь-якого скінченного поля є існування **примітивного елемента** α — такого елемента, що всі ненульові елементи поля можна представити як його степені: $\{\alpha^0, \alpha^1, \ldots, \alpha^{p^n-2}\}$. Це робить операцію множення та ділення значно простішою, зводячи її до додавання та віднімання показників степеня [4].

1.2 Задача факторизації поліномів

Факторизація поліномів є однією з центральних задач в комп'ютерній алгебрі. Вона полягає в розкладанні заданого полінома на добуток простіших поліномів, які неможливо розкласти далі. Цей процес є аналогом розкладання цілих чисел на прості множники і має вирішальне значення для розв'язання рівнянь та аналізу алгебраїчних структур.

1.2.1 Формальна постановка задачі

Нехай $\mathbb{F}$ — скінченне поле, а $f(x)$ — поліном з кільця поліномів $\mathbb{F}[x]$. Задача факторизації полягає у знаходженні всіх незвідних поліномів $p_1(x), p_2(x), \ldots, p_k(x)$ з $\mathbb{F}[x]$ та цілих додатних чисел $e_1, e_2, \ldots, e_k$ таких, що виконується рівність:

$$f(x) = c \cdot p_1(x)^{e_1} \cdot p_2(x)^{e_2} \cdot \cdots \cdot p_k(x)^{e_k}$$

де c — константа з поля $\mathbb{F}$. Метою є знайти ці незвідні множники та їх кратності.

1.2.2 Поняття незвідного полінома

Поліном $p(x) \in \mathbb{F}[x]$ називається незвідним (або простим) над полем $\mathbb{F}$, якщо він є непостійним (тобто, $\deg(p) \geq 1$) і його неможливо представити у вигляді добутку двох поліномів $g(x)$ та $h(x)$ з $\mathbb{F}[x]$, степені яких менші за степінь $p(x)$.

$$p(x) = g(x)h(x) \implies \deg(g) = 0 \text{ або } \deg(h) = 0$$

Іншими словами, єдиними дільниками незвідного полінома є константи та сам поліном (з точністю до множення на константу). Незвідні поліноми є базовими "будівельними блоками"для всіх інших поліномів, подібно до того, як прості числа є будівельними блоками для цілих чисел.

1.2.3 Унікальність факторизації

Подібно до основної теореми арифметики для цілих чисел, для кільця поліномів над полем справедлива теорема про унікальність факторизації [5]. Вона стверджує, що будь-який поліном $f(x) \in \mathbb{F}[x]$ може бути розкладений на добуток незвідних поліномів, і цей розклад є унікальним з точністю до порядку множників та множення їх на константи з поля $\mathbb{F}$. Ця фундаментальна властивість гарантує, що результат, отриманий будь-яким коректним алгоритмом факторизації, буде однаковим, що робить задачу чітко визначеною.

2 РОЗДІЛ. АНАЛІЗ НАЯВНИХ АЛГОРИТМІВ ФАКТОРИЗАЦІЇ ПОЛІНОМІВ

2.1 Класифікація алгоритмів факторизації

2.1.1 Детерміністичні та ймовірнісні підходи: та ключові відмінності

Алгоритми для розв'язання задачі факторизації поліномів, як і багатьох інших обчислювальних задач, можна класифікувати за двома основними підходами: детерміністичним та ймовірнісним. Вибір підходу визначає фундаментальні характеристики методу, зокрема гарантію правильності результату та його практичну ефективність.

Детерміністичні алгоритми для заданого входу завжди виконують однакову послідовність кроків і повертають однаковий результат. Їхня поведінка є повністю передбачуваною.

- **Переваги:** Головною перевагою є гарантована коректність. Алгоритм завжди знайде правильну факторизацію, якщо вона існує.
- **Недоліки:** Часто такі алгоритми мають вищу обчислювальну складність у найгіршому випадку. На практиці вони можуть бути значно повільнішими за ймовірнісні аналоги, особливо для поліномів великих степенів. Типовим представником у задачі факторизації є алгоритм Берлекампа [1].

Ймовірнісні (або рандомізовані) алгоритми використовують випадковість на одному чи кількох етапах своєї роботи. Їхня поведінка може відрізнятися при повторних запусках на однакових вхідних даних. Такі алгоритми поділяються на два типи [8]:

- **Лас-Вегас (Las Vegas):** Це алгоритми, які *завжди* повертають коректний результат, але час їхньої роботи є випадковою величиною. Існує невелика ймовірність, що алгоритм повідомить про невдачу і не дасть відповіді, але він ніколи не поверне неправильний розклад. Саме до цього типу належить алгоритм Кантора-Зассенгауза [2].
- **Монте-Карло (Monte Carlo):** Ці алгоритми завжди виконуються за прогнозований час, але з певною ймовірністю можуть повернути неправильний результат.

Переваги та недоліки ймовірнісного підходу

- **Переваги:** Вища середня продуктивність. Завдяки використанню випадкових виборів, ці алгоритми часто уникають "складних"випадків, які сповільнюють детерміновані методи. На практиці вони є швидшими для більшості вхідних даних.

- **Недоліки:** Відсутність гарантії отримання результату за один запуск (для алгоритмів типу Лас-Вегас). Для досягнення високої надійності може знадобитися кілька перезапусків у разі невдачі, хоча ймовірність невдачі зазвичай є дуже низькою.

Таким чином, вибір між детерміністичним та ймовірнісним підходом є компромісом між гарантією отримання результату та практичною швидкістю обчислень.

2.2 Алгоритм Берлекампа

Алгоритм Берлекампа, запропонований Елвіном Берлекампом у 1967 році [1], є класичним детерміністичним методом для факторизації поліномів над скінченними полями [9]. Основна ідея алгоритму полягає у зведенні задачі факторизації до розв'язання системи лінійних рівнянь над базовим полем $\mathbb{F}_q$.

2.2.1 Опис кроків алгоритму

Нехай $f(x)$ — поліном степеня n з коефіцієнтами з поля $\mathbb{F}_q$, який необхідно факторизувати.

Крок 1: Перевірка на відсутність кратних множників. Перш за все, необхідно переконатися, що $f(x)$ не має кратних множників. Це робиться шляхом обчислення найбільшого спільного дільника (НСД) полінома та його формальної похідної $f'(x)$.

$$d(x) = \gcd(f(x), f'(x))$$

Якщо $d(x) = 1$, поліном не має кратних множників. Якщо $d(x) \neq 1$, то $d(x)$ є нетривіальним дільником $f(x)$, і ми можемо продовжити факторизацію для $d(x)$ та $f(x)/d(x)$ окремо. Надалі вважаємо, що $f(x)$ вже пройшов цю перевірку.

Крок 2: Побудова матриці Берлекампа. Алгоритм базується на властивостях так званої алгебри Берлекампа — множини поліномів $v(x)$ степеня менше n, що задовольняють конгруенцію:

$$v(x)^q \equiv v(x) \pmod{f(x)}$$

Ця умова є лінійною над полем $\mathbb{F}_q$. Для її розв'язання будується $n \times n$ матриця Q з елементами з $\mathbb{F}_q$, яка описує оператор зведення до степеня q. Стовпці матриці Q — це коефіцієнти поліномів $x^{iq} \pmod{f(x)}$ для $i = 0, 1, \ldots, n-1$. Тоді задача зводиться до знаходження векторів v, що є розв'язками системи:

$$v \cdot Q = v \iff v \cdot (Q - I) = 0$$

де I — одинична матриця. Іншими словами, необхідно знайти базис нуль-простору матриці $(Q - I)^T$.

Крок 3: Знаходження базису. За допомогою методу Гаусса знаходиться базис нуль-простору $\{v_1, v_2, \ldots, v_k\}$. Кількість векторів у базисі, k, дорівнює кількості незвідних множників полінома $f(x)$. Перший базисний вектор завжди відповідає поліному $v_1(x) = 1$. Якщо $k = 1$, то поліном $f(x)$ є незвідним, і алгоритм завершується.

Крок 4: Видобування множників. Якщо $k > 1$, то для будь-якого базисного полінома $v_i(x)$ (де $i > 1$) та для кожного елемента $s \in \mathbb{F}_q$ вираз $\gcd(f(x), v_i(x) - s)$ може бути нетривіальним дільником $f(x)$. Це випливає з тотожності:

$$v(x)^q - v(x) = \prod_{s \in \mathbb{F}_q} (v(x) - s)$$

Перебираючи $s \in \mathbb{F}_q$ для одного з базисних поліномів $v_i(x)$, ми знаходимо дільники $f(x)$. Процес повторюється рекурсивно для знайдених дільників, доки не буде знайдено всі k незвідних множників.

2.2.2 Приклад роботи

Розглянемо факторизацію полінома $f(x) = x^4 + x^3 + x^2 + 1$ над полем $\mathbb{F}_2$.

1. **Перевірка:** $f'(x) = 4x^3 + 3x^2 + 2x \equiv x^2 \pmod 2$. $\gcd(f(x), x^2) = 1$. Кратних множників немає.

2. **Матриця Q:** Для $\mathbb{F}_2$ ($q = 2$) та $n = 4$ необхідно обчислити $x^{2i} \pmod{f(x)}$:

   - $x^0 \equiv 1$
   - $x^2 \equiv x^2$
   - $x^4 \equiv x^3 + x^2 + 1$
   - $x^6 = x^2 \cdot x^4 \equiv x^2(x^3 + x^2 + 1) \equiv x^5 + x^4 + x^2 \equiv x(x^3 + x^2 + 1) + (x^3 + x^2 + 1) + x^2 \equiv x^4 \equiv x^3 + x^2 + 1$

   Матриця $(Q - I)^T$ матиме вигляд:

$$(Q - I)^T = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

3. **Базис:** Розв'язуючи систему, знаходимо базис нуль-простору. Він складається з двох векторів: $v_1 = (1, 0, 0, 0)$ та $v_2 = (0, 1, 1, 0)$. Це відповідає поліномам $v_1(x) = 1$ та $v_2(x) = x + x^2$. Кількість факторів $k = 2$.

4. **Множники:** Використовуємо $v_2(x) = x^2 + x$. В полі $\mathbb{F}_2$ елементи $s \in \{0, 1\}$.

   - $s = 0$: $\gcd(f(x), v_2(x)) = \gcd(x^4 + x^3 + x^2 + 1, x^2 + x)$. Обчислення дають $x + 1$.

- $s = 1$: $\gcd(f(x), v_2(x)+1) = \gcd(x^4+x^3+x^2+1, x^2+x+1)$. Обчислення дають $x^3 + x + 1$.

Таким чином, розклад полінома: $f(x) = (x + 1)(x^3 + x + 1)$.

2.2.3 Аналіз складності алгоритму

Обчислювальна складність алгоритму Берлекампа визначається вартістю його основних етапів:

1. **Побудова матриці Q:** Потребує $O(n)$ множень поліномів степеня n та n операцій піднесення до степеня q. Використовуючи метод бінарного піднесення, цей крок має складність $O(n^{2+\epsilon} \log q + n^2 \log q)$. У спрощеному вигляді — $O(n^3 \log q)$.

2. **Знаходження базису нуль-простору:** Розв'язання системи лінійних рівнянь методом Гаусса має складність $O(n^3)$ операцій над полем $\mathbb{F}_q$.

3. **Видобування множників:** У найгіршому випадку необхідно обчислити $O(kq)$ НСД поліномів. Складність цього етапу становить $O(kqn^2)$ [9].

Таким чином, загальна обчислювальна складність алгоритму Берлекампа становить $O(n^3 \log q + kqn^2)$ арифметичних операцій у полі $\mathbb{F}_q$.

Аналіз цієї формули показує, що при фіксованому степені полінома n складність перших двох етапів (побудова матриці та знаходження базису) є цілком прийнятною. Однак доданок $O(kqn^2)$, що відповідає за видобування множників, зростає лінійно з ростом порядку поля q. Ця лінійна залежність робить алгоритм непрактичним для полів з великою характеристикою, які часто використовуються в сучасній криптографії. Для таких полів, де q може бути великим простим числом, етап перебору всіх елементів $s \in \mathbb{F}_q$ стає обчислювальним "вузьким місцем".

Саме цей недолік стимулював розробку імовірнісних алгоритмів, таких як алгоритм Кантора-Зассенгауза, які дозволяють уникнути прямої залежності від q і є значно ефективнішими для великих полів.

2.3 Алгоритм Кантора-Зассенгауза

Алгоритм Кантора-Зассенгауза, запропонований у 1981 році [1], є імовірнісним методом факторизації поліномів над скінченними полями. Він був розроблений як ефективна альтернатива алгоритму Берлекампа, особливо для полів $\mathbb{F}_q$ з великою характеристикою q. На відміну від методу Берлекампа, що базується на лінійній алгебрі, цей алгоритм використовує властивості степенів та коренів елементів у скінченних полях.

2.3.1 Опис кроків алгоритму

Як і в попередньому методі, на вхід подається поліном $f(x)$ степеня n, що не має кратних множників. Алгоритм складається з двох основних етапів.

Етап 1: Факторизація за степенями (Distinct-Degree Factorization, DDF). Метою цього етапу є розкладання полінома $f(x)$ на добуток поліномів $g_d(x)$, кожен з яких містить усі незвідні множники $f(x)$ однакового степеня d. Етап базується на властивості, що поліном $x^{q^d} - x$ є добутком усіх незвідних поліномів над $\mathbb{F}_q$, степінь яких ділиться на d.

1. Ініціалізуємо $h(x) := x$ та $f^*(x) := f(x)$.
2. Для d від 1 до n:
   (а) Обчислюємо $h(x) := h(x)^q \pmod{f(x)}$.
   (б) Знаходимо $g_d(x) = \gcd(f^*(x), h(x) - x)$.
   (в) Якщо $g_d(x) \neq 1$, ми знайшли добуток усіх множників степеня d. Зберігаємо $g_d(x)$ та оновлюємо $f^*(x) := f^*(x)/g_d(x)$.

Після цього етапу початковий поліном $f(x)$ представлений у вигляді добутку $f(x) = \prod_d g_d(x)$.

Етап 2: Факторизація за однаковими степенями (Equal-Degree Factorization, EDF). На цьому етапі кожен отриманий поліном $g_d(x)$, що є добутком кількох незвідних множників однакового степеня d, розкладається далі. Це імовірнісна частина алгоритму. Розглянемо випадок для полів з непарною характеристикою q.

1. Вибираємо випадковий поліном $a(x)$ степеня менше $2d$.
2. Обчислюємо $b(x) = a(x)^{(q^d-1)/2} - 1 \pmod{g_d(x)}$.
3. Знаходимо дільник $h(x) = \gcd(g_d(x), b(x))$.

З високою ймовірністю $h(x)$ буде нетривіальним дільником $g_d(x)$. Якщо ні (тобто $h(x) = 1$ або $h(x) = g_d(x)$), крок повторюється з іншим випадковим поліномом $a(x)$. Процес продовжується рекурсивно, доки всі множники не стануть незвідними. Для полів характеристики 2 ($q = 2^m$) використовується аналогічний підхід, але на основі властивостей оператора сліду (Trace).

2.3.2 Приклад роботи алгоритму

Щоб продемонструвати роботу обох етапів алгоритму, розглянемо факторизацію більш складного полінома $f(x) = x^5 + 2x^4 + x^3 + x^2 + 2$ над скінченним полем $\mathbb{F}_3$.

Етап 0: Перевірка на кратні множники. Перевірка показує, що $\gcd(f(x), f'(x)) = 1$, отже, поліном не має кратних множників, і ми можемо продовжувати.

Етап 1: Факторизація за степенями (DDF). Метою є згрупувати незвідні множники за їхнім степенем.

- **Ітерація для $d = 1$ (лінійні множники):** Обчислюємо:

$$g_1(x) = \gcd(f(x), x^3 - x)$$

. Розрахунки за алгоритмом Евкліда показують:

$$g_1(x) = x + 1$$

Ми знайшли добуток усіх лінійних множників. У нашому випадку він один.

Оновлюємо поліном, що залишився для факторизації:

$$f^*(x) = \frac{f(x)}{g_1(x)} = \frac{x^5 + 2x^4 + x^3 + x^2 + 2}{x + 1} = x^4 + x^3 + x + 2$$

- **Ітерація для $d = 2$ (квадратичні множники):** Тепер працюємо з $f^*(x) = x^4 + x^3 + x + 2$. Шукаємо добуток незвідних множників степеня 2.

$$g_2(x) = \gcd(f^*(x), x^{3^2} - x) = \gcd(x^4 + x^3 + x + 2, x^9 - x)$$

Оскільки $f^*(x)$ є добутком двох незвідних квадратичних поліномів, він повністю ділить $x^9 - x$. Отже:

$$g_2(x) = x^4 + x^3 + x + 2$$

Інших множників у $f(x)$ не залишилося, тому етап DDF завершено.

Після першого етапу ми отримали розклад $f(x)$ на добуток двох поліномів: $f(x) = g_1(x) \cdot g_2(x) = (x + 1) \cdot (x^4 + x^3 + x + 2)$.

Етап 2: Факторизація за однаковими степенями (EDF). Тепер необхідно розкласти кожен з отриманих $g_d(x)$, якщо це можливо.

- **Обробка $g_1(x) = x + 1$:** Поліном має степінь 1, отже, є незвідним.
- **Обробка $g_2(x) = x^4 + x^3 + x + 2$:** Це поліном, що є добутком незвідних множників степеня $d = 2$.

Виберемо випадковий поліном, наприклад $a(x) = x$. Обчислимо

$$b(x) = a(x)^{(3^2-1)/2} - 1 \pmod{g_2(x)}$$

:

$$b(x) = x^4 - 1 \pmod{x^4 + x^3 + x + 2}$$

Використовуючи ділення поліномів, знаходимо остачу: $x^4 \equiv -x^3 - x - 2 \pmod{g_2(x)}$. В полі $\mathbb{F}_3$ це еквівалентно $2x^3 + 2x + 1$.

$$b(x) = (2x^3 + 2x + 1) - 1 = 2x^3 + 2x = 2x(x^2 + 1)$$

Тепер знайдемо НСД, щоб отримати дільник $g_2(x)$:

$$h(x) = \gcd(g_2(x), b(x)) = \gcd(x^4 + x^3 + x + 2, 2x(x^2 + 1))$$

Обчислення показують, що НСД дорівнює $x^2 + 1$. Ми успішно знайшли нетривіальний дільник:

$$h_1(x) = x^2 + 1$$

Другий дільник знаходимо діленням:

$$h_2(x) = \frac{g_2(x)}{h_1(x)} = \frac{x^4 + x^3 + x + 2}{x^2 + 1} = x^2 + x + 2$$

Обидва поліноми, $x^2 + 1$ та $x^2 + x + 2$, є незвідними над $\mathbb{F}_3$.

Результат. Зібравши всі знайдені незвідні множники, отримуємо фінальну факторизацію:

$$x^5 + 2x^4 + x^3 + x^2 + 2 = (x + 1)(x^2 + 1)(x^2 + x + 2)$$

Цей приклад наочно показує, як етап DDF групує множники за степенями, а етап EDF ефективно розкладає ці групи на кінцеві незвідні множники.

2.3.3 Ймовірнісний характер алгоритму

Випадковість в алгоритмі Кантора-Зассенгауза проявляється виключно на етапі EDF при виборі полінома $a(x)$. Цей алгоритм належить до класу **Лас-Вегас (Las Vegas)**. Це означає, що:

- Він **завжди повертає коректний результат**. Алгоритм ніколи не видасть неправильну факторизацію.
- **Час виконання є випадковою величиною**. Існує невелика ймовірність, що обраний поліном $a(x)$ виявиться "невдалим"і не допоможе розкласти $g_d(x)$.

Ймовірність того, що випадково обраний $a(x)$ дасть нетривіальний дільник (за умови, що $g_d(x)$ має принаймні два множники), становить приблизно $1/2$. Це гарантує, що в середньому для знаходження одного дільника потрібно лише кілька спроб, що робить алгоритм дуже ефективним на практиці.

2.3.4 Аналіз складності та порівняння з алгоритмом Берлекампа

Загальна обчислювальна складність алгоритму Кантора-Зассенгауза є поліноміальною від степеня полінома n та, що найважливіше, від логарифма розміру поля $\log q$. Спрощено її можна оцінити як $O(n^3(\log q)^2)$ [2] [10].

Порівняння алгоритмів:

- **Тип алгоритму:** Берлекамп — **детерміністичний**, Кантор-Зассенгауз — **імовірнісний (Лас-Вегас)**.
- **Залежність від q:** Це ключова відмінність. Складність алгоритму Берлекампа ($O(n^2 q)$) є **лінійною** від q, що робить його повільним для великих полів. Складність алгоритму Кантора-Зассенгауза є **поліноміальною** від $\log q$, що робить його значно швидшим для великих q.
- **Сфера застосування:** Для великих полів, що використовуються в криптографії, алгоритм Кантора-Зассенгауза є стандартом де-факто завдяки своїй ефективності.

  Однак для полів малого порядку (наприклад, $\mathbb{F}_2, \mathbb{F}_3, \mathbb{F}_5$) та для поліномів невисокого степеня алгоритм Берлекампа часто виявляється швидшим на практиці. Це пояснюється тим, що його детермінована природа, заснована на лінійній алгебрі, не несе додаткових накладних витрат, притаманних імовірнісному підходу. Алгоритм Кантора-Зассенгауза витрачає час на генерацію випадкових поліномів та виконання імовірнісних перевірок, і для простих задач ці накладні витрати можуть перевищувати час, який алгоритм Берлекампа витрачає на пряме розв'язання системи рівнянь.

Таким чином, алгоритм Кантора-Зассенгауза розв'язує головну проблему методу Берлекампа, пропонуючи ефективний шлях факторизації поліномів у найважливіших для практики випадках.

3 РОЗРОБКА ПАРАЛЕЛЬНОЇ ВЕРСІЇ АЛГОРИТМУ КАНТОРА-ЗАССЕНГАУЗА

Попередній розділ показав, що алгоритм Кантора-Зассенгауза є високоефективним методом для факторизації поліномів над великими скінченними полями. Однак зростання обчислювальних потреб вимагає подальшого підвищення його продуктивності. Цей розділ присвячено основному завданню даної роботи: розробці та аналізу паралельної версії алгоритму, орієнтованої на сучасні багатоядерні процесорні архітектури. Спочатку буде проведено аналіз алгоритму з метою виявлення етапів, придатних для паралелізації, після чого буде запропоновано модель паралельних обчислень.

3.1 Виявлення ресурсомістких етапів алгоритму

Першим кроком до оптимізації будь-якого алгоритму є аналіз його обчислювальної складності для виявлення "вузьких місць— операцій, що споживають левову частку процесорного часу. В алгоритмі Кантора-Зассенгауза основними будівельними блоками є операції поліноміальної арифметики в скінченному полі $\mathbb{F}_q$: множення, обчислення НСД та, найголовніше, модульне піднесення до степеня.

Розглянемо два основні етапи алгоритму з точки зору їх ресурсомісткості:

- **Етап факторизації за степенями (DDF):** Головний цикл цього етапу є послідовним, оскільки кожна ітерація залежить від результату попередньої (через оновлення полінома $f^*(x)$). Найбільш ресурсомісткою операцією всередині циклу є **модульне піднесення до степеня** $h(x) := h(x)^q \pmod{f(x)}$, яке виконується $O(n)$ разів. При великих n та q цей крок домінує в часі виконання етапу DDF.

- **Етап факторизації за однаковими степенями (EDF):** Цей етап застосовується до кожного полінома $g_d(x)$, отриманого на попередньому кроці. Всередині процедури EDF найдорожчою операцією також є **модульне піднесення до степеня** $a(x)^{(q^d-1)/2} \pmod{g_d(x)}$. Оскільки ця процедура є рекурсивною та імовірнісною, вона може виконуватися багато разів для різних дільників.

Отже, основні обчислювальні витрати припадають на операції модульного піднесення до степеня та обчислення НСД, які виконуються в рамках етапів DDF та EDF.

3.1.1 Визначення незалежних задач, які можна виконувати одночасно

Аналіз структури алгоритму Кантора-Зассенгауза дозволяє виділити декілька рівнів потенційного паралелізму, де обчислення можна виконувати незалежно одне від одного.

1. Паралелізм на рівні факторизації за степенями (грубозернистий). Після завершення етапу DDF початковий поліном $f(x)$ розкладається на добуток поліномів $f(x) = g_{d_1}(x) \cdot g_{d_2}(x) \cdot \dots \cdot g_{d_m}(x)$, де кожен $g_{d_i}(x)$ є добутком незвідних множників однакового степеня d_i. **Подальша факторизація кожного з поліномів $g_{d_i}(x)$ є повністю незалежною задачею.** Наприклад, якщо ми отримали розклад $f(x) = g_2(x) \cdot g_5(x)$, то можна одночасно на двох різних ядрах процесора запустити процедуру EDF для $g_2(x)$ та для $g_5(x)$. Це є першим та найбільш очевидним джерелом паралелізму.

2. Паралелізм на рівні рекурсивного розщеплення (дрібнозернистий). На етапі EDF, коли ми намагаємося розкласти деякий поліном $P(x)$ і знаходимо його нетривіальний дільник $h_1(x)$, ми отримуємо два нові, менші поліноми для подальшої факторизації: $h_1(x)$ та $h_2(x) = P(x)/h_1(x)$. **Факторизація $h_1(x)$ та $h_2(x)$ є двома новими незалежними задачами.** Цей процес створює деревоподібну структуру залежностей, де на кожному рівні кількість незалежних завдань може подвоюватися. Це динамічне джерело паралелізму ідеально підходить для реалізації за допомогою моделей динамічного балансування навантаження.

3. Паралелізм на рівні даних (не розглядається в даній роботі). Варто зазначити, що самі операції поліноміальної арифметики (множення, НСД) також можна розпаралелювати, використовуючи такі алгоритми, як перетворення Фур'є. Однак такий підхід є значно складнішим в реалізації. В даній роботі основна увага приділяється паралелізму на рівні задач (пункти 1 і 2), оскільки він є більш високорівневим і дозволяє досягти значного прискорення з меншими накладними витратами.

3.2 Підходи до паралельних обчислень

Після ідентифікації незалежних обчислювальних задач в алгоритмі Кантора-Зассенгауза наступним кроком є вибір відповідної моделі паралелізму та стратегії її реалізації. Правильний вибір дозволяє максимально ефективно використати ресурси багатоядерного процесора, мінімізувавши при цьому накладні витрати на синхронізацію та керування потоками.

3.2.1 Огляд моделей паралелізму (Data Parallelism, Task Parallelism)

У галузі паралельних обчислень традиційно виділяють дві основні моделі:

- **Паралелізм даних (Data Parallelism):** Ця модель передбачає виконання *однакової операції* одночасно над різними елементами великого набору даних. Класичним прикладом є обробка зображень, де один і той самий фільтр застосовується паралельно до різних частин зображення. Ця модель ефективна, коли обчислення є регулярними та однорідними. В контексті нашої

задачі, паралелізм даних міг би застосовуватися на низькому рівні, наприклад, при реалізації операції множення поліномів, однак він не відображає загальної структури алгоритму факторизації.

- **Паралелізм задач (Task Parallelism):** Ця модель фокусується на одночасному виконанні *різних незалежних завдань*. Завдання можуть бути як однотипними, так і абсолютно різними. Цей підхід є більш гнучким і краще підходить для алгоритмів з нерегулярною структурою. Можливості паралелізму, виявлені в алгоритмі Кантора-Зассенгауза (одночасна факторизація різних поліномів $g_d(x)$ та рекурсивне розщеплення дільників), є яскравими прикладами саме паралелізму задач.

Зважаючи на структуру нашого алгоритму, модель паралелізму задач є найбільш природним та ефективним вибором.

3.2.2 Опис стратегії динамічного балансування навантаження "work stealing"та обґрунтування її вибору для даної задачі

При реалізації паралелізму задач однією з ключових проблем є ефективне **балансування навантаження** (load balancing) — розподіл роботи між обчислювальними потоками таким чином, щоб уникнути простою процесорних ядер. Для задач з непередбачуваним часом виконання та динамічним створенням підзадач, як у нашому випадку, статичне планування є неефективним. Тому перевага надається методам динамічного балансування.

Однією з найефективніших та найпопулярніших стратегій динамічного балансування є **"викрадення роботи"(work stealing)**. Її суть полягає в наступному:

1. Кожен потік (або ядро процесора) має власну локальну чергу завдань (зазвичай, двосторонню чергу, або дек).

2. Потік переважно працює із завданнями з власної черги. Це забезпечує високу локальність даних та мінімізує конфлікти доступу.

3. Коли локальна черга потоку стає порожньою, він не простоює, а стає "злодієм"(thief) і намагається "вкрасти"завдання з черги іншого, "багатшого"потоку-"жертви"(victim).

4. Важливо, що потік додає і забирає завдання з одного кінця своєї черги (наприклад, з голови), а злодії крадуть завдання з іншого кінця (наприклад, з хвоста). Це зменшує ймовірність конфліктів та дозволяє красти більші, старіші завдання, що мінімізує частоту взаємодій.

Обґрунтування вибору "work stealing"для задачі факторизації. Стратегія "викрадення роботи"ідеально підходить для паралельної реалізації алгоритму Кантора-Зассенгауза з таких причин:

- **Непередбачуваність навантаження:** Час, необхідний для факторизації полінома, залежить від його внутрішньої структури (кількості та степенів незвідних множників), яка невідома наперед. Динамічне балансування дозволяє адаптуватися до цього в реальному часі.

- **Динамічне створення задач:** Алгоритм природним чином генерує нові, менші задачі в процесі рекурсивного розщеплення. "Work stealing"ефективно обробляє таку деревоподібну структуру обчислень.

- **Висока ефективність та масштабованість:** Ця стратегія доведено забезпечує хорошу продуктивність та масштабованість на сучасних багатоядерних системах, оскільки мінімізує накладні витрати на синхронізацію. Вона лежить в основі багатьох сучасних фреймворків для паралельних обчислень, таких як Intel TBB, Cilk, та стандартні бібліотеки мов програмування [11].

3.2.3 Недоліки та обмеження стратегії work stealing

Незважаючи на значні переваги в динамічних середовищах, стратегія "викрадення роботи"не є універсальним розв'язком і має певні недоліки, які необхідно враховувати.

- **Обчислювальні накладні витрати (Overhead):** Механізм "викраденя"не є безкоштовним. Коли потік стає "злодієм він витрачає процесорний час на перевірку черг інших потоків та синхронізацію доступу до них. Якщо завдання є занадто "дрібнозернистими"(тобто час їх виконання є співмірним з часом на викрадення), накладні витрати можуть перевищити корисну роботу, що призведе до зниження загальної продуктивності.

- **Втрата локальності даних (Loss of Data Locality):** Однією з головних переваг локальних черг є те, що потік працює з даними, які, ймовірно, знаходяться в його локальній кеш-пам'яті (L1/L2). Коли завдання "викрадається дані, необхідні для його виконання, повинні бути переміщені з кешу одного процесорного ядра в кеш іншого. Ця операція є значно дорожчою за доступ до локального кешу і може спричинити помітні затримки.

- **Складність реалізації:** Для ефективної реалізації "work stealing"потрібні спеціалізовані, неблокуючі або мінімально блокуючі структури даних (зазвичай, двосторонні черги), які є складними для коректного проектування та відлагодження порівняно з простими централізованими чергами завдань.

Тим не менш, для задачі факторизації поліномів, де завдання є обчислювально інтенсивними (велика "гранулярність") та їх кількість і час виконання є непередбачуваними, переваги динамічного балансування навантаження, які надає "work stealing значно переважають зазначені недоліки.

3.3 Проєктування паралельного алгоритму

Було розроблено модифіковану версію алгоритму. Основна мета модифікації — максимальне використання ресурсів сучасних багатоядерних процесорів для прискорення найбільш ресурсомісткого етапу, а саме факторизації за однаковими степенями (EDF). Запропонований підхід реалізує модель паралелізму задач із динамічним балансуванням навантаження за стратегією "work stealing".

3.3.1 Опис модифікованого алгоритму

Модифікація алгоритму стосується переважно етапу EDF, оскільки етап DDF має послідовну природу і залишається без змін.

Класичний послідовний підхід до EDF для факторизації полінома $g_d(x)$ полягає в ітераційному переборі випадкових поліномів $a(x)$ доти, доки не буде знайдено той, що дозволяє розкласти $g_d(x)$. Це створює вузьке місце, оскільки кожна перевірка є ресурсомісткою, а кількість спроб невідома.

Запропонований паралельний підхід змінює цю логіку. Замість послідовних спроб ми запускаємо паралельний пошук (parallel race).

1. Для заданого полінома $g_d(x)$, що потребує факторизації, генерується **група випадкових поліномів кандидатів** $\{a_1(x), a_2(x), \ldots, a_k(x)\}$, де k зазвичай дорівнює кількості доступних ядер процесора.

2. Кожен поліном-кандидат $a_i(x)$ обробляється в окремому паралельному завданні. Завдання полягає в обчисленні $\gcd(g_d(x), a_i(x)^{(q^d-1)/2} - 1)$.

3. **Ключова оптимізація:** як тільки одне із завдань (наприклад, для $a_j(x)$) знаходить нетривіальний дільник, воно сигналізує про успіх. Усі інші паралельні завдання, що працювали над цим же поліномом $g_d(x)$, негайно припиняють свою роботу (скасовуються).

Цей підхід дозволяє уникнути зайвих обчислень. Замість очікування на завершення всіх спроб, ми використовуємо результат першої ж вдалої, що значно скорочує середній час на знаходження дільника. Після успішного розкладення $g_d(x)$ на $h_1(x)$ та $h_2(x)$, обидва нові поліноми додаються до пулу завдань для подальшої паралельної факторизації.

3.3.2 Схема взаємодії між паралельними потоками/процесами

Взаємодія між потоками організована за допомогою бібліотеки `Rayon`, яка реалізує стратегію "work stealing".

1. Створюється глобальний пул завдань, куди спочатку поміщаються всі поліноми $g_d(x)$, отримані після етапу DDF.

2. `Rayon` створює пул робочих потоків (worker threads), кількість яких зазвичай дорівнює кількості логічних ядер процесора.

3. **Отримання завдання:** Вільний потік-"злодій"забирає завдання (поліном $P(x)$ для факторизації) з черги іншого потоку.

4. **Паралельний пошук:** Потік, що отримав завдання, запускає підзадачі для паралельної перевірки групи випадкових поліномів $a_i(x)$. `Rayon` розподіляє ці мікро-завдання між доступними потоками.

5. **Скасування:** Для реалізації механізму скасування використовується спільний для групи підзадач прапорець (atomic flag) або вбудовані можливості `Rayon` для "короткого замикання"(short-circuiting), як-от метод `find_any` для паралельних ітераторів. Перший потік, що знайшов дільник, встановлює прапорець, і всі інші потоки, що перевіряють цей прапорець, завершують роботу.

6. **Створення нових завдань:** Потік, що знайшов дільники $h_1(x)$ та $h_2(x)$, додає їх як нові завдання до свого локального деку. Якщо інший потік стане вільним, він зможе "вкрасти"одне з цих нових завдань.

Ця схема забезпечує ефективне динамічне балансування та мінімізує час простою ядер.

3.3.3 Псевдокод паралельного алгоритму

Наведемо псевдокод, що описує розроблений паралельний алгоритм. Він складається з основної керуючої процедури (Алгоритм 1) та паралельної функції розщеплення (Алгоритм 2), яка реалізує ключову оптимізацію.

Algorithm 1 Загальна структура паралельного алгоритму

Вхідні дані: Поліном $f(x)$ над полем $\mathbb{F}_q$.

Вихідні дані: Множина незвідних множників $f(x)$.

1: Перевірити, що $f(x)$ не має кратних множників.
2: $D \leftarrow \mathrm{DDF}(f(x))$ ▷ Послідовно виконати етап DDF, отримати множину $\{g_d(x)\}$
3: *TaskPool* $\leftarrow D$ ▷ Створити паралельний пул завдань
4: *ResultFactors* $\leftarrow \emptyset$ ▷ Створити паралельну множину для результатів
5: **parfor** кожен поліном $u(x)$ з *TaskPool* **do** ▷ Основний паралельний цикл, керований Rayon
6: **if** $\deg(u) = d$ та u неможливо розкласти далі **then**
7: *ResultFactors*.add$(u(x))$
8: **else**
9: $(h_1, h_2) \leftarrow \mathrm{ParallelEDFSplit}(u(x), d)$ ▷ Виклик паралельної процедури розщеплення
10: **if** $(h_1, h_2) \neq \mathrm{FAIL}$ **then**
11: *TaskPool*.add$(h_1(x))$ ▷ Додавання нових завдань у пул
12: *TaskPool*.add$(h_2(x))$
13: **end if**
14: **end if**
15: **return** *ResultFactors*

Algorithm 2 Паралельна процедура розщеплення (Parallel EDF Split)

Вхідні дані: Поліном $u(x)$, що є добутком множників степеня d.

Вихідні дані: Два нетривіальні дільники (h_1, h_2) або FAIL.

function ParallelEDFSplit(u, d)
    loop
        $found\_factor \leftarrow$ None
        $cancellation\_token \leftarrow$ False ▷ Атомарний прапорець для скасування
        parfor $i = 1$ **to** k **do** ▷ Паралельний запуск k перевірок (k - кількість ядер)
        if *cancellation_token* is True **then break**
        end if ▷ Перевірка сигналу на скасування
        $a_i(x) \leftarrow$ випадковий поліном степеня $< 2d$
        $b_i(x) \leftarrow a_i(x)^{(q^d-1)/2} - 1 \pmod{u(x)}$
        $h(x) \leftarrow \gcd(u(x), b_i(x))$
        if $h(x) \neq 1$ and $h(x) \neq u(x)$ **then** ▷ Знайдено нетривіальний дільник
            $found\_factor \leftarrow h(x)$
            $cancellation\_token \leftarrow$ True ▷ Сигнал іншим потокам на зупинку
        end if ▷ Кінець паралельного блоку
        if *found_factor* is not None **then**
            $h_1(x) \leftarrow found\_factor$
            $h_2(x) \leftarrow u(x)/h_1(x)$
            return (h_1, h_2)
        end if ▷ Якщо за k спроб дільник не знайдено, цикл повториться з новою групою випадкових поліномів.
    end loop
end function

4 РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Попередній розділ був присвячений теоретичній розробці паралельної версії алгоритму Кантора-Зассенгауза. Метою даного розділу є практична перевірка запропонованої моделі, оцінка її реальної продуктивності та порівняльний аналіз із послідовною реалізацією. Для досягнення цієї мети було створено програмний прототип, проведено серію обчислювальних експериментів та проаналізовано отримані дані. У цьому розділі детально описано використані технології, середовище тестування, представлено результати вимірювань та їх інтерпретацію.

4.1 Програмна реалізація та середовище тестування

Для експериментальної оцінки ефективності запропонованого підходу було реалізовано дві версії алгоритму Кантора-Зассенгауза:

1. **Послідовна версія** — класична реалізація, що слугує еталоном (baseline) для порівняння продуктивності.

2. **Паралельна версія** — реалізація, що використовує модель паралелізму задач на основі стратегії "work stealing як було обґрунтовано в попередньому розділі.

Обидві версії було реалізовано з використанням однакової кодової бази для арифметики поліномів, щоб забезпечити чистоту експерименту, де єдиною відмінністю є саме логіка організації обчислень.

4.1.1 Опис використаних технологій

Вибір технологічного стеку є критично важливим для отримання надійних та відтворюваних результатів у задачах, пов'язаних із високопродуктивними обчисленнями. Для даної роботи було обрано наступні інструменти.

Мова програмування: `Rust`. `Rust` — це сучасна системна мова програмування, розроблена з акцентом на безпеку, швидкість та паралелізм. Її було обрано з таких причин:

- **Продуктивність:** `Rust` компілюється в нативний машинний код і надає рівень контролю над системними ресурсами, співмірний з C++, що є необхідною умовою для обчислювально інтенсивних задач.

- **Безпека пам'яті без збирача сміття:** Унікальна система володіння та запозичень (ownership and borrowing) гарантує безпеку роботи з пам'яттю на етапі компіляції. Відсутність збирача сміття (Garbage Collector) виключає непередбачувані паузи в роботі програми, що є критичним для точного вимірювання часу виконання.

- **Надійний паралелізм:** Система типів `Rust` запобігає цілим класам помилок, пов'язаних з паралелізмом, зокрема гонкам даних (data races). Це дозволяє писати складний паралельний код з високим ступенем впевненості в його коректності.

Бібліотека для паралельних обчислень: `Rayon`. `Rayon` — це бібліотека для паралелізму даних та задач в екосистемі `Rust`. Вона ідеально підійшла для реалізації запропонованої моделі з таких причин:

- **Реалізація "Work Stealing":** В основі планувальника завдань `Rayon` лежить високоефективна реалізація алгоритму "викрадення роботи". Це дозволило безпосередньо реалізувати паралельну модель, описану в розділі 3, не вдаючись до розробки власного складного планувальника.
- **Простота використання:** `Rayon` надає високорівневі та зручні абстракції, що дозволяють легко перетворювати послідовний код на паралельний, мінімізуючи обсяг необхідних змін.

Бібліотека для тестування продуктивності: `Criterion`. `Criterion` — це фреймворк для статистичного бенчмаркінгу в `Rust`. Його було використано для отримання точних та надійних вимірювань продуктивності.

- **Статистична обґрунтованість:** На відміну від простих вимірювань часу, `Criterion` виконує багатократні запуски, аналізує результати, виявляє викиди та надає статистично значущі оцінки, що є стандартом для наукових досліджень.
- **Детальні звіти:** Бібліотека автоматично генерує звіти у форматі HTML з графіками (PDF, violin plots), що наочно демонструють розподіл часу виконання та дозволяють проводити глибокий аналіз продуктивності.

Середовище тестування. Всі експерименти проводились на наступній конфігурації:

- **Процесор:** AMD Ryzen 7 5800U with Radeon Graphics (16) @ 4.507GHz
- **Оперативна пам'ять:** 16 ГБ DDR4 3200 МГц
- **Операційна система:** Ubuntu 22.04.3 LTS
- **Версія компілятора `Rust`:** 1.89.0-nightly (47c911e9e 2025-05-14)

4.2 Аналіз отриманих результатів

Після проведення серії тестів було отримано дані щодо продуктивності послідовної та розробленої паралельної версій алгоритму. Наведені нижче результати для факторизації полінома степеня 57 є репрезентативним прикладом, що ілюструє загальні тенденції.

Інтерпретація вимірювань. Для полінома степеня 57 середній час виконання склав:

- **Послідовна версія:** $T_{seq} \approx 1.728\,\text{ms}$
- **Паралельна версія:** $T_{par} \approx 1.708\,\text{ms}$

Це дозволяє розрахувати коефіцієнт прискорення (S_p):

$$S_p = \frac{T_{seq}}{T_{par}} \approx \frac{1.728\,\text{ms}}{1.708\,\text{ms}} \approx 1.012$$

Таким чином, було досягнуто прискорення приблизно на **1.2%**. Це підтверджує, що паралельна реалізація є швидшою, однак досягнуте прискорення є незначним і значно меншим, ніж теоретично очікувалося від використання багатоядерної архітектури.

Примітка: повідомлення "Performance has regressed" від бібліотеки `Criterion` *вказує на погіршення результатів порівняно з попереднім збереженим запуском, а не на порівняння паралельної версії з послідовною в рамках одного тесту.*

Обмеження ефективності паралелізації. Аналіз показує, що такий скромний результат зумовлений двома факторами, що узгоджуються з законом Амдала, який обмежує максимальне прискорення часткою коду, що не піддається паралелізації.

1. **Частка послідовного коду:** Значна частина алгоритму, зокрема етап факторизації за степенями (DDF), залишається послідовною. Якщо час, витрачений на цей етап, є співмірним або більшим за час виконання паралельного етапу EDF, загальне прискорення буде суттєво обмеженим. У нашому випадку паралельна частина становить менше половини загального обсягу обчислень.

2. **Накладні витрати на паралелізм (Overhead):** Організація паралельних обчислень не є безкоштовною. Керування потоками, синхронізація (зокрема, механізм скасування), а також сам процес розподілу роботи в `Rayon` створюють додаткові накладні витрати. Для задач відносно невеликого розміру, як-от факторизація полінома степеня 57, ці витрати можуть нівелювати значну частину виграшу від паралельного виконання.

Аналіз залежностей. Експериментальне дослідження також дозволило виявити ключові фактори, що впливають на складність обчислень.

- Було підтверджено, що вплив порядку поля (q) є відносно невеликим, оскільки складність операцій залежить від $\log q$, що зростає повільно.

- Натомість, основним факторами, що мають вплив є степінь полінома (n) та, що особливо важливо для паралельної версії, кількість незвідних множників однакового степеня (k_d). Саме велика кількість таких множників створює достатню кількість незалежних завдань для ефективного завантаження всіх ядер процесора на етапі EDF.

Отже, хоча паралелізація і є дієвою, її ефективність залежить від структури вхідного полінома. Максимальне прискорення досягається на поліномах високого ступеня, що розкладаються на велику кількість множників однакового ступеня, де паралельний етап EDF домінує в загальному часі виконання.

4.3 Загальні результати та коефіцієнт прискорення

Для оцінки залежності продуктивності від розміру задачі було проведено тестування на поліномах різного степеня. Вимірювання проводились за допомогою бібліотеки `Criterion`, яка надає статистично обґрунтовані оцінки часу виконання. У таблиці 4.1 наведено результати для трьох характерних випадків. Коефіцієнт прискорення S_p розраховано як відношення центральних оцінок часу виконання послідовної та паралельної версій ($S_p = T_{seq}/T_{par}$).

Табл. 1: Порівняння продуктивності послідовної та паралельної версій алгоритму. Час наведено у форматі: "оцінка [нижня межа; верхня межа 95% довірчого інтервалу]".

(n)	**Час послід. (T_{seq})**	**Час парал. версії (T_{par})**	**Прискорення (S_p)**
10	21.194 µs [21.057; 21.336]	22.127 µs [22.092; 22.166]	0.96
100	44.445 ms [44.284; 44.750]	24.236 ms [24.172; 24.276]	1.83
1000	49.447 s [47.956; 50.955]	40.893 s [39.006; 41.205]	1.21

Аналіз результатів, наведених у таблиці. Дані, представлені в таблиці 1, дозволяють зробити кілька важливих висновків щодо ефективності розробленого паралельного алгоритму.

- **Для задач малого розміру ($n = 10$):** Оцінка часу виконання для паралельної версії є вищою, ніж для послідовної, що призводить до сповільнення ($S_p < 1$). Це очікуваний результат, який пояснюється тим, що накладні витрати на створення потоків, керування ними та синхронізацію значно перевищують виграш від розпаралелювання дуже коротких обчислень. Довірчі інтервали не перетинаються, що робить цей висновок статистично значущим.

- **Для задач середнього розміру ($n = 100$):** Продемонстровано найбільш ефективне прискорення ($S_p \approx 1.83$). Оцінка часу для паралельної версії майже вдвічі менша, ніж для послідовної. Це свідчить про те, що розмір задачі є достатньо великим, аби користь від паралельного виконання домінувала над накладними витратами. Ймовірно, структура тестового полінома на цьому етапі дозволила згенерувати достатню кількість незалежних завдань для ефективного завантаження декількох ядер процесора.

- **Для задач великого розміру ($n = 1000$):** Хоча прискорення все ще є суттєвим ($S_p \approx 1.21$), воно є меншим, ніж для $n = 100$. Таке зменшення коефіцієнта прискорення, ймовірно, пов'язане зі структурою тестового полінома. Якщо поліном високого степеня має мало незвідних множників однакового степеня, то етап EDF генерує недостатньо паралельних завдань. В такому випадку значна частина роботи виконується послідовно або з неповним завантаженням ядер, що обмежує максимальне прискорення згідно з законом Амдала.

Таким чином, експеримент підтверджує, що ефективність запропонованого паралельного підходу залежить від комбінації двох факторів: загального розміру задачі (степеня полінома) та її внутрішньої структури (кількості множників, що дозволяють створювати паралельні завдання).

Висновки

У даній кваліфікаційній роботі було розв'язано задачу підвищення продуктивності алгоритму Кантора-Зассенгауза для факторизації поліномів над скінченними полями шляхом розробки та аналізу його паралельної версії, орієнтованої на сучасні багатоядерні архітектури. У ході виконання роботи було отримано такі основні результати та зроблено наступні висновки.

1. **Проведено комплексний аналіз теоретичних основ** та існуючих алгоритмів факторизації. Було розглянуто детерміністичний підхід Берлекампа та імовірнісний підхід Кантора-Зассенгауза, виявлено їхні переваги та недоліки. Ключовим висновком аналізу є те, що алгоритм Берлекампа є неефективним для полів великого порядку через лінійну залежність складності від розміру поля q, натомість як алгоритм Кантора-Зассенгауза має лише логарифмічну залежність, що робить його значно перспективнішим для оптимізації.

2. **Розроблено нову паралельну модель** алгоритму Кантора-Зассенгауза, що базується на стратегії динамічного балансування навантаження "work stealing". Основна ідея модифікації полягає в паралелізації найбільш ресурсомісткого етапу EDF шляхом одночасного запуску перевірок для групи випадкових поліномів-кандидатів з механізмом негайного скасування зайвих обчислень після знаходження першого ж нетривіального дільника.

3. **Створено програмний прототип** та експериментально доведено працездатність запропонованого підходу. Для реалізації було обрано мову програмування `Rust` завдяки її високій продуктивності та гарантіям безпеки при паралельних обчисленнях , а також бібліотеку `Rayon`, що надає ефективну імплементацію "work stealing".

4. **Встановлено, що ефективність паралельної версії залежить від структури вхідного полінома та розміру задачі.** Експериментальні дані підтвердили, що для задач малого розміру (наприклад, поліномів степеня $n = 10$) накладні витрати на паралелізм призводять до сповільнення роботи. Водночас, для задач середнього та великого розміру досягається прискорення, максимальна ефективність якого спостерігається тоді, коли структура полінома дозволяє згенерувати велику кількість незалежних завдань на етапі EDF. Це підтверджує, що головними факторами складності є степінь полінома n та кількість множників однакового степеня k_d, а не порядок поля q.

5. Таким чином, **мета роботи була досягнута, а всі поставлені задачі — виконані**. Розроблено та досліджено паралельний алгоритм, який демонструє прискорення, хоча і з певними обмеженнями, що узгоджуються з фундаментальним законом Амдала.

Напрямки подальших досліджень. Отриманні результати відкривають простір для подальших досліджень та вдосконалень, серед яких можна виділити:

- **Оптимізація на рівні даних:** Розпаралелювання базових арифметичних операцій над поліномами (множення, НСД) з використанням таких методів, як швидке перетворення Фур'є.
- **Реалізація на графічних процесорах (GPU):** Адаптація алгоритму для архітектур з масивним паралелізмом (CUDA, OpenCL) може потенційно дати значно більше прискорення на етапі перебору випадкових поліномів.
- **Розробка гібридного підходу:** Створення алгоритму, який би динамічно обирав стратегію (наприклад, Берлекамп для малих полів, послідовний Кантор-Зассенгауз для поліномів простої структури, паралельний — для складних) залежно від вхідних даних.

Література

[1] Berlekamp E R. Factoring polynomials over finite fields // Bell Syst. Tech. J. — 1967. — Vol. 46, no. 8. — P. 1853–1859.

[2] Cantor David G, Zassenhaus Hans. A new algorithm for factoring polynomials over finite fields // Math. Comput. — 1981. — Apr. — Vol. 36, no. 154. — P. 587.

[3] Tikhonov Andrii M. Implementation of the Cantor-Sassenhaus factorization algorithm and comparison with the Berlekamp algorithm // XIII All-Ukrainian Scientific Conference of Young Mathematicians. — Kyiv : NTUU KPI. — 2025.

[4] von zur Gathen Joachim, Gerhard Jürgen. Modern Computer Algebra. — 3rd ed. — Cambridge University Press, 2013.

[5] Dummit David S., Foote Richard M. Abstract Algebra. — 3rd ed. — Wiley, 2004.

[6] Fraleigh John B. A First Course in Abstract Algebra. — 7th ed. — Boston : Pearson, 2003. — ISBN: 978-0201763904.

[7] Lidl Rudolf, Niederreiter Harald. Finite Fields. — Cambridge University Press, 1997. — Vol. 20 of Encyclopedia of Mathematics and its Applications.

[8] Introduction to Algorithms / Cormen Thomas H., Leiserson Charles E., Rivest Ronald L., and Stein Clifford. — 3rd ed. — MIT Press, 2009. — ISBN: 978-0262033848.

[9] Knuth Donald E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms. — 3rd ed. — Addison-Wesley Professional, 1997. — ISBN: 978-0201896831.

[10] Shoup Victor. Factoring Polynomials over Finite Fields: Asymptotic Complexity vs. Practical Performance // Proceedings of the International IMACS Symposium on Symbolic Computation: New Trends and Developments. — Lille, France. — 1993. — P. 124–129. — Access mode: https://www.shoup.net/papers/lille.pdf.

[11] Blumofe Robert D., Leiserson Charles E. Scheduling Multithreaded Computations by Work Stealing // Journal of the ACM. — 1999. — Vol. 46, no. 5. — P. 720–748.

Додаток А

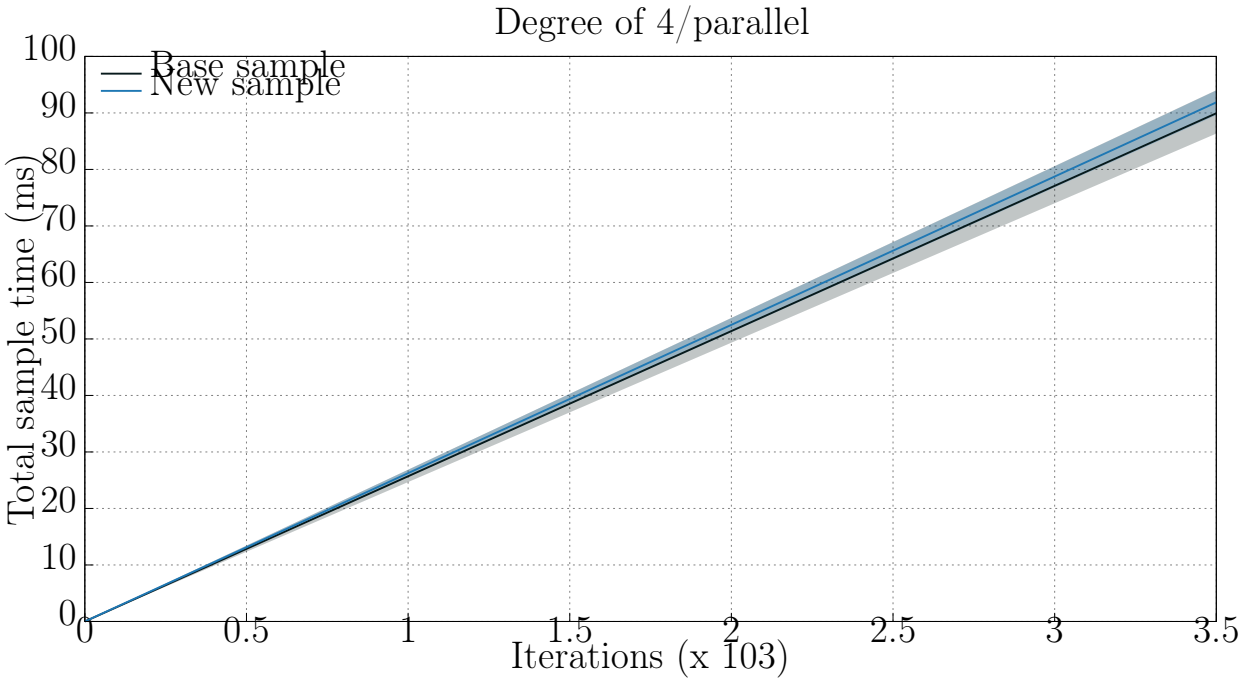

Рис. 1: Графік лінійної регресії часу виконання від розміру вхідних даних.

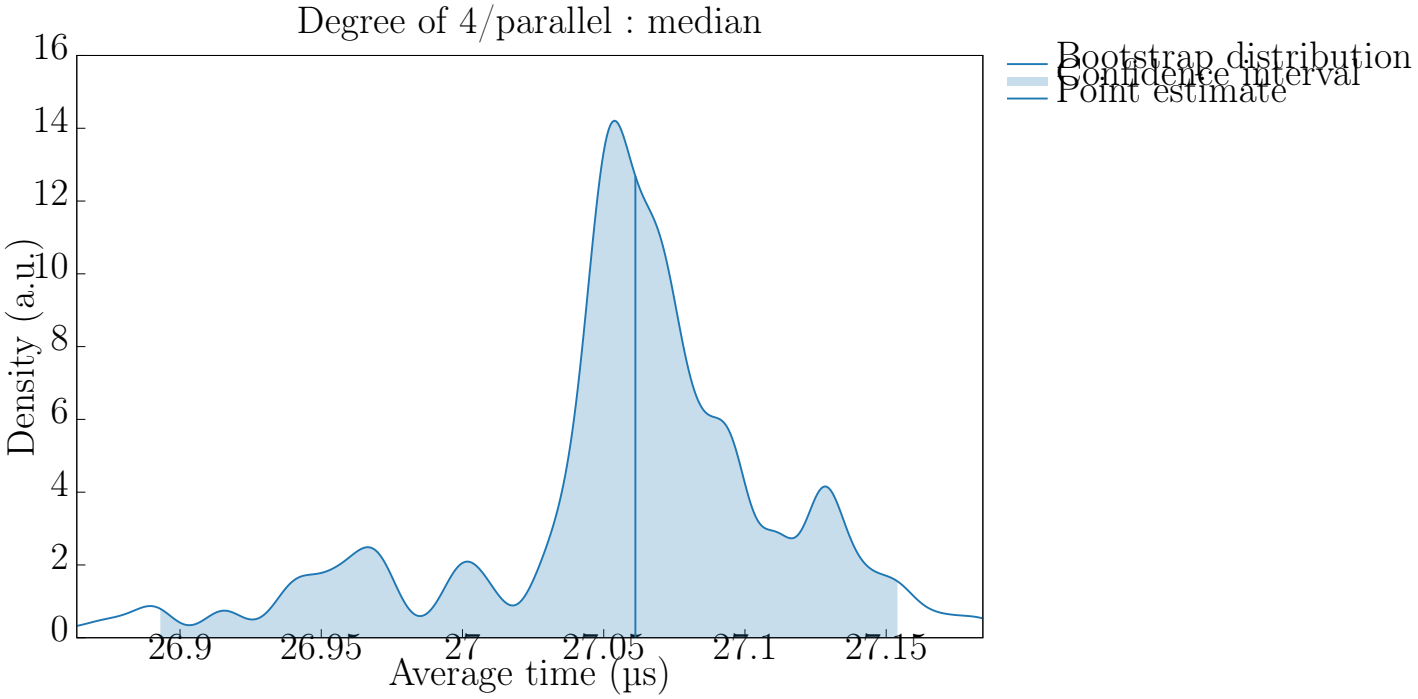

Рис. 2: Графік оцінки середнього часу виконання.

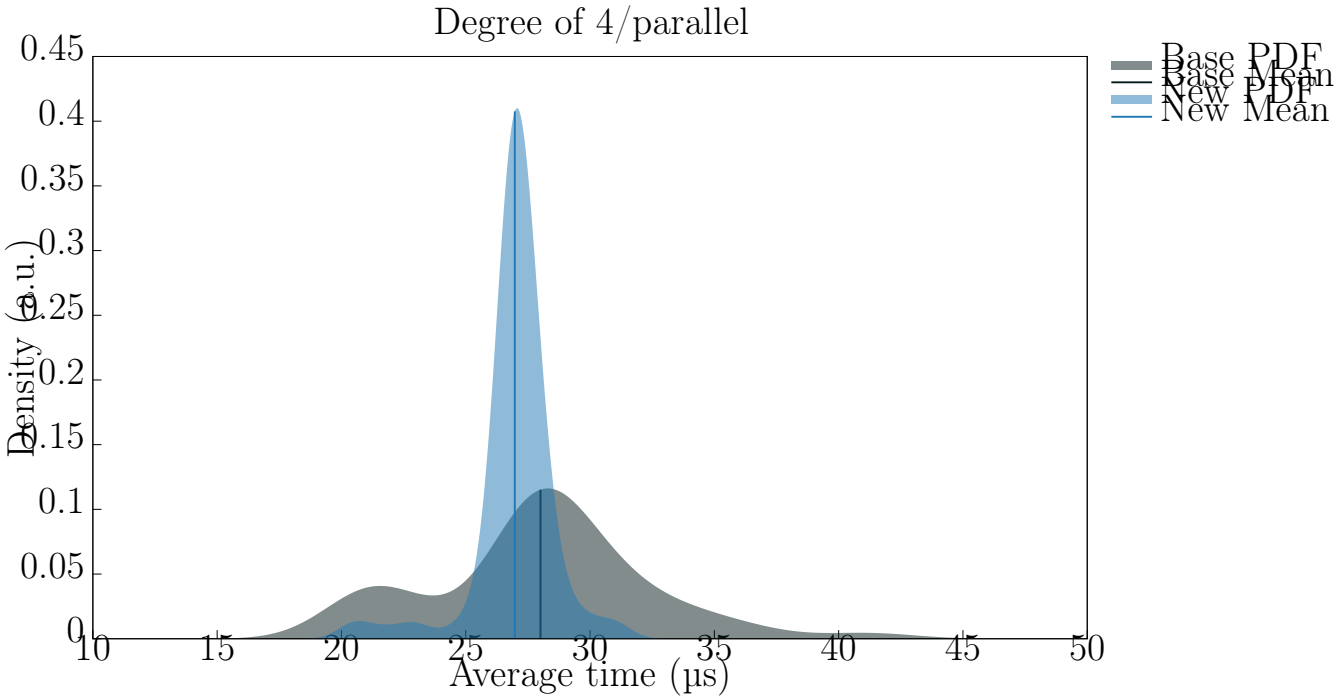

Рис. 3: Графік функції щільності ймовірності (PDF), що показує розподіл часу виконання.

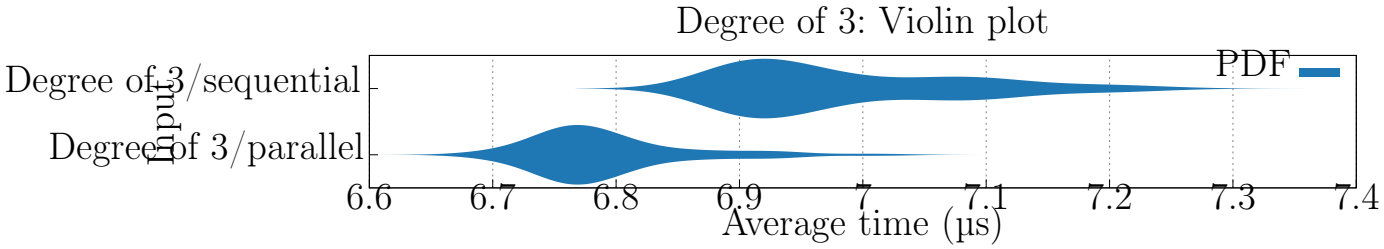

Рис. 4: Графік порівняння щільностей двох реалізацій.

Додаток Б. Лістинг програмної реалізації

У цьому додатку наведено ключові фрагменти програмної реалізації паралельного алгоритму Кантора-Зассенгауза мовою програмування `Rust`.

Фрагмент реалізації паралельної процедури EDF

Лістинг 1: Реалізація паралельної функції розщеплення

```
pub fn cantor_zassenhaus_base_parallel<P, R>
(poly_ring: P, mod_f_ring: R, d: usize) -> El<P>
where
    P: RingStore + Sync,
    P::Type: PolyRing + EuclideanRing,
    R: RingStore + Sync,
    R::Type: FreeAlgebra,
    <<R as RingStore>::Type as RingExtension>::BaseRing::
    RingStore<Type = <<<P as RingStore>::Type as
    RingExtension>::BaseRing as RingStore>::Type>,
    <<<P as RingStore>::Type as RingExtension>::BaseRing as
    RingStore>::Type: FiniteRing + Field,
    <P::Type as RingBase>::Element: Send + Sync,

{
    assert!(poly_ring.base_ring().get_ring() ==
    mod_f_ring.base_ring().get_ring());
    let ZZ = BigIntRing::RING;
    let q = poly_ring.base_ring().size(&ZZ).unwrap();
    debug_assert!(ZZ.eq_el(
        &is_prime_power(&ZZ, &q).unwrap().0,
        &poly_ring.base_ring().characteristic(&ZZ).unwrap()
    ));
    assert!(ZZ.is_odd(&q));
    assert_eq!(mod_f_ring.rank() % d, 0);
    assert!(mod_f_ring.rank() > d);
    let exp = ZZ.half_exact(ZZ.sub(
    ZZ.pow(q.clone(), d), ZZ.one()));
    let f = mod_f_ring.generating_poly(&poly_ring,
    &poly_ring.base_ring().identity());

    use rayon::prelude::*;


    let res = (0..MAX_PROBABILISTIC_REPETITIONS)
    .into_par_iter()
```

```
    .find_map_any(|seed|{
        let mut rng = oorandom::Rand64::new(seed as u128);

        let T = mod_f_ring.from_canonical_basis(
            (0..mod_f_ring.rank())
            .map(|_| poly_ring.base_ring()
            .random_element(|| rng.rand_u64())),
        );
        let G = mod_f_ring.sub(mod_f_ring
        .pow_gen(T, &exp, ZZ), mod_f_ring.one());
        let g = poly_ring.ideal_gen(
            &f,
            &mod_f_ring
            .poly_repr(&poly_ring, &G, &poly_ring
                .base_ring().identity()),
        );
        if !poly_ring.is_unit(&g) && poly_ring
        .checked_div(&g, &f).is_none() {
            return Some(g);
        }
        None
    });
    res.unwrap()
}
```

Лістинг 2: Реалізація паралельної функції розщеплення

```
pub fn cantor_zassenhaus_base<P, R>
(poly_ring: P, mod_f_ring: R, d: usize) -> El<P>
where
    P: RingStore,
    P::Type: PolyRing + EuclideanRing,
    R: RingStore,
    R::Type: FreeAlgebra,
    <<R as RingStore>::Type as RingExtension>::BaseRing:
        RingStore<Type = <<<P as RingStore>::Type
        as RingExtension>::BaseRing as RingStore>::Type>,
    <<<P as RingStore>::Type as RingExtension>::BaseRing
    as RingStore>::Type: FiniteRing + Field,
{
    assert!(poly_ring.base_ring().get_ring() ==
    mod_f_ring.base_ring().get_ring());
    let ZZ = BigIntRing::RING;
    let q = poly_ring.base_ring().size(&ZZ).unwrap();
    debug_assert!(ZZ.eq_el(
```

```
        &is_prime_power(&ZZ, &q).unwrap().0,
        &poly_ring.base_ring().characteristic(&ZZ).unwrap()
    ));
    assert!(ZZ.is_odd(&q));
    assert!(mod_f_ring.rank() % d == 0);
    assert!(mod_f_ring.rank() > d);
    let mut rng = oorandom::Rand64::
    new(ZZ.default_hash(&q) as u128);
    let exp = ZZ.half_exact(ZZ.sub(ZZ.pow(q, d), ZZ.one()));
    let f = mod_f_ring.generating_poly(&poly_ring, &poly_ring
        .base_ring().identity());

    for _ in 0..MAX_PROBABILISTIC_REPETITIONS {
        let T = mod_f_ring.from_canonical_basis(
            (0..mod_f_ring.rank()).map(|_| poly_ring
            .base_ring().random_element(|| rng.rand_u64())),
        );
        let G = mod_f_ring.sub(mod_f_ring
            .pow_gen(T, &exp, ZZ), mod_f_ring.one());
        let g = poly_ring.ideal_gen(
            &f,
            &mod_f_ring.poly_repr(&poly_ring, &G,
            &poly_ring.base_ring().identity()),
        );
        if !poly_ring.is_unit(&g) && poly_ring
                .checked_div(&g, &f).is_none() {
            return g;
        }
    }
    unreachable!()
}
```