

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Програмної інженерії _____

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)

Дослідження несиметричних криптографічних алгоритмів в умовах використання
квантових комп'ютерів. Факторизація

Виконав: студент 2 курсу, групи ПЗМ-17-2
спеціальності 121- Інженерія програмного забезпечення
Освітньо-наукової програми
Інженерія програмного забезпечення

Керівник Островський А.М.
проф. Качко О.Г.

Допускається до захисту
Зав. кафедри, проф. _____

З.В.Дудар

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121-Інженерія програмного забезпечення

освітньо-наукова програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« ____ » _____ 20 ____ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Островському Альберту Миколайовичу

(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження несиметричних криптографічних алгоритмів в умовах використання квантових комп'ютерів. Факторизація

затверджена наказом по університету від “18” квітня 2019 р № 546ст

заповнюється вручну після отримання наказу

2. Термін подання студентом роботи до екзаменаційної комісії

05 червня 2019 р.

3. Вихідні дані до роботи алгоритми факторизації числа, пояснювальна записка. розробити клієнт-серверну систему для факторизації числа за допомогою алгоритма Шора. Використовувати ОС Windows, середовище розробки IntelliJ IDEA 2017 та Visual Studio, мови об'єктно-орієнтованого програмування Java8, C#, мова програмування для квантових обчислень Q#

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз предметної області, дослідження особливостей квантових обчислень, дослідження алгоритмів факторизації, аналіз розроблених реалізацій, висновки. Додатки: а) слайди презентації; б) код реалізованих алгоритмів факторизації в) електронні матеріали до проекту на CD

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Структура квантового комп'ютера, Діаграма квантової схеми алгоритму факторизації Шора, Зациклення ряду у алгоритмі Полларда, Схема функціональних цілей java.util.concurrent, Діаграма основних executors класів з java.util.concurrent, Діаграма послідовності, Діаграма класів, Структура Q# програми для моделювання алгоритма Шора Графічний інтерфейс користувача, Діаграми залежності часу виконання від розміра числа для факторизації реалізацій алгоритма Шора, Ферма та Полларда окремо та разом, слайди презентації

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Качко О.Г.		6 травня 2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	25 лютого 2019	виконано
2.	Дослідження парадигми квантових обчислень	11 березня 2019	виконано
3.	Проектування системи	18 березня 2019	виконано
4.	Реалізація алгоритмів факторизації	1 квітня 2019	виконано
5.	Тестування і налагодження програми	8 квітня 2019	виконано
6.	Підготовка пояснювальної записки	22 квітня 2019	виконано
7.	Спецчастина	6 травня 2019	виконано
8.	Підготовка презентації та доповіді	20 травня 2019	виконано
9.	Попередній захист	27 травня 2019	виконано
10.	Нормоконтроль, рецензування	5 липня 2019	виконано
11.	Занесення диплома в електронний архів	5 липня 2019	виконано
12.	Допуск до захисту у зав. кафедри	5 липня 2019	виконано
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання 11 лютого 2019 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Качко О.Г.
(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи магістра: 85 сторінок, 30 рисунків, 9 таблиць, 27 джерел, 4 додатку.

Об'єктом дослідження є криптостійкість несиметричних та квантових криптографічних алгоритмів, які базуються на складності факторизації числа.

Метою роботи є дослідження атак, пов'язаних з факторизацією, в умовах використання квантових комп'ютерів.

Методи дослідження базуються на теорії квантової механіки та квантових обчислень, криптоаналізі, об'єктно-орієнтованому підході, технології Java з використанням спеціальної бібліотеки для реалізації розподілених обчислень `java.util.concurrent` та мови для використання квантових обчислень Q#.

У результаті роботи були досліджені переваги та недоліки квантових обчислень, квантовий алгоритм Шора для факторизації чисел, розроблено програмний додаток для дослідження та порівняння алгоритмів факторизації, а саме квантового алгоритму Шора, Ферма та ρ -Полларда.

КВАНТОВИЙ КОМП'ЮТЕР, КВАНТОВИЙ АЛГОРИТМ, АЛГОРИТМ ШОРА, КВАНТОВИЙ ПАРАЛЕЛІЗМ, JAVA, Q#, ФАКТОРИЗАЦІЯ, ДИСКРЕТНИЙ ЛОГАРИФМ, NP ЗАДАЧА.

The object of research is the cryptostability of asymmetric and quantum cryptographic algorithms, which are based on complexity of factorization of number.

The aim is to investigate the attacks associated with factorization, in the use of quantum computers.

Development methods are based on on the theory of quantum mechanics and quantum computing, cryptanalysis, object-oriented approach, Java technology using a special library for the implementation of distributed computing `java.util.concurrent` and language for the use of quantum computing Q #.

QUANTUM COMPUTER, QUANTUM ALGORITHM, SHOR'S ALGORITHM, QUANTUM PARALLELISM, JAVA, Q#, FACTORIZATION, DISCRETE LOGARITHM, NP COMPLEXITY.

ЗМІСТ

Вступ.....	4
1 Аналіз предметної області.....	6
1.1 Односторонні функції.....	8
1.2 Алгоритми факторизації.....	12
1.3 NP задачі.....	16
2 Дослідження особливостей квантових обчислень.....	21
2.1 Особливості квантових алгоритмів.....	22
2.2. Проблеми квантових обчислень.....	25
2.2.1 Проблема когерентизації.....	26
2.2.2 Проблема клонування квантового стану регістра.....	27
2.2.3 Проблема вимірювання поточного стану системи.....	29
2.2.4 Ймовірнісний характер квантових обчислень.....	30
2.3 Фізична реалізація квантових комп'ютерів.....	31
2.4 Квантові комп'ютери сьогодні.....	32
3 Дослідження алгоритмів факторизації.....	38
3.1 Алгоритм факторизації Шора.....	38
3.2 Алгоритм факторизації Ферма.....	45
3.3 Реалізація методу факторизації ρ -Полларда.....	46
3.4 Опис програмних засобів.....	47
3.5 Проектування програмної системи.....	50
4 Аналіз отриманих результатів.....	55
4.1 Аналіз алгоритма Шора для класичного комп'ютера.....	55
4.2 Аналіз алгоритма Ферма.....	57
4.3 Аналіз алгоритма ρ -Полларда.....	60
4.4 Аналіз змодельованого алгоритма Шора.....	62
4.5 Порівняльний аналіз алгоритмів факторизації.....	65
Висновки.....	67

Перелік джерел.....	69
Додаток А Слайди презентації	71
Додаток Б Код реалізованих алгоритмів факторизації	79
Додаток В Підготовлені тези на 23-ий міжнародний молодіжний форум "Радіoeлектроніка та молодь у ХХІ столітті"	83
Додаток Г CD з матеріалами роботи	

ВСТУП

Квантові обчислення і квантовий зв'язок - самі ці поняття були винайдені буквально 30 років тому, і перші роботи вчених навіть не брали в наукові журнали: говорили, що фантастика, а не наука. Сьогодні ж квантові системи не тільки існують, але і продаються за гроші, створюючи і вирішуючи нові проблеми безпеки, в основному в сфері криптографії.

Ми живемо в світі радіохвиль і електромагнітних сигналів. Wi-Fi, GSM, супутникове телебачення і GPS, точний час і FM-тюнер - лише деякі з повсякденних технологій, в яких використовуються електромагнітні хвилі. Звичайно, в список потрібно включити і всі види комп'ютерів, від гігантських дата-центрів до смартфонів і ноутбуків. Одна з особливостей електромагнітних сигналів полягає в тому, що їх досить легко виміряти, тобто перехопити. Саме тому, практично все вище перелічене сьогодні забезпечено технологією шифрування, що захищає інформацію від читання і зміни сторонніми. При цьому запасного каналу зв'язку зазвичай немає, і розробники криптосистем блискуче вирішили складну проблему - як домовитися про секретний ключі шифрування, коли весь процес переговорів можуть слухати сторонні? Саме рішення цієї проблеми лежить в основі всіх сучасних систем захисту, і саме йому імовірно покладуть край квантові комп'ютери. Суть рішення в тому, щоб використовувати так звані односторонні функції, які дуже швидко рахуються в одну сторону, а в іншу, тобто рішення зворотної задачі, може складати декілька років, а може і більше [1].

Назва квантових систем точно передає зміст - їх робота заснована на квантових ефектах, таких як суперпозиція і сплутування (зчеплення) мікрочастинок. Принциповою відмінністю квантового комп'ютера від звичайного є те, що його операційна одиниця - кубіт (квантовий біт) може перебувати в стані невизначеності, тобто в декількох станах одночасно. Звучить заплутано, ще складніше на практиці, але, як показали роки досліджень, це працює. Квантовий комп'ютер сильно відрізняється від класичного і навряд чи придатний для гри в

"Тетріс", зате він незмірно швидше вирішує імовірнісні та оптимізаційні завдання. Серед речей, які можна радикально прискорити квантовими обчисленнями, - оптимізація маршрутів транспорту, секвенування ДНК, пророкування біржових котировань і підбір криптографічних ключів. Правда, відповідь теж завжди буде імовірнісною, навіть вважати його за комп'ютер є складною проблемою, але, зробивши кілька досить швидких прогонів однієї і тієї ж задачі, можна прийти до однієї-єдиної, правильної відповіді: в нашому випадку - ключу шифрування.

На сьогодні квантові комп'ютери привертають до себе все більше і більше уваги. Все більше грошей йде на розвинення цієї галузі. Найбільші ІТ-гіганти такі, як Google, Microsoft, International Business Machines (IBM), D-Wave Systems змагаючись між собою швидко наближують людство до перших квантових комп'ютерів, які будуть здатні вирішувати реальні проблеми. На сьогодні вже існує багато реалізацій, які здатні вирішувати незначні задачі за допомогою квантових обчислень. Окрім розробки квантових комп'ютерів, дуже велика увага приділена також і до постквантових алгоритмів криптографії. Отже, квантові комп'ютери у недалекому майбутньому це реальність.

Метою даної роботи є дослідження криптографічних алгоритмів, які базуються на факторизації числа, а саме квантового алгоритма Шора, алгоритма Ферма та алгоритма ρ -Полларда для факторизації числа. Методи розробки базуються на технології Java з використанням спеціальної бібліотеки для реалізації розподілених обчислень `java.util.concurrent` та мові для квантових обчислень Q#. Також використовуються наступні методи: паралельні обчислення, об'єктно-орієнтований підхід до розробки програмної системи, односторінкові додатки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

В 1977 р. троє вчених Рональд Райвест (Ronald Linn Rivest), Ади Шамір (Adi Shamir) та Леонард Адлеман (Leonard Adleman) з Массачусетського Технологічного Інституту (MIT) опублікували в журналі Scientific American новий алгоритм шифрування, заснований на ідеї двохключового шифрування, названий за першими літерами прізвищ авторів методом RSA [2]. У цьому методі відомим параметром служить деяке ціле число n великої довжини (зазвичай від 1024 до 4096 бітів), що є добутком двох простих чисел p і q . Ці числа p і q являються секретними параметрами методу, і для злому системи RSA досить знайти множники p і q , тобто виконати розкладання числа n на прості множники. На момент опублікування алгоритму RSA було відомі лише невелика кількість алгоритмів факторизації, найвідомішим з яких був метод Ферма. Ці методи дозволяли на той день факторизувати числа, що складаються не більше ніж з 25 - 30 цифр. Тому використання в якості n натурального числа, що має понад 100 десяткових знаків, гарантовано забезпечувало безпеку шифрування цим методом. Самі творці методу запропонували всієї математичної громадськості для тестового злому 129-значне десяткове число, пообіцявши за його розкладання умовне винагороду в 100 доларів. Масла у вогонь підлила також опублікована в 1977 р в журналі Sci.Amer. стаття відомого математика і популяризатора Мартіна Гарднера "A new kind of cipher that would take millions of years to break" ("Новий алгоритм шифрування, для злому якого буде потрібно мільйони років"). Виклик, кинутий всьому світу, не залишився непоміченим. У змагання з пошуку швидких алгоритмів факторизації включилося величезна кількість людей, серед яких були відомі математики, фахівці з теорії чисел і криптографи. В результаті цієї гонки були створені кілька алгоритмів факторизації, що збагатили алгоритмічну теорію чисел низкою чудових ідей. В кінці 80-х р XX століття були розроблено основні методи факторизації. Фактично, з'явився новий напрямок у сучасній алгоритмічній теорії чисел - дослідження методів перевірки простоти цілих чисел і методів знаходження дільників непростих

(складових) цілих чисел. В середині березня 1991 року RSA лабораторія оголосила конкурс з грошовими винагородами за злом. Суть конкурсу полягала в тому, що було викладено 54 числа різної довжини від (100 десяткових знаків до 2048 біт) і потрібно було розкласти ці числа на прості множники. Конкурс закінчився в 2007 році, але ентузіасти досі намагаються розкласти числа, які залишилися. Основні з цих чисел наведені в таблиці 1.1.

Сама ж історія з розкладанням 129-значного числа творців методу RSA закінчилася в 1994 р, коли за допомогою алгоритму квадратичного решета, реалізованого в мережі колективом авторів, очолюваним А.Ленстром, було виконано розкладання цього числа на множники. Ця процедура вимагала колосальних зусиль. Була задіяна мережа, що складається з 1600 комп'ютерів, які пропрацювавши 220 днів, підготували систему лінійних рівнянь, що містить більше 0,5 млн невідомих.

Таблиця 1.1 – Основні числа RSA Factoring Challenge

Число	Десятичних цифр	Двоїчних чисел	Винагорода	Факторизовано
RSA-100	100	330		01.03.1991 А.К. Ленстра
RSA-129	129	426	\$100	26.03.1994 А.К. Ленстра
RSA-576	174	576	\$10,000	03.12.2003 Д. Франк
RSA-200	200	663		09.05.2005 Д. Франк
RSA-704	212	704	\$30,000	02.07.2012 Shi Bai, Emmanuel Thomé and Paul Zimmermann
RSA-768	270	896	\$50,000	-
RSA-1024	309	1024	\$100,000	-
RSA-1536	463	1536	\$150,000	-
RSA-2048	617	2048	\$200,000	-

Потім ця система була вирішена за допомогою суперкомп'ютера за 2 дні обчислень. В даний час дослідження в області побудови швидких алгоритмів факторизації інтенсивно ведуться в усьому світі. Щорічно проводяться десятки конференцій з цієї тематики, досягаються нові рекорди факторизації довгих чисел, досліджуються відомі проблеми алгоритмічної теорії чисел і ставляться нові проблеми. В кінці 2009 р колективом європейських вчених, очолюваним

Торштенем Кляйн'юнгом, був встановлений новий рекорд по розкладанню 768-бітного натурального числа за допомогою методу решета числового поля [3]. Попередній рекорд в 512-біт був встановлений у 2000 році, тобто перехід від 512-бітових до 768-бітовим числах зайняв майже 10 років. Тому наступний рекорд в 1024 біта при збереженні колишніх темпів зростання досліджень планується виконати не раніше, ніж в 2020 р. RSA-2048 має 2048 бітів (617 десяткових знаків). Це найбільше з RSA-чисел і за нього покладено приз в 200000 доларів США. Найбільше факторизоване RSA-число має довжину 768 біт (232 десяткових знаків) і RSA-2048 може не бути розкладено протягом довгих років, до значного поліпшення обчислювальних потужностей і просувань в факторизації цілих чисел.

1.1 Односторонні функції

У роботі "New Directions in Cryptography" Діффі і Хеллман запропонували принципово новий спосіб організації секретного зв'язку без попереднього обміну ключами, так зване шифрування з відкритим ключем. При цьому для шифрування та розшифрування використовуються різні ключі, і знання одного з них не дає практичної можливості визначити другий. В результаті ключ шифрування може бути відкритим без втрати стійкості шифру, і лише ключ розшифрування повинен триматися одержувачем у секреті, тому криптосистеми з відкритим ключем називають асиметричними (несиметричними) криптосистемами [4].

Базовим поняттям криптографії з відкритим ключем є поняття односторонньої функції (one-way function) [5]. Згідно з заданого аргументу $x \in X$ легко обчислити значення цієї функції $F(x)$, в той же час значення x з $F(x)$ визначити важко, тобто немає алгоритму для вирішення цього завдання за поліноміальний час роботи. Теоретично x по відомим значенням $F(x)$ можна знайти завжди, перевіряючи по черзі всі можливі значення x до тих пір, поки

відповідне значення $F(x)$ не співпадає з заданим. Однак практично при значній розмірності безлічі X такий підхід неможливий.

Існування таких односторонніх функцій досі є відкритою проблемою. З їхнього існування випливає твердження, що класи складності P та NP не рівні. Сучасна асиметрична криптографія ґрунтується на припущенні, що односторонні функції все-таки існують [6].

Односторонньою функцією називається функція $F(x): X \rightarrow Y, x \in X$, що володіє двома властивостями:

- існує поліноміальний алгоритм обчислення значень $y = F(x)$;
- не існує поліноміального алгоритму інвертування функції $F(x) = y$.

Зауважимо, що до сих пір не доведено існування односторонніх функцій. Використання їх в якості основи асиметричних алгоритмів шифрування допустимо тільки до тих пір, поки не знайдені ефективні алгоритми, що виконують знаходження односторонніх функцій за поліноміальний час.

Прикладом кандидата на звання односторонньої функції є модульне піднесення до ступеня (1.1):

$$F(x) \equiv a^x \bmod p, \quad (1.1)$$

де a – примітивний елемент поля $GF(p)$,

p – велике просте число.

Те, що ця функція може бути ефективно обчислена навіть при розрядності параметрів в кілька сотень знаків, можна показати на прикладі: a^{25} можна обчислити за допомогою шести операцій множення (множенням вважається і зведення в квадрат). Число 25 в двійковій системі числення записується як 11001, тому $25 = 2^4 + 2^3 + 2^0$ (1.2):

$$a^{25} \bmod p \equiv (a^{16} a^8 a) \bmod p \equiv (((a^2 a)^2)^2 a) \bmod p, \quad (1.2)$$

де a – примітивний елемент поля $GF(p)$,
 p – велике просте число.

Завдання обчислення функції, яка є оберненою модульному зведенню в ступінь, називається завданням дискретного логарифмування. На сьогоднішній день невідомо жодного ефективного алгоритму обчислення дискретних логарифмів великих чисел.

Одностороння функція як функція шифрування непридатна, оскільки, якщо $F(x)$ – зашифроване повідомлення x , то ніхто, в тому числі і законний одержувач, не зможе відновити x . Обійти цю проблему можна за допомогою односторонньої функції з секретом (one-way trapdoor function). Розглянемо приклад: функція $E_k: X \rightarrow Y$, що має обернену функцію $D_k: Y \rightarrow X$, однак дізнатися обернену функцію тільки за E_k без знання секрету k неможливо. Односторонньою функцією з секретом k називається функція $E_k: X \rightarrow Y$, що залежить від параметра k і володіє трьома властивостями:

- при будь-якому k існує поліноміальний алгоритм обчислення значень $E_k(x)$;
- при невідомому k не існує поліноміального алгоритму інвертування E_k ;
- при відомому k існує поліноміальний алгоритм інвертування E_k .

Функцію E_k можна використовувати для шифрування інформації, на обернену їй функцію D_k – для розшифрування, так як при всіх $x \in X$ справедливо $D_k(E_k(x)) = x$. При цьому мається на увазі, що той, хто знає, як зашифровувати інформацію, зовсім не обов'язково повинен знати, як розшифровувати її. Так само як і у випадку з односторонньою функцією, питання про існування односторонніх функцій з секретом відкритий.

Для практичної криптографії знайдено кілька функцій – кандидатів на звання односторонньої функції з секретом [7]. Для них друга властивість не доведена, проте відомо, що задача інвертування еквівалентна розв'язку важкої математичної задачі.

Найбільш важливою односторонньою функцією, яка використовується у сучасній криптографії з відкритим ключем, – це розкладення на множники або факторизація цілих чисел.

Застосування односторонньої функцій з секретом в криптографії дозволяє:

- організовувати обмін зашифрованими повідомленнями з використанням тільки відкритих каналів зв'язку, тобто відмовитися від секретних каналів зв'язку для попереднього обміну ключами;

- включати при розкритті шифру складну задачу і тим самим підвищувати стійкість шифру;

- вирішувати нові криптографічні завдання, відмінні від шифрування (електронний цифровий підпис та ін.);

- підвищувати розмір секретного ключа, щоб зробити зашифровані повідомлення більш надійними, в умовах постійного підвищення обчислювальних потужностей у комп'ютерів.

Стійкість більшості сучасних асиметричних алгоритмів базується на двох математичних проблемах, які на даному етапі є складнообчислюваними:

- дискретного логарифмування в кінцевих полях;

- факторизація великих чисел.

Оскільки на сьогоднішній день не існує ефективних алгоритмів вирішення цих завдань або їх рішення вимагає залучення великих обчислювальних ресурсів або часових витрат, ці математичні задачі знайшли широке застосування в побудові асиметричних алгоритмів.

Таким чином, односторонні функції є фундаментальними інструментами криптографії, персональної ідентифікації, аутентифікації, криптоаналізу та інших областей захисту даних. Хоча існування таких функцій, як і раніше залишається недоведеною гіпотезою, існує кілька претендентів, які витримали десятиліття пильного вивчення. Багато з них є невід'ємною частиною більшості телекомунікаційних систем, а також систем електронної комерції та інтернет-банкінгу.

1.2 Алгоритми факторизації

Завдання пошуку ефективних способів розкладання цілих чисел на множники цікавила математиків з давніх часів, особливо фахівців в області теорії чисел [8]. Існують припущення про те, що Ферма був одним з перших, хто запропонував метод розкладання, що полягає в тому, щоб представити число у вигляді різниці квадратів $n = x^2 - y^2$, а потім, обчислюючи найбільший спільний дільник від n та $x - y$, спробувати знайти нетривіальний дільник n . Даний спосіб дозволяє знаходити два мало відрізняючихся за величиною дільника числа швидше, ніж простий перебір дільників.

Далі Лежандр виявив, що при такому підході досить отримати порівняння $x^2 \equiv y^2 \pmod{n}$, і використовував для цього ланцюгові дроби. Також Ейлером і Гауссом були запропоновані деякі способи знаходження чисел, пов'язаних цим порівнянням. Одним з ключових моментів в розвитку факторизації цілих чисел було створення алгоритму RSA, що відновило інтерес вчених в даному напрямку, так як мало практичне застосування в області шифрування. На популярність завдання факторизації також вплинула публікація в 1977 році в журналі *Scientific American* Мартіна Гарднера "Новий алгоритм шифрування, для злому якого будуть потрібні мільйони років". Таку гучну назву було сприйнято як виклик всьому математичній спільноті. В результаті цієї гонки було запропоновано кілька нових і нестандартних ідей факторизації.

Як правило, на вхід алгоритмів факторизації подається число n , яке належить до N , яке необхідно факторизувати, що складається з $N = \lceil \log_2 n \rceil + 1$ символів (якщо n представлено в двійковому вигляді)[9]. При цьому алгоритм шукає перший простий дільник, після чого, при необхідності, можна запустити алгоритм заново для подальшої факторизації. Також, перш ніж починати факторизацію великого числа, слід переконатися в тому, що воно не просте. Для цього достатньо пройти тест числа на простоту. Це завдання детерміновано вирішуване за поліноміальний час.

Залежно від складності алгоритми факторизації можна розбити на дві групи. Перша група - експоненціальні алгоритми, складність яких експоненціально залежить від довжини вхідних параметрів (тобто від довжини N самого числа в бінарному вигляді). Друга група - субекспоненціальні алгоритми.

Питання про існування алгоритму факторизації з поліноміальною складністю на класичному комп'ютері є однією з важливих відкритих проблем сучасної теорії чисел.

Розглянемо експоненціальні алгоритми:

- перебір можливих дільників[10]. Один з найпростіших і очевидних алгоритмів факторизації, що полягає в тому, щоб послідовно ділити факторизуєме число n на натуральні числа від 1 до кореня з n . Формально досить ділити тільки на прості числа в цьому інтервалі, проте, для цього необхідно знати їх множину. На практиці складається таблиця простих чисел і проводиться перевірка невеликих чисел. Для дуже великих чисел алгоритм не використовується в силу низької швидкості роботи;

- метод факторизації Ферма[11]. Ідея алгоритму полягає в пошуку таких чисел A і B , що факторизуєме число n представимо у вигляді: $n = A^2 - B^2 = (A-B)(A+B)$. Як і метод пробного поділу, зазвичай не застосовується на практиці для факторизації великих чисел, так як має експонентну складність. Метод примітний тим, що його можливо реалізувати без операції ділення, а тільки лише з операціями додавання і віднімання. Слід так само відзначити, що якщо $n = pq$, за умови того, що p і q - прості числа не сильно відрізняються за величиною, то метод Ферма факторизує n досить швидко;

- ρ -алгоритм Полларда. Алгоритм Полларда є імовірнісним алгоритмом, що дозволяє знаходити дільник складеного числа n , що працює зі складністю, що залежить лише від величини дільника, але не величини факторизуємого числа n . Це обумовлює зручність застосування даного алгоритму в тих випадках, коли інші алгоритми, складність яких залежить від n , стають неефективні. Примітний також тим, що існує варіант реалізації такого алгоритму, при якому досить в пам'яті зберігати всього 3 цілих числа;

– алгоритм Ленстри [12]. Слід зазначити, що незважаючи на відносно непогану ефективність серед експоненційних алгоритмів, в алгоритмі Ленстра є необхідність неодноразово обчислювати квадратний корінь в одному з кроків алгоритму, що, безумовно, є більш трудомістким, ніж додавання чи віднімання;

– алгоритм Полларда — Штрассена. Даний алгоритм має оцінку складності схожу з методом квадратичних форм Шенкса (що є найкращою серед детермінованих алгоритмів факторизації), однак вимагає виділення більшої кількості пам'яті. Він може використовуватися безпосередньо для факторизації не дуже великих цілих чисел, а також як допоміжний алгоритму в субекспоненціальних методі Діксона;

– метод квадратичних форм Шенкса. На відміну від алгоритму Полларда - Штрассена не вимагає виділення великих обсягів пам'яті, до того ж має досить прості розрахункові формули;

– P-1 алгоритм Полларда. Незважаючи на експонентну оцінку складності, алгоритм у всіх випадках швидко знаходить невеликі прості дільники n тому, що вони є статечно-гладкими для невеликої кордону гладкості. У практичних завданнях даний алгоритм зазвичай використовується до застосування субекспоненціальних алгоритмів факторизації, щоб відокремити невеликі прості дільники числа n ;

– метод Лемана. В даний час алгоритм являє швидше історичний, ніж практичний інтерес, так як цей алгоритм був першим детермінованим алгоритмом зі складністю виконання швидше, ніж корінь з n [13];

Розглянемо субекспоненціальні алгоритми (для позначення складності прийнята L-нотація наведена у формулі (1.3):

$$L_n(a, c) := O(\exp((c + o(1))(\log n)^\alpha (\log \log n)^{1-\alpha})), \quad (1.3)$$

де n — число для факторизації,

$0 < \alpha < 1$ та c — деякі константи.

– алгоритм Діксона. В 20-х р. XX сторіччя Морис Крайчик (1882-1957), узагальнюючи теорему Ферма запропонував замість пар чисел, що задовольняють рівняння $x^2 - y^2 = n$, шукати пари чисел, що задовольняють більш загальному рівнянню $x^2 \equiv y^2 \pmod{n}$. Крайчик знайшов кілька корисних для вирішення фактів. У 1981 р Джон Діксон опублікував розроблений їм метод факторизації, який використовує ідеї Крайтчіка, і розрахував його обчислювальну складність;

– факторизація методом безперервних дробів. Алгоритм розкладання цілих чисел на прості множники. Це алгоритм загального вигляду, придатний для факторизації довільного цілого m . Метод безперервних дробів розроблений на основі алгоритму Крайчик і використовує безперервний дріб, що сходиться до кореня з добутку k та m для деякого цілого позитивного числа k . На основі методу безперервних дробів був побудований алгоритм Діксона, за схемою якого потім був розроблений метод квадратичного решета;

– метод квадратичного решета. Даний метод з використанням декількох многочленів ефективний і досить легко можливо реалізувати на комп'ютері. Є підстави вважати, що він є найкращим з відомих алгоритмів факторизації для $n < 10^{110}$ (Не рахуючи метод факторизації за допомогою еліптичних кривих, який в деяких випадках може працювати швидше. Варто так само відзначити, що для чисел $n > 10^{110}$ алгоритми решета числового поля спрацюють швидше, ніж метод квадратичного решета;

– факторизація Ленстра за допомогою еліптичних кривих[8], де p — найменше просте, яке ділить n . Параметри a, x, y вибираються випадково. Значення w, v слід вибирати емпірично, розглянувши деяку серію зростаючих значень. На практиці при заданих n, v, w алгоритм полягає в тому, щоб виконати алгоритм з однієї кривої. Це повторюється до тих пір, поки не розкладеться на множники або поки час, відведений для алгоритму, не закінчиться. Існують модифікації алгоритму, що дозволяють працювати з декількома кривими одночасно;

– решето числового поля. Є найбільш ефективним алгоритмом факторизації чисел довжиною понад 110 десяткових знаків Метод є узагальненням спеціального методу решета числового поля: тоді як останній дозволяє факторизувати числа

тільки деякого спеціального виду, загальний метод працює на множині цілих чисел, за винятком ступенів простих чисел (які факторизуються тривіально витяганням коренів);

Передбачувана велика обчислювальна складність завдання факторизації лежить в основі криптостійкості деяких алгоритмів шифрування з відкритим ключем, таких як RSA. Більш того, якщо відомий хоча б один з параметрів ключів RSA, то система зламується однозначно, крім того, існує безліч алгоритмів відновлення всіх ключів в системі, володіючи якимись даними

1.3 NP задачі

В рамках класичної теорії здійснюється класифікація завдань за класами складності (P-складні, NP-складні, експоненціально складні і ін.). До класу P відносяться завдання, які можуть бути вирішені за час, поліноміально залежний від обсягу вихідних даних, за допомогою детермінованої обчислювальної машини (наприклад, машини Т'юрінга), а до класу NP[14] - завдання, які можуть бути вирішені за поліноміально виражений час за допомогою недетермінованої обчислювальної машини, тобто машини, наступний стан якої не завжди однозначно визначається попередніми. Роботу такої машини можна уявити як, та, яка розгалужується на кожній неоднозначності процес: завдання вважається вирішеною, якщо хоча б одна гілка процесу прийшла до відповіді. Інше визначення класу NP: до класу NP відносяться завдання, вирішення яких за допомогою додаткової інформації поліноміальної довжини, даної нам згори, ми можемо перевірити за поліноміальний час. Зокрема, до класу NP відносяться всі завдання, вирішення яких можна перевірити за поліноміальний час. Клас P міститься в класі NP. Класичним прикладом NP-задачі є задача про комівояжера.

Оскільки клас P міститься в класі NP, приналежність того чи іншого завдання до класу NP часто відображає наше поточне уявлення про способи вирішення

даного завдання і носить незакінчений характер. У загальному випадку немає підстав вважати, що для того чи іншого NP-завдання не може бути знайдено P-рішення. Питання про можливу еквівалентності класів P і NP (тобто про можливість знаходження P-рішення для будь-якої NP-завдання) вважається багатьма одним з основних питань сучасної теорії складності алгоритмів. Відповідь на це питання не знайдений досі. Сама постановка питання про еквівалентність класів P і NP можлива завдяки введенню поняття NP-повних задач. NP-повні задачі складають підмножина NP-задач і відрізняються тою властивістю, що всі NP-завдання можуть бути тим чи іншим способом зведені до них. З цього випливає, що якщо для NP-повної задачі буде знайдено P-рішення, то P-рішення буде знайдено для всіх завдань класу NP. Прикладом NP-повної задачі є задача про кон'юнктивну форму.

Дослідження складності алгоритмів дозволили по-новому поглянути на вирішення багатьох класичних математичних задач і знайти для ряду таких завдань (множення многочленів і матриць, рішення лінійних систем рівнянь і ін.) рішення, які вимагають менше ресурсів, ніж традиційні.

У теорії алгоритмів (англ. non-deterministic polynomial) називають безліч алгоритмів, час роботи яких істотно залежить від розміру вхідних даних; в той же час, якщо надати алгоритму деякі додаткові відомості (так званих свідків рішення), то він зможе досить швидко (за час, що не перевершує многочлена від розміру даних) вирішити задачу.

Інтуїтивно, клас NP містить завдання, для яких порівняно легко (за поліноміальний час на детермінованою машині Т'юрінга) можна довести, що потенційне рішення задачі дійсно є таким. Або, що еквівалентно, завдання NP-класу можуть бути вирішені за поліноміальний час на недетермінованої машині Т'юрінга [15].

Клас складності NP визначається для безлічі мов, тобто множин слів над кінцевим алфавітом. Мова L називається належить класу NP, якщо існують двомісний предикат $R(x, y)$ з класу P (тобто обчислюваний за поліноміальний час) і многочлен n^c такі, що для будь-якого слова x умова " x належить L" рівносильно

умові "знайдеться у довжини менше n^c такий, що вірно $R(x, y)$ (де n - довжина слова x). Слово y називається свідком приналежності x мови L . Таким чином, якщо у нас є слово, яке належить мові, і ще одне слово-свідок обмеженої довжини (яке буває важко знайти), то ми швидко зможемо переконатися в тому, що x дійсно належить L .

Еквівалентну визначення можна отримати, використовуючи поняття недетермінованої машини Т'юрінга (тобто такої машини Т'юрінга, у програми якої можуть існувати різні рядки з однаковою лівою частиною). Якщо машина зустріла "розвилку", тобто неоднозначність в програмі, то далі можливі різні варіанти обчислення. Предикат $R(x)$, який представляє дана недетермінована машина Т'юрінга, вважається рівним одиниці, якщо існує хоч один варіант обчислення, який повертає 1, і нулю, якщо всі варіанти повертають 0. Якщо довжина обчислення, що дає 1, не перевищує деякого многочлена від довжини x , то предикат називається таким, що належить класу NP. Якщо у мові існує предикат з класу NP, який розпізнає його, то мова називається такою, що належить класу NP. Це визначення еквівалентно наведеним вище: в якості свідка можна взяти номери потрібних гілок при розвилках в обчисленні. Так як для x належить мові, то довжина всього шляху обчислення не перевищує многочлена від довжини x , то і довжина свідка також буде обмежена многочленом від довжини x .

Будь-яку задачу, яка належить NP, можна вирішити за експоненціальний час за допомогою перебору всіх можливих свідків довжини менше n^c .

Легко бачити, що безліч мов з NP не замкнене щодо доповнення. Клас мов, доповнення яких належить NP, називається класом co-NP.

Серед усіх завдань класу NP можна виділити "найскладніші" - NP-повні задачі. Якщо ми навчимося вирішувати будь-яку з них за поліноміальний час, то все завдання класу NP можна буде вирішити за поліноміальний час.

У теорії алгоритмів NP-повна задача - це таке завдання з класу NP, до якої можна звести будь-яку іншу задачу з класу NP. Таким чином, NP-повні задачі утворюють в деякому сенсі підмножину "найскладніших" завдань в класі NP; і

якщо для якоїсь з них буде знайдено "швидкий" алгоритм рішення, то і будь-яка інша задача з класу NP може бути вирішена так само "швидко".

Можна привести багато завдань, про які на сьогоднішній день невідомо, чи належать вони P, але відомо, що вони належать NP[16]. Серед них:

- задача здійсненності булевих формул: дізнатися з даної булевої формули, чи існує набір змінних, які входять до неї, звертає її в 1. Свідок - такий набір;
- задача про кліки: з даного графу дізнатися, чи є в ньому кліки (повні підграфи) даного розміру. Свідок - номери вершин, що утворюють кліку;
- проблема існування Гамільтона циклу в графі. Свідок - послідовність вершин, що утворюють гамільтонов цикл;
- задача про комівояжера - розширений і більш наближений до реальності варіант попередньої задачі;
- існування цілочисельного рішення системи лінійних нерівностей. Свідок - рішення.

Також до NP завдань відносяться завдання факторизації і дискретного логарифма. Хоча вони і не є NP-повними завданнями і знаходження поліноміального алгоритму обчислення дискретного логарифма, ще не буде означати рівності $P = NP$.

Таким чином, несиметричні алгоритми шифрування широко використовуються у сучасних криптосистемах. Серед таких алгоритмів найвідомішим є алгоритм RSA, стійкість якого базується на складності задачі факторизації.

На сучасному етапі розвитку криптоаналізу не було виявлено ніяких руйнівних атак RSA. Основною атакою є атака розкладення на множники. Безпека алгоритму RSA заснована на складності задачі факторизації (розкладання на множники великих чисел). Відкритий та закритий ключі є функціями двох великих (200-300 розрядів та більше) простих чисел. Передбачається, що відновлення відкритого тексту за шифротекстом та відкритим ключем еквівалентно розкладанню на множники двох великих чисел. Існує багато алгоритмів розкладання на множники, але жоден з них не може знайти співмножники великого

цілого числа з поліноміальною складністю часу. Для того щоб забезпечити безпеку, RSA вимагає, щоб n було більше ніж 300 десяткових цифр. Це означає, що модуль повинен бути принаймні 1024 біта. Навіть при використанні потужного і найшвидшого комп'ютера, доступного на сьогодні, розкладання на множники цілого числа такого розміру вимагає нездійсненно великого часу. Це означає, що RSA безпечний, поки не буде знайдений ефективний алгоритм розкладання на множники.

З алгоритмами задачі дискретного логарифмування та NP-задачами на сьогодні така ж ситуація, через те, що алгоритми рішення цих задач мають експоненційну складність, то для великих чисел це може зайняти декілька років, навіть на сучасних комп'ютерах. Тому проводити атаки на оборону, в основі якої лежать ці алгоритми, є невигідною роботою, бо затрачені ресурси будуть значно перевищувати вигоду від результату.

Отже очевидно, що на даний момент існує проблема рішення таких задач, і виходом не може бути звичайне нарощування потужності за допомогою розподілених обчислень або використанні найсучасніших класичних комп'ютерів. Далі будуть розглянуті більш детально особливості квантових алгоритмів факторизації та їх реалізація

2 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ КВАНТОВИХ ОБЧИСЛЕНЬ

Відповідно до гіпотези М. Мінського продуктивність паралельної системи зростає (приблизно) пропорційно логарифму числа процесорів, тобто набагато повільніше, ніж лінійна функція, що вказує на необхідність розвитку не тільки класичних технологій, але і принципово нових підходів. Одним з напрямків зростання продуктивності є підвищення рівня паралелізму і відхід від фон-неймановської архітектури з послідовним виконанням команд і перехід до нових архітектур, які мають за основи нові обчислювальні парадигми.

Квантові комп'ютери і нейрообчислювачі є обчислювальними системами з природним паралелізмом. Квантовий паралелізм полягає в тому, що дані в процесі обчислень є квантовою інформацією, яка після закінчення процесу перетворюється в класичну шляхом вимірювання кінцевого стану квантового регістра. Підвищення продуктивності в квантових алгоритмах досягається за рахунок того, що при застосуванні однієї квантової операції велике число коефіцієнтів суперпозиції квантових станів перетвориться одночасно.

Електрон або будь-який інший квантовий об'єкт частково буде знаходитися в одній точці, частково в іншій, частково в третій і т. д. Такий стан електрона, коли він знаходиться відразу в декількох точках простору, називають суперпозицією квантових станів і описують зазвичай хвильової функцією, введеної в 1926 році німецьким фізиком Е. Шредінгером. Модуль значення хвильової функції у будь-якій точці, зведений в квадрат, визначає ймовірність знайти частинку в цій точці в даний момент. Після вимірювання положення частинки її хвильова функція як би стягується (колапсує) в ту точку, де частка була виявлена, а потім знову починає розпливатися. Властивість квантових частинок бути одночасно в багатьох станах, називається квантовим паралелізмом і успішно використовується в квантових обчисленнях[17].

2.1 Особливості квантових алгоритмів

Основний осередок квантового комп'ютера (квантовий біт, або скорочено кубіт (q-bit)) - це квантова частинка, що має два базових стани, які позначаються, як прийнято в квантовій механіці, 0 і 1. Двом значенням кубіта можуть відповідати, наприклад, основний і збуджений стани атома, напрям вгору і вниз спина атомного ядра, напрямок струму в надпровідному кільці, два можливих положення електрона в напівпровіднику і т. п.

Квантовий регістр влаштований майже так само, як і класичний. Він складається з ланцюжка квантових бітів, над якими можна проводити одно- і двохбітові логічні операції, подібно операціям НЕ, 2І-НЕ тощо.

До базових станів квантового регістра, утвореного L кубітами, відносяться всі можливі послідовності нулів і одиниць довжиною L . Усього може бути 2^L різних комбінацій, які можна вважати записом чисел в двійковій формі від 0 до $2^L - 1$. Однак ці базові стани не вичерпують всіх можливих значень квантового регістра (на відміну від класичного), оскільки існують ще й стани суперпозиції, що задаються комплексними амплітудами, пов'язаними умовою нормування. Класичного аналога у більшості можливих значень квантового регістра за винятком базових просто не існує, а його стани лише мала частина всього багатства станів квантового комп'ютера.

Ідея квантових обчислень, вперше висловлена Ю.І. Манінім і Р. Фейнманом, полягає в тому, що квантова система з L дворівневих кубітів (квантових елементів) має 2^L лінійно незалежних станів, а значить внаслідок принципу квантової суперпозиції простором станів такого квантового регістра є 2^L -мірний гільбертовий простір. Операція в квантових обчисленнях відповідає повороту вектора стану регістра в цьому просторі. Таким чином, квантовий обчислювальний пристрій розміром L кубіт може виконувати паралельно 2^L операцій. У загальному випадку системи з L кубітів мають 2^L класичних станів (00000 (L -нулів), 00001 (L -

цифр), ..., 11111 (L-одиниць)), кожне з яких може бути виміряна з вірогідністю 0-100%.

$$|\psi\rangle = \sum_{n=0}^{2^L-1} c_n |n\rangle \quad (2.1)$$

де $|n\rangle$ - базисні квантові стани (наприклад, стан $|001101\rangle$,
 $|c_n|^2$ - ймовірність знаходження в базисному стані $|n\rangle$.

Таким чином, одна операція над групою кубітів зачіпає всі значення, які вона може приймати на відміну від класичного біта, що і забезпечує безпрецедентний паралелізм обчислень.

У квантовому процесорі введені дані піддаються послідовності квантових логічних операцій, які з математичної точки зору описуються унітарним перетворенням, яке впливає на стан всього регістру (рис. 2.1). В результаті через кілька тактів роботи квантового процесора вихідний квантовий стан системи стає новою суперпозицією[18].

Для квантового випадку можна побудувати необмежену кількість перетворень, які не мають класичних аналогів.

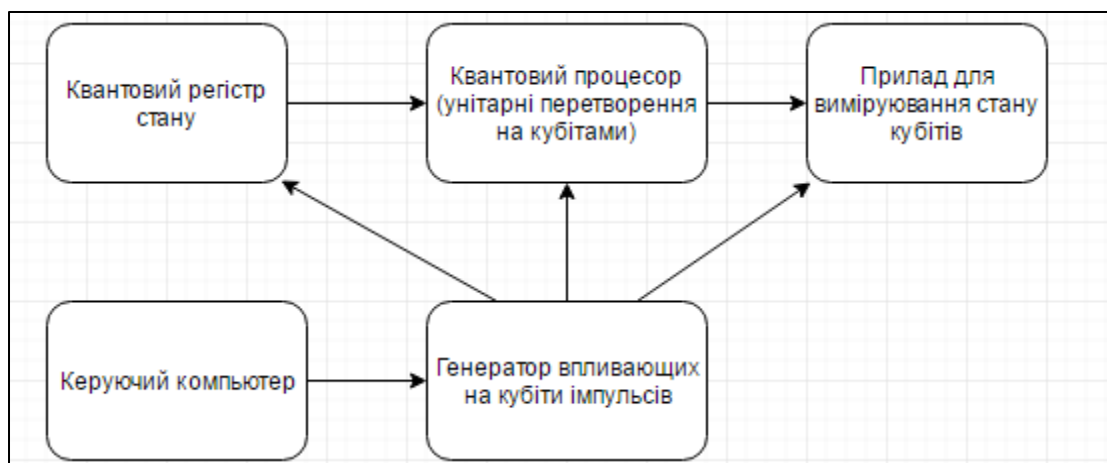


Рисунок 2.1 – Структура квантового комп'ютера

Конкретну реалізацію обчислювального перетворення прийнято називати вентилям (гейтом). У класичному випадку єдиним однобітним гейтом є тільки

базисне перетворення not . Серед квантових однокубітних перетворень, які не мають класичних аналогів, виділяються наступні для яких існують прості фізичні інтерпретації H-гейт Адамара, S-фазовий гейт, $\pi/8$ -гейт (2.2).

$$\begin{aligned} H &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \\ S &= \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \\ R &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \end{aligned} \quad (2.2)$$

де H-гейт Адамара,
S-фазовий гейт,
R - $\pi/8$ -гейт.

Якщо два кубіта знаходяться в певних станах $|Q_1\rangle$ і $|Q_2\rangle$, то стан 2-розрядного регістра буде визначатися їх тензорним добутком (2.3).

$$|Q_{12}\rangle = |Q_1\rangle \otimes |Q_2\rangle \quad (2.3)$$

де $|Q_1\rangle$ та $|Q_2\rangle$ - певні стани регістра,
 $|Q_{12}\rangle$ - загальний стан регістра.

Стани двох кубітів, що не описуються таким тензорним добутком, називаються сплєтеними (entangled). Саме такі стани відповідальні за специфічність квантових обчислювальних систем.

У загальному випадку стан двох кубітів визначається формулою (2.4).

$$|Q_{12}\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \quad (2.4)$$

де a, b, c та d – деякі коефіцієнти при можливих станах,
 $|Q_{12}\rangle$ - загальний стан регістра.

З визначення умови нормування і ототожнення (2.5).

$$|ij\rangle \equiv |i\rangle \otimes |j\rangle \quad (2.5)$$

де $|i\rangle$ та $|j\rangle$ - певні стани регістра,
 $|ij\rangle$ - загальний стан регістра.

Маємо, що умовою невиконання є нерівність (2.6).

$$AD - BC \neq 0 \quad (2.6)$$

де A,B,C та D – деякі квантові стани системи.

Це є умовою того, що стан $|Q_{12}\rangle$ сплутений. Кожний несплутений стан може бути перетворений у сплутений з допомогою квантового гейта, який є універсальним так, що сума ймовірностей всіх станів зберігається[19].

Квантові алгоритми мають декілька цікавих особливостей.

На відміну від випадку звичайної формальної логіки, операції над кубітами носять квантовий, імовірнісний характер, що обумовлює деякі особливості таких операцій. У загальному випадку, виділяють три класи квантових алгоритмів:

- алгоритми, засновані на квантових версіях перетворення Фур'є;
- алгоритми квантового пошуку;
- алгоритми моделювання квантових систем.

У всіх випадках квантовий алгоритм вирішує задачу ефективніше класичного, завдяки фундаментальній властивості квантових обчислень квантового паралелізму, що дозволяє обчислювати функцію $f(x)$ для деяких значень x одночасно[20].

2.2. Проблеми квантових обчислень

В основі квантових обчислень лежить квантова теорія у якій існують певні правила, які крім позитивних моментів (квантовий паралелізм) накладають також і

обмеження, які можуть викликати у програмістів і розробників алгоритмів труднощі. Розглянемо основні з них.

2.2.1 Проблема когерентизації

В теорії ідеального квантового комп'ютера вважається, що квантові суперпозиції, що описують стан регістра з L кубітів, залишаються когерентними в процесі обчислень скільки завгодно довго. Однак взаємодія регістра з неконтрольованим оточенням, неточності в значеннях параметрів керуючих імпульсів, а також неконтрольована взаємодія кубітів між собою є джерелами декогеренції квантового стану. Декогеренція означає, що когерентний стан системи перетворюється в змішане, що описується матрицею щільності.

В описі системи матрицею щільності немає інформації про фази базисних станів, що позбавляє систему властивостей інтерферувати і заплутуватися. По суті декогеренція стану квантової системи означає її класифікацію, тобто перехід в стан, що описується законами класичної фізики.

Важливим параметром квантової системи є час декогеренції її стану. Процеси декогеренції квантових станів комп'ютера забороняють існування повномасштабного (здатного вирішувати великі завдання квантового комп'ютера) через те, що час когерентизації має обернено пропорційну залежність до кількості операцій. Виходом є або скоротити час операцій, або збільшити час декогеренції кубіта. Тобто необхідні способи стабілізації когерентного стану комп'ютера на скільки завгодно тривалий час, щоб можна було завершити обчислення будь-якого завдання з більшим (але все ще поліноміальним) алгоритмом обчислень. Таким методом є метод квантової корекції помилок.

Метод передбачає періодичну очистку стану квантового комп'ютера від малих помилок, що виникли в векторі стану в результаті процесів декогерентизації за час після останнього очищення. Запропоновані також активні методи

придушення когерентізації. Однак квантовий метод корекції помилок є головною надією дослідників на можливість побудови повномасштабного квантового комп'ютера, що працює в когерентному стані скільки завгодно тривалий час[21].

2.2.2 Проблема клонування квантового стану регістра

У класичному комп'ютері однією з основних операцій є копіювання (передача) інформації з однієї точки пам'яті в іншу або в регістр. У квантовій теорії існує "Теорема про заборону клонування" (теорема була сформульована Вуттерсом, Зуреком і Дієксом [22] у 1982 році і має величезне значення в області квантових обчислень, квантової теорії інформації та суміжних областях), яка стверджує, що неможливо створити ідеальну копію довільного невідомого квантового стану.

Обійти цю проблему можна за допомогою квантової телепортації. Передача квантового стану на відстань за допомогою роз'єднаної в просторі зчепленої (заплутаної) пари і класичного каналу зв'язку, при якій стан руйнується в точці відправлення при проведенні вимірювання, після чого відтворюється в точці прийому. Термін встановився завдяки опублікованій в 1993 році статті в журналі "Physical Review Letters", де описано, яке саме квантове явище пропонується називати "телепортацією" (англ. Teleporting) і чим воно відрізняється від популярної в науковій фантастиці "телепортації". Квантова телепортація не передає енергію або речовину на відстань [23].

При здійсненні квантової телепортації крім передачі інформації по квантовому каналу, необхідно також здійснити передачу додаткової інформації, необхідної для прочитання повідомлення, за класичним каналом. Для передачі "квантової частини" використовуються характерні для квантово-заплутаних часток кореляції Ейнштейна - Подільського - Розена, а для передачі класичної інформації годиться будь-який звичайний канал зв'язку.

Для простоти розглянемо квантову систему з двома можливими станами ψ_1 і ψ_2 (наприклад, проекцію спіна електрона або фотона на задану вісь). Такі системи часто називають кубітами. Однак описаний нижче спосіб придатний для передачі стану будь-якої системи, що має кінцеве число станів.

Розглянемо опис квантової телепортації. Нехай у відправника є частка А, що знаходиться в довільному квантовому стані $\psi_A = \alpha\psi_1 + \beta\psi_2$, і він хоче передати цей квантовий стан одержувачу, тобто зробити так, щоб у одержувача виявилася в розпорядженні частка В в тому ж самому стані. Іншими словами, необхідно передати значення двох комплексних чисел α і β (з максимальною точністю). Зауважимо, що головна мета тут - це передати інформацію не як можна швидше, а якомога точніше. Для досягнення цієї мети виконуються наступні кроки:

- відправник і одержувач домовляються заздалегідь про створення пари квантово-заплутаних часток С і В, причому С потрапить відправнику, а В - одержувачу. Оскільки ці частки заплутані, то кожна з них не володіє своєю хвильовою функцією (вектором стану), але вся пара цілком (а точніше, що цікавлять нас ступеня свободи) описуються єдиним чотиривимірним вектором стану ψ_{BC} ;

- квантова система частинок А і С має чотири стани, однак ми не можемо описати її стан вектором - чистим (повністю визначеним) станом володіє лише система з трьох частинок А, В, С. Коли відправник робить вимір, який має чотири можливих результати, над системою з двох частинок А і С, він отримує одне з 4 власних значень вимірюваної величини. Оскільки при цьому вимірі система з трьох частинок А, В, С колапсує в якийсь новий стан, причому стани частинок А і С стають відомі повністю, то зчепленість руйнується і частка В виявляється в деякому певному квантовому стані;

- саме в цей момент відбувається як би "передача" "квантової частини" інформації. Однак відновити передану інформацію поки неможливо: одержувач знає, що стан частинки В якось пов'язаний зі станом частинки А, але не знає як саме;

— для з'ясування цього необхідно, щоб відправник повідомив одержувачу за звичайним класичним каналом результат свого вимірювання (витративши при цьому два біта, відповідні зачепленому стану АС, вимірюваному відправником). За законами квантової механіки виходить, що, маючи результат вимірювання, проведеного над парою частинок А і С і плюс до того заплутану з С частку В, одержувач зможе зробити необхідне перетворення над станом частинки В і відновити початковий стан А.

Повна передача інформації здійсниться тільки після того, як одержувач буде володіти даними, отриманими по обох каналах. До того як отримано результат за класичним каналом, одержувач нічого не може сказати про переданий стан.

Фантастичне поняття телепортації відбувається із специфічної інтерпретації експерименту: "початковий стан частинки А після всього, що сталося руйнується. Тобто стан був не скопійований, а перенесений з одного місця в інше"[24].

2.2.3 Проблема вимірювання поточного стану системи

Проблема вимірювання поточного стану системи заснована на парадоксі Ейнштейна - Подільського - Розена. Суть парадоксу полягає в тому, що згідно із співвідношенням невизначеностей Гейзенберга, немає можливості одночасно точно виміряти координату частинки і її імпульс. Припускаючи, що причиною невизначеності є те, що вимір однієї величини вносить принципово непереборні обурення в стан і виробляє спотворення значення іншої величини, можна запропонувати гіпотетичний спосіб, яким співвідношення невизначеностей можна обійти. Припустимо, дві однакові частки А і В утворилися в результаті розпаду третьої частки С. В цьому випадку, за законом збереження імпульсу, їх сумарний імпульс $p_A + p_B$ має дорівнювати вихідному імпульсу третьої частки p_C , тобто, імпульси двох частинок повинні бути пов'язані. Це дає можливість виміряти імпульс однієї частки (А) і за законом збереження імпульсу $p_B = p_C - p_A$ розрахувати

імпульс другий (B), не вносячи в її рух ніяких обурень. Тепер, вимірявши координату другої частки, можна отримати для цієї частки значення двох невимірних одночасно величин, що за законами квантової механіки неможливо. Виходячи з цього, можна було б зробити висновок, що співвідношення невизначеностей не є абсолютним, а закони квантової механіки є неповними і повинні бути в майбутньому уточнені.

Якщо ж закони квантової механіки в даному випадку не порушуються, то вимір імпульсу однієї частки рівносильний виміру імпульсу другої частки. Однак це створює враження миттєвого впливу першої частки на другу в протиріччі з принципом причинності.

Грунтуючись на цьому, отримуємо проблему того, що під час виконання програми ми не можемо подивитися, що знаходиться всередині квантових реєстрів в даний момент інакше ми зруйнуємо стан суперпозиції квантової системи і не зможемо далі виконувати наш квантовий алгоритм. Це означає, що налагодження виконання квантових програм стає неможливою [25].

2.2.4 Ймовірнісний характер квантових обчислень

Так як квантовий стан описується як суперпозиція, тобто система має кілька дискретних станів одночасно з певними можливостями, то і кінцевий результат роботи квантового алгоритму ми одержуємо з деякою ймовірністю, що призводить до того, що ми будемо отримувати правильний результат не у всіх випадках. І тому або потрібно створювати квантові алгоритми, в яких правильний результат виходить з високою ймовірністю, або запускати нашу програму багато разів і постійно перевіряти результати. Останній спосіб може підійти для вирішення NP завдань, в яких знаходження результату в сучасних реалізаціях виконується за експоненціальний час, а перевірка результату за поліноміальний.

2.3 Фізична реалізація квантових комп'ютерів

Реалізація квантового комп'ютера має такі вимоги:

- масштабованість (можливість збільшення) числа кубітів. Виділення і фіксація в просторі досить великого числа ($L \sim 10^2 \div 10^3$) 2-х рівневий частинок - кубітів, на які можна було б вибірково впливати для організації їх квантової еволюції відповідно до виконуваним алгоритмом;
- кубіти можуть бути ініційовані у будь-який початковий стан. Можливість приготування L кубітів вхідного регістра в вихідному базисному стані $|O_1, O_2, O_3, \dots, O_L\rangle$ (ініціалізація);
- квантові вентиля повинні спрацьовувати швидше часу декогеренції. Перешкодостійкість обчислювальних процесів і придушення ефектів декогерентності квантових станів, обумовлених взаємодією кубітів з навколишнім середовищем. Час декогерентності має $\geq 10^4$ разів перевершувати час виконання основних квантових операцій (час такту). Помилка при виконанні окремої операції повинна бути $\leq 10^{-4}$;
- реалізація повного набору вентилів Тюрінга. Так як будь-яка унітарна квантова операція може зводиться до сукупності однокубітних і двохкубітних операцій, то при виборі фізичної системи необхідна наявність певних нелінійних взаємодій між керованими кубітами, що забезпечують виконання двохкубітних операцій. Керовані операціями імпульси повинні контролюватися з точністю не гірше, ніж 10^{-4} ;
- зчитування інформації з кубітів. Виконання вимірювання стану квантової системи на виході з високою точністю.

На сьогоднішній день відомо кілька можливих фізичних систем, які розглядаються як кандидати для квантових комп'ютерів:

- іони або нейтральні атоми з двома низьколежачими коливальними або надтонкими рівнями, утримувані в силових пастках, створених в вакуумі за

допомогою електричних і магнітних полів, при лазерному охолодженні до мікрокельвінових температур;

- сверхпроводникові структури з переходами Джозефсона;
- окремі електронні та ядерні спini в магнітному полі;
- квантові точки з двома електронними орбітальними та спиновими станами;
- певні стани квантового електромагнітного поля в електродинамічних резонаторах і фотонних кристалах.

2.4 Квантові комп'ютери сьогодні

Квантові комп'ютери, здатні вирішувати реальні завдання, можуть з'явитися вже скоро, і вони змінять світ. Їх поява може зробити революцію у фізиці та медицині, і абсолютно точно - в інформаційній безпеці.

Про квантових комп'ютерах, здатних в числі іншого швидко підбирати ключі до шифрів, говорять уже кілька десятиліть, але до недавнього часу вони залишалися красивою концепцією в голові вчених, а не реально діючими зразками техніки.

Тепер в ситуації назрів перелом, про що найкраще свідчить випущена в серпні 2016 року рекомендація американського Агентства національної безпеки, яке повинно захищати держтаємниці і рекомендувати американському державному і приватному сектору найбільш ефективні алгоритми комп'ютерного захисту, в тому числі шифрування. Саме свій набір рекомендованих криптоалгоритмів, відомий як SUITE B, і змінив в статусі агентство.

Посилаючись на несподівано швидкий прогрес фізики і техніки, АНБ рекомендує готуватися до швидкої появи практичних квантових комп'ютерів, які зроблять найпопулярніші сьогодні системи шифрування, цифрового підпису та обміну ключами набагато більш уразливими.

Розглянемо програмні продукти Google, пов'язані з квантовими обчисленнями.

7 липня 2016 року Google повідомили про початок експерименту з постквантовою криптографією в браузері Chrome. Конкретно - в версії для розробників браузера Chrome Canary у деяких користувачів буде включена система шифрування підключення до серверів Google за допомогою нового алгоритму. Система ж побудована на основі наукової роботи по впровадженню постквантового алгоритму в протокол TLS. Алгоритм названий символічно - "Нова надія"; Розробники пишуть, що експеримент переслідує дві мети. По-перше, дати дослідникам зламати практичну реалізацію постквантового алгоритму. Друга, оцінити швидкість і надійність з'єднання в такій конфігурації. У пості команди Google прямо говориться, що "Надію" не планується нав'язувати як новий криптостандарт.

Quantum Computing Playground. Майданчик є веб-додатком Chrome (Chrome Experiment), який використовує WebGL, щоб імітувати до 22 кубітів на GPU. Присутнє невелике середовище розробки, щоб писати, компілювати і виконувати код. Також є вже готові приклади алгоритмів (Гровер, Шор), зручний відлагоджувач і інструмент для 3D візуалізації квантових станів, так що можна своїми очима побачити, що відбувається всередині маленького квантового комп'ютера. Програми написані на мові QScript, яка дуже схожа на будь-яку іншу скриптову мову. Найбільше число, які ви можете факторізувати будь-які числа до 64 (6 біт), так що взламувати щось реальне не вийде[1].

Розглянемо успіхи в створенні квантових комп'ютерів.

D-Wave Systems. Дана компанія 23 січня 2017 анонсувала початок продажів свого 2000-кубітного квантового комп'ютера D-Wave 2000Q і продала першу модель за 15 мільйонів доларів. D-Wave представили свій квантовий комп'ютер публіці ще в вересні 2016 року, заявивши, що нове рішення буде містити 2 тисячі кубітів. Це в два рази більше, ніж у квантового комп'ютера попереднього покоління - D-Wave X2, запущеного в серпні 2016 року. D-wave 2000Q є так званий адіабатичний комп'ютер, що працює за принципом квантового відпалу. Це

квантова система з великою кількістю компонентів і контрольованих параметрів. Охолоджуючи її до дуже низької температури (комп'ютер попередньої моделі функціонував при температурі в 15 мілікельвінов - близько -273°C), розробники припускають, що система досягає мінімальної енергії, і потім, повільно змінюючи задані параметри, використовують закони квантової механіки для переведення системи з початкового стану в новий стан мінімальної енергії за рахунок квантового тунелювання. При такій великій кількості кубітів, комп'ютер компанії D-Wave має дуже великий недолік в тому, що його можна застосовувати тільки для одного типу завдань, завдань оптимізації. Так що повноцінним квантовим комп'ютером дану машину ще назвати не можна.

21 Квітня 2017 року Джон Мартінез, глава дослідницької групи Google, заявив, що до кінця року його команда розробить чіп, який досягне "квантової переваги", що означає що він буде перевершувати сучасні комп'ютери в виконанні певних обчислень. Його впевненість полягає в тому, що його команда з 25 осіб, створила новий квантовий чіп з шести кубітів. До цього його команда створювала дев'яти-кубітний чіп. Але особливістю шестибітного є те, що в дев'яти-кубітному кубіти розташовані в ряд, в той час, як в шести-кубітному згруповані в сітку 2 на 3, таким чином вони будуть розташовані і у великих чіпах. А вже у березні 2018 року компанія Google оголосила про створення чіпа на 72 кубіти під назвою Bristlecone.

IBM Q є першою, яка виступила з ініціативою створення комерційно доступних універсальних квантових обчислювальних систем для бізнесу та науки. В рамках виставки CES 2019 підрозділ IBM Research провело анонс першої в світі квантової системи, придатної для комерційного застосування. Анонсований квантовий комп'ютер IBM Q System One включає в себе систему з 20 кубітів. Залізо здатне до самокалібрування і оптимізовано для роботи в криогенних умовах, ці заходи необхідні для зменшення числа помилок. Крім того, комп'ютер має власну високопродуктивну криогенну систему, а за заявою IBM допустимо проводити діагностику, обслуговування і навіть модернізацію системи без її виключення, зі збереженням можливості роботи користувачів. Важливою характеристикою, на якій виробник акцентує увагу, є вбудований функціонал для підключення

квантового комп'ютера до хмарної системи. Таким чином відбувається функціональний перехід від спеціалізованих квантових систем на чіпі, використовуваних в експериментальних цілях, до повноцінної інтеграції квантових обчислень для призначених для користувача систем і потреб бізнесу. Також IBM вже надало API клієнт, написане на мові програмування Python, для взаємодії з хмарної системою IBM Q.

Генеральний директор Intel Брайан Крзаніч оголосив на виставці Consumer Electronics Show (CES), що Intel виготовила 49-кубітний надпровідний чіп під кодовою назвою Tangle Lake і поставила його партнеру QuTech в Нідерландах. Раніше вони розробили чіп на 17 кубітів, який вони оголосили в жовтні 2018 року. Компанія Intel не розкрила жодних характеристик продуктивності або якості для цього чіпа. Цікавий аспект партнерства Intel та Qutech полягає в тому, що вони також працюють над технологією спін-кубіту. Спін-кубіти також мають потенціал виготовлення в напівпровідниковій фабриці, але потенційно можуть бути більш масштабованими, ніж надпровідні мікросхеми, оскільки вони мають менший розмір. На сьогодні Intel ще не оголосила, що виготовила будь-які квантові мікросхеми, використовуючи технологію spin qubit.

Способи захисту своїх особистих даних від впевнено наступаючих квантових комп'ютерів:

– уникати обміну ключами за допомогою асиметричних криптоалгоритмів. "Квантові алгоритми ефективно вирішують NP-повні задачі, які використовуються в сучасній асиметричній криптографії (завдання розкладання числа на множники та обчислення дискретного логарифма), що компрометує еліптичні криві, шифрування і цифрові підписи RSA, Ель-Гамаль і інші, а також схему обміну ключами Діффі - Хеллмана, - прокоментував Віктор Алюшин, експерт "Лабораторії Касперського". Необхідно або використовувати інші протоколи узгодження ключів по мережі, або обмінюватися ключами фізично. Наприклад, в мобільному месенджері Threema передбачена процедура взаємного сканування QR-кодів на телефонах співрозмовників, що є непоганою запорукою секретності подальшого листування;

– не скупитися на біти в ключах. Поки квантовий комп'ютер залишається в десятиліттях від нас, хакери не шкодують сил на розробку інших витончених кріптоатак, тому використання ключів RSA-8192 або P-256 цілком виправдовує себе для конфіденційних документів;

– використовувати посилені симетричні алгоритми. "Квантові алгоритми дозволяють ефективно підбирати паролі і знаходити симетричні ключі шифрування: квантовий комп'ютер знаходить ключ шифрування довжини $2N$ за той же час, за який звичайний комп'ютер знаходить ключ шифрування довжини N байт. Тому довжини ключів існуючих симетричних шифрів слід збільшити в два рази для збереження колишньої стійкості", - радить Віктор Алюшин. Варто врахувати, що кріптоатаки на AES численні і витончені, і, хоча здебільшого вони безуспішні, перехід на 256-бітові ключі не зашкодить навіть без оглядки на квантові обчислення;

– використовувати експериментальні "постквантові" рішення. Вони варіюються в практичності, зручності, а головна - доведена криптостійкість ряду з них залишається під питанням. Проте бажаючих поекспериментувати відправимо до короткого огляду наявних утиліт на базі однієї з найперспективніших "постквантових" криптосистем на основі завдань теорії решіток.

На основі усього вищенаведеного очевидно, перспективність квантових обчислень полягає в досить швидкому вирішенні цілого класу задач. Однак в даний час вони досить складні і важко оброблювані.

Квантові обчислення не дають ефективного вирішення для всіх завдань. Також не дають вони універсального способу подолання закону Мура про фундаментальне межі мініатюризації. Квантові обчислення дозволяють вирішувати лише деякі певні проблеми, але дуже ефективно; завдання, які на класичному комп'ютері зайняли б час більше ніж вік всесвіту, на квантовому комп'ютері можуть бути вирішені за пару днів. Але для інших завдань було доведено, що квантові обчислення не перевершують класичні, або перевершують, але не значно. Квантові обчислення сильно вплинуть на безпеку.

Як показують дослідження, природний паралелізм квантових комп'ютерів і нейрообчислювачів дозволяє подолати обмеження по швидкодії за рахунок паралельної обробки інформації з величезною кількістю каналів $\sim 10^5 - 10^6$ і здійснювати до $10^{13} - 10^{15}$ операцій в секунду, що значно перевершує число операцій в секунду в сучасних існуючих електронних системах. У свою чергу застосування квантових комп'ютерів дозволить вирішувати складні завдання в таких областях, як криптографія.

Реалізація квантових рішень є дуже складною задачею через велику кількість проблем, таких як вплив навколишнього середовища, яке може зруйнувати стан суперпозиції під час підрахунків, неможливість копіювання стану квантових регістрів, неможливість відлагодження під час виконання квантової програми, складність створення та управління квантовими процесами всередині комп'ютера тощо. Але незважаючи на всі ці складності на сьогоднішній день вже існує досить велика кількість готових рішень, розроблених різними компаніями. Лідерами у цій галузі є компанії D-Wave Systems, Google, IBM, Microsoft. Також існують рішення, в яких можливо змоделювати квантові процеси та квантові обчислення, наприклад Quantum Computing Playground від Google. Також треба зауважити, що вже йдуть дослідження, розробка та тестування постквантових алгоритмів та протоколів для криптографії.

3 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ФАКТОРИЗАЦІЇ

3.1 Алгоритм факторизації Шора

Алгоритм Шора [26] – квантовий алгоритм факторизації (розкладання числа на прості множники), квантова реалізація якого, дозволяє розкласти число M за час $O(\lg^3 M)$, використовуючи $O(\lg M)$ логічних кубітів. Алгоритм був розроблений Пітером Шором у 1994 році. Сім років потому, в 2001 році, його працездатність була продемонстрована групою спеціалістів з IBM, які розклали число 15 на множники 3 та 5 за допомогою квантового комп'ютера з 7 кубітами.

Суть алгоритму Шора полягає в зведенні задачі факторизації до задачі пошуку періоду функції. Якщо відомий період функції, то факторизація здійснюється за допомогою алгоритму Евкліда за поліноміальний час на класичному комп'ютері. Квантова частина алгоритму факторизації якраз пошуком періоду функції. А класична частина алгоритму спочатку спеціальним чином готує дану функцію, а потім перевіряє знайдений квантової частиною період на достатність для вирішення завдання. Якщо період знайдений правильно (алгоритм імовірнісний, так що може знайти не те, що хочеться), то задача вирішена. Якщо ж ні, то квантова частина алгоритму запускається ще раз. А, оскільки, перевірка правильності рішення для задачі факторизації дуже проста (помножити два числа та порівняти з третім), то цю частину алгоритму можна не брати до уваги з точки зору підрахунку складності [27].

Розглянемо класичний алгоритм Шора на прикладі розкладання числа 35. Спочатку необхідно знайти період періодичної функції наведеної у формулі 3.1.

$$f(x) = a^x \pmod{M} \quad (3.1)$$

де a – деякий параметр, який вибирається довільно до старту алгоритма,
 M – число, яке необхідно факторизувати.

Найголовніше обмеження полягає в тому, щоб у чисел a і M не було спільних дільників, більших за 1. Оскільки ми хочемо факторизувати число 35, як параметр

"a" візьмемо число 2. Зазвичай це значення і беруть (найчастіше воно підходить), але іноді необхідно брати інше значення, і критерії вибору будуть описані пізніше.

Тепер складемо таку таблицю (3.1), у якій буде наведено значення функції.

З таблиці 3.1 видно, що в даному конкретному прикладі періодом функції є значення r , яке дорівнює 12. Іншими словами, щоб знайти період за допомогою такої таблиці, необхідно знайти мінімальне значення x , більше 1, для якого буде виконуватися нерівність (3.2).

Таблиця 3.1 – Значення періодичної функції для різних x

x		0	1	2	3	4	5	6	7	8	9	10	11	12
a^x	2^x	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
$a^x \bmod M$	$2^x \bmod 35$	1	2	4	8	16	32	29	23	11	22	9	18	1

Алгоритм Шора виграє саме на етапі знаходження періоду, оскільки за допомогою моделі квантових обчислень знайти період функції можна з поліноміальною складністю, чого не можна зробити в рамках класичної обчислювальної моделі.

$$a^x \equiv 1 \pmod{M} \quad (3.2)$$

де a – довільне число,

M – число для факторизації,

x – змінна, яка подається на вхід до функції.

Далі знаходимо множники числа M , які визначаються як найбільші спільні дільники (НСД) за формулою (3.3).

$$factor_{1,2} = \gcd(a^{r/2} \pm 1, M) \quad (3.3)$$

де $factor_{1,2}$ – множники числа M ,
 $\gcd$ – формула для обчислення найбільшого спільного дільника,
 a – довільно вибране раніше число,
 r - період,
 M – число для факторизації.

З формули 3.3 слідує ще одне обмеження на параметр a , яке довільно обирається перед запуском алгоритму, знайдений період повинен бути парним, тобто націло ділитися на 2. Оскільки в нашому прикладі r дорівнює 12, то дана умова виконана.

Тепер знаходимо два числа за формулою $a^{r/2} \pm 1$ виходить два числа 63 та 65. Далі за допомогою алгоритму Евкліда знаходимо найбільші спільні дільники, це буде пара 7 та 5. Це і буде відповідь на нашу задачу розкладання числа 35 на прості множники.

Цей алгоритм можливо оптимізувати. Як бачимо з таблиці 3.1, значення рівняння 3.1 при вхідному даному від 1 до 5 буде просто число a у ступені x . Тобто поки $a^x < M$ рахувати залишок від ділення не має сенсу. Тому існує можливість пропустити перші декілька кроків.

Кількість кроків для пропуску можливо знайти вирахувавши логарифм від M з основою a (формула 3.4).

$$n_{skip} = \lceil \log_a M \rceil \quad (3.4)$$

де n_{skip} – кількість кроків для пропуску,
 a – довільно вибране число,
 M – число для факторизації.

На комп'ютері дуже легко знайти цілу частину двоїчного логарифма від цілого числа, це кількість бітів в ньому. Тому формулу 3.4 за допомогою формули логарифма про перехід до нової основи можливо записати у вигляді 3.5.

$$n_{skip} = \left\lceil \frac{\log_2 M}{\log_2 a} \right\rceil \quad (3.5)$$

де n_{skip} – кількість кроків для пропуску,
 a – довільно вибране число,
 M – число для факторизації.

Тепер складемо структурований та формалізований алгоритм, який включає до себе наступні кроки:

- а) вибрати випадкове число a , яке менше за M ($a < M$);
- б) за допомогою алгоритму Евкліда вирахувати НСП(a, M);
- в) якщо НСП(a, M) не дорівнює 1, то існує нетривіальний дільник числа M , алгоритм закінчився (вироджений випадок);
- г) вирахувати кількість зайвих шагів для періодичної функції за формулою 3.4 або 3.5;
- д) шукаємо період функції 3.1;
- е) якщо період є непарним то повернутися на шаг a та вибрати інше число a ;
- ж) якщо виконується нерівність (3.6) то повернутися на шаг a та вибрати інше число a ;

$$a^{r/2} \equiv -1 \pmod{M} \quad (3.6)$$

де a деякий параметр, який вибирається довільно до старту алгоритму,
 M – число, яке необхідно факторизувати,
 r – період функції 3.1.

з) визначити два значення за формулою 3.3, які і являються нетривіальними дільниками числа M .

З усіх цих кроків найбільше затратним по ресурсам є крок д). Цей шаг і буде оптимізований за допомогою квантових обчислень.

Важливим моментом в цьому алгоритмі є вибір кількості кубітів. Відповідно до рекомендацій Пітера Шора, автора алгоритму, необхідно вибрати таку кількість кубітів q , при якому виконується нерівність 3.7.

$$M^2 \leq 2q \leq 2M^2 \quad (3.7)$$

де M – кількість бітів числа для факторизацій,
 q – кількість бітів необхідних для квантового алгоритму.

Виконання цієї нерівності означає, що даної кількості кубітів буде досить для того, щоб виконати функцію, для якої шукається період, достатня кількість разів, щоб спрацювала конструктивна інтерференція.

Розглянемо спочатку квантову схему алгоритму (рис. 3.1).

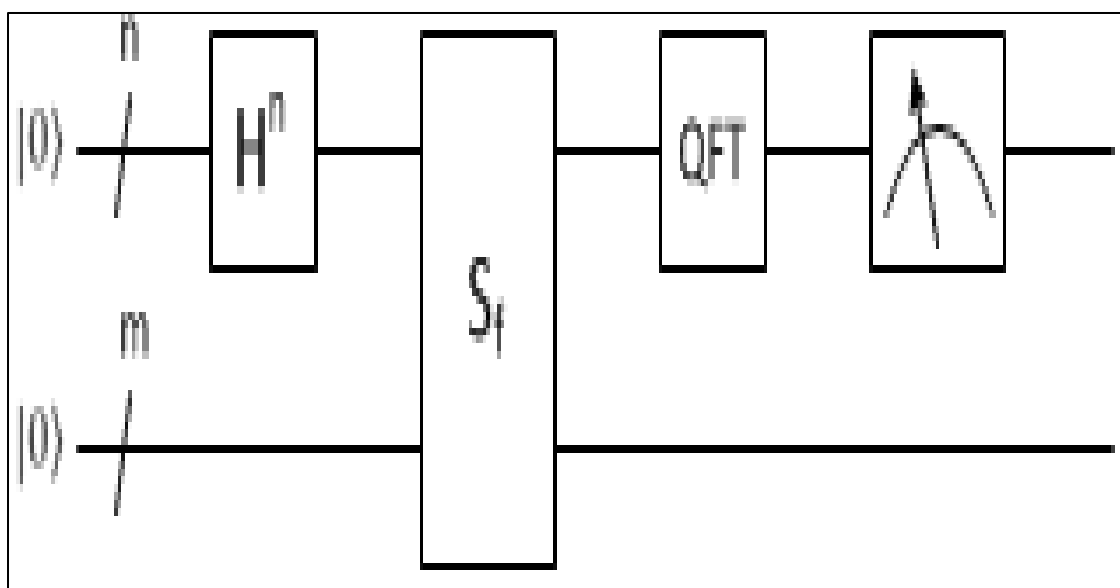


Рисунок 3.1 – Діаграма квантової схеми алгоритму факторизації Шора

Для реалізації квантової частини алгоритму обчислювальна схема представляє собою два квантових регістра m та n . Спочатку кожен з них складається з сукупності кубітів в нульовому квантовому стані $|0\rangle$. Регістр m використовується для розміщення аргументів x функції $f(x)$ (формула 3.1). Регістр n (додатковий) використовується для розміщення значень функції $f(x)$ з періодом r , який підлягає обчисленню.

Саме квантове обчислення складається з чотирьох кроків.

На першому кроці за допомогою операції Уолша-Адамара (H^n) первинний стан $|0\rangle$ регістра n переводиться в рівно імовірнісну суперпозицію усіх булевих станів N . Другий регістр залишається в стані $|0\rangle$. У підсумку залишається наступний стан для системи двох регістрів (формула 3.8).

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x, 0\rangle \quad (3.8)$$

де N – кількість усіх можливих булевих станів.

На другому кроці до двох регістрів застосовується унітарне перетворення (оракул S_f), яке переводить стан $|x, 0\rangle$ у $|x, f(x)\rangle$. У результаті отримуємо наступний стан системи (3.9).

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x, 0\rangle \xrightarrow{S_f} \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x, a^x \bmod M\rangle \quad (3.9)$$

де N – кількість усіх можливих булевих станів,

M – число для факторизації.

На третьому кроці виконується квантове перетворення Фур'є (QFT), яке уявляє собою унітарне перетворення стану квантового регістра, яке задається N -мірним вектором стану за формулою 3.10. У двувимірній x, k -площині перетворення Фур'є відповідає повороту осі координат на 90 градусів, яке призводить до перетворення шкали x у шкалу k . В результаті цього кроку ми отримуємо перетворений перший регістр (формула 3.11).

$$QFT_N : \sum_{x=0}^{N-1} f(x) |x\rangle \longrightarrow \frac{1}{N} \sum_{k=0}^{N-1} \left(\sum_{x=0}^{N-1} \exp(2\pi i kx/N) f(x) \right) |k\rangle, \quad (3.10)$$

$$\frac{1}{N} \sum_{x=0}^{N-1} \sum_{k=0}^{N-1} \exp(2\pi i kx/N) |k, a^x \bmod M\rangle \quad (3.11)$$

де N – кількість усіх можливих булевих станів.

На четвертому кроці виконується вимірювання регістра X відносно ортогональної проекції (3.12).

$$|0,0\rangle \oplus I, |1,1\rangle \oplus I, |1,1\rangle \oplus I, \dots, |N-1, N-1\rangle \oplus I \quad (3.12)$$

де N – кількість усіх можливих булевих станів,

I - тотожний оператор на гільбертовому просторі другого регістра m .

У результаті отримуємо наступну формулу (3.13).

$$|k, a^k \bmod M\rangle \quad (3.13)$$

де $k \in [0, N-1]$,

a – довільно вибране число,

M – число для факторизації.

Із ймовірністю 3.14.

$$\frac{1}{N} \sum_{x: a^x \equiv a \bmod M}^{N-1} \exp(2\pi i kx/N) \quad (3.14)$$

де N – кількість усіх можливих булевих станів,

$k \in [0, N-1]$.

Далі знаходимо найкраще приближення (знизу) до k/N . Та передаємо знайдений період до наступного кроку класичного алгоритма.

В деякій мірі визначення періоду функції за допомогою перетворення Фур'є аналогічно вимірюванню постійних решітки кристала методом рентгенівської або

нейтронної дифракції. Щоб визначити період r не потрібно обчислювати усі значення $f(x)$.

3.2 Алгоритм факторизації Ферма

Метод факторизації Ферма – алгоритм розкладання на множники непарного цілого числа n . Розроблений П'єром Ферма у 1643 році. Метод Ферма засновано на теоремі про подання числа у вигляді різниці двох квадратів. Зараз більшість сучасних методів криптоаналізу засновано на цій ідеї.

Метод заснований на пошуку таких цілих чисел x і y , які задовольняють співвідношенню $x^2 - y^2 = n$, що веде до розкладання $n = (x + y)(x - y)$.

Крок 1: необхідно обчислити цілу частину квадратного кореня від числа n .

Крок 2: для $x = 1, 2 \dots$ будемо розраховувати значення згідно з формули 3.15 доки чергове значення $q(x)$ не буде дорівнювати цілому квадрату.

$$q(x) = (m + x)^2 - n, \quad (3.15)$$

де x – факторизуєме число,

m – константа, яка додається при кожній ітерації циклу.

Крок 3: нехай $q(x)$ є повним квадратом, наприклад числа B : $q(x) = B^2$. Визначимо $A = (m + x)$, звідки $A^2 - n = B^2$ знайдемо $n = A^2 - B^2 = (A + B)(A - B)$.

Послідовний алгоритм реалізується згідно з описаними кроками. Для оптимізації методу факторизації Ферма використовуються паралельні обчислення на кроці 2 та 3. Сутність паралелізації полягає у тому, що усі обчислення, які виконуються для конкретного значення x не залежать від обчислень для інших значень x і можуть бути виконані у паралельних потоках.

3.3 Реалізація методу факторизації ρ -Полларда

У всіх ρ -методах Полларда будується числова послідовність, елементи якої утворюють цикл, починаючи з деякого номера n , що може бути проілюстровано, розташуванням чисел у вигляді грецької букви ρ . Також варто відзначити, що цей алгоритм відносить до ймовірносних. Тобто даний метод не гарантує, що число буде повністю розкладено, але дуже часто швидко повертає результат. Основною ідеєю ρ -розкладання є побудова ряду, який би схилювався до відповіді. Так як даний ряд в якийсь момент зациклиться, то його можна представити у вигляді латинської літери ρ , що зображена на рисунку 3.2, де цикл якраз і є верхня частина, а перші члени – ніжка.

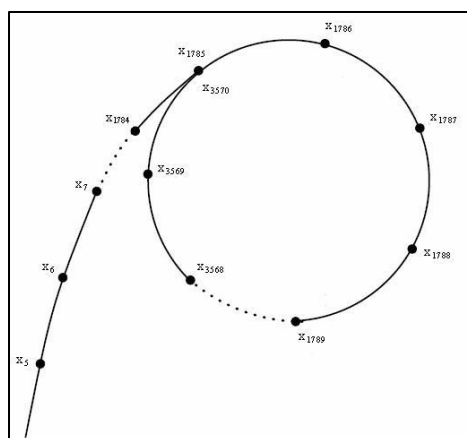


Рисунок 3.2 – Зациклення ряду

Алгоритм ρ -Полларда має кроки, які описані нижче.

Крок 1: вибираємо невелике число x_0 та будуємо послідовність $\{x_n\}$, $n = 0, 1, 2, \dots$, визначаючи кожне наступне як $x_{n+1} = F(x_n) \pmod{N}$;

Крок 2: одночасно на кожному i -ому кроці обчислюємо $d = \text{GCD}(N, |x_i - x_j|)$ для будь-яких i, j таких, що $j < i$, наприклад, $i = 2j$.

Крок 3: якщо виявили, що $d > 1$, то обчислення закінчується, і знайдене на попередньому кроці число d є дільником N . Якщо N/d не є простим числом, то

процедуру пошуку дільників можна продовжити, узявши в якості N число $N' = N/d$.

Функція $F(x)$ повинна бути не надто складною для обчислення, але в той же час не повинна бути лінійним многочленом, а також не повинна породжувати взаємно однозначне відображення. Тому у даній роботі для реалізації алгоритму р-Полларда була використана функція $F(x) = x^2 + a \pmod{N}$.

Послідовний алгоритм реалізується згідно з описаними вище крокам.

Головною ідеєю розподілення обчислень є вибір різного поліному для окремих потоків. Дана реалізація підвищує ймовірність розкладання факторизуємого числа на множники.

3.4 Опис програмних засобів

Під час моделювання програмного продукту було прийнято рішення створити додаток, який надає можливість факторизувати число за допомогою класичного алгоритму Шора, змодельованого квантового алгоритма Шора, алгоритма Ленстри, або алгоритма р-Полларда. На вхід програми повинно бути надано число для факторизації та вид алгоритму, який потрібно застосувати. На вихід буде надано два спів множника числа та додаткові параметри, такі як період функції, випадкове вибране число для знаходження періоду, час роботи алгоритму.

Головною задачею системи є оптимальна реалізація алгоритмів факторизації та проведення статистичного аналізу. Для реалізації підпрограм, які реалізують функціонал факторизації числа, використовуючи різні алгоритми, була обрана мова програмування Java версії 8. Мова програмування Java розроблена з підтримкою паралельних обчислень, і всі обчислення виконуються у контекстів потоку. На рис. 3.3 наведено схему функціональних цілей класів з пакету `java.util.concurrent`.

Платформа Java підтримує парадигму паралельного програмування. З першої версії в мові програмування Java є конструкції, які не належать до конкретної

бібліотеки такі як `synchronized`, `volatile` тощо. Також є спеціальні методи для синхронізації, які знаходяться у базовому класі `Object` з пакету `java.lang`. Це методи `wait`, `notify` тощо. Починаючи з п'ятої версії платформи Java був добавлений пакет `java.util.concurrent` з високорівневими елементами для паралельного програмування. А у версії 8 був разом з лямбда виразами був добавлений клас `CompletableFuture`, який зробив паралельне програмування у Java більш виразним та простим для розуміння.

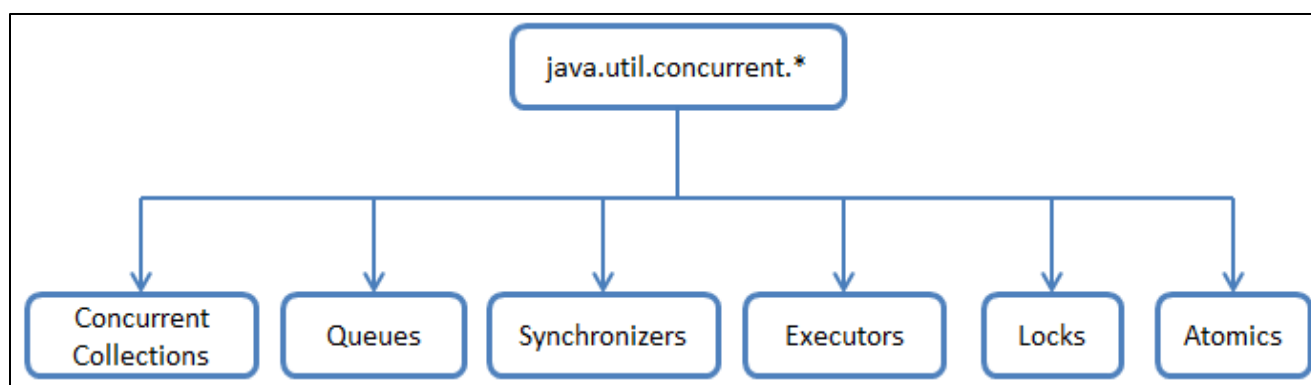


Рисунок 3.3 – Схема функціональних цілей `java.util.concurrent`

Основні класи, які використовуються у даній програмній системі, являються виконавцями та належать до `Executors`. На рис. 3.4 представлена діаграма основних класів, що використовуються для асинхронних результатів. Розглянемо основні з них:

- `Future<V>` — інтерфейс для отримання результатів роботи асинхронної операції. Ключовим методом тут є метод `get`, який блокує поточний потік (з таймаутом або без) до завершення роботи асинхронної операції в іншому потоці. Також, додатково існують методи для скасування операції і перевірки поточного статусу. Як імплементації часто використовується клас `FutureTask`. А починаючи за Java 8 `CompletableFuture`.

- `Callable <V>` - Розширений аналог інтерфейсу `Runnable` для асинхронних операцій. Дозволяє повертати типізоване значення і кидати `checked exception`. Незважаючи на те, що в цьому інтерфейсі відсутній метод `run()`, багато класів `java.util.concurrent` підтримують його поряд з `Runnable`.

– `CompletableFuture <V>` - реалізація `Future` та `CompletionStage`, яка дозволяє явно задавати стан `Future`, також дозволяє будувати з цих класів ланцюг виконання операцій (конвеєр). Ще містить статичні методи для побудови нових `CompletableFuture` поєднуючи інші `CompletableFuture`'и (наприклад можливо створити новий `CompletableFuture`, який буде чекати виконання усіх попередніх через метод `allOf`).

Для реалізації змодельованого квантового алгоритма Шора буде використана мова `Q#`. `Q#` (`Q Sharp`) - це квантово-орієнтована мова програмування з нативними типами, операторами і іншими абстракціями. `Q#` інтегрований з `Visual Studio` і `VS Code`, а також має сумісність з мовою програмування `Python`. Моделювання рішень, що вимагають менше 30 кубітів, відбувається на локальному комп'ютері, для рішень, які вимагають більшої кількості кубітів, можна використовувати симулятор на хмарному провайдері `Azure`. Також, `development kit` дозволяє займатися відладкою коду і оцінювати реальні затрати на виконання програми. `Q#` має ліцензію відкритого вихідного коду, яка дозволяє розробникам зі всього світу доповнювати вже готові рішення і ділитися їми з іншими. Мова програмування `Q#` виглядає не так, як більшість інших мов програмування, але, тим не менш, дуже схожа на `C#`. `Q-шарп` містить кілька цікавих примитивних типів. Додатково до стандартних типів, таких як `int`, `double`, `bool` і `string`, існують також типи `Pauli`, `Range`, `Result` і `Qubit`.

3.5 Проектування програмної системи

Програмна система для факторизації числа за допомогою різних алгоритмів факторизації написана на мові програмування `Java` з можливістю проводити основні операції через веб-інтерфейс. Також до системи треба віднести програму, написану на мові програмування для квантових обчислень `Q#`, яка дозволяє

змодельовати квантове виконання алгоритма Шора. Побудуємо діаграму послідовності.

Діаграми взаємодії використовуються в UML для моделювання динамічних аспектів систем. У загальних рисах діаграма взаємодії показує взаємодію ряду об'єктів, а також їх зв'язку і повідомлення, які можуть передаватися між ними. Здебільшого під моделюванням динамічних аспектів системи стосовно до діаграм взаємодії мається на увазі моделювання конкретних або прототипних прикладів класів, інтерфейсів, компонентів і вузлів поряд з переданими між ними повідомленнями – і все це в контексті сценарію, що ілюструє деяку поведінку. Діаграми взаємодії можуть існувати самостійно, щоб візуалізувати, уточнити, конструювати і документувати динаміку певних спільнот об'єктів, або ж використовуватися для моделювання одного конкретного потоку управління в межах варіанту використання.

Розглянемо діаграму послідовності, яка наведена на рис. 3.5.

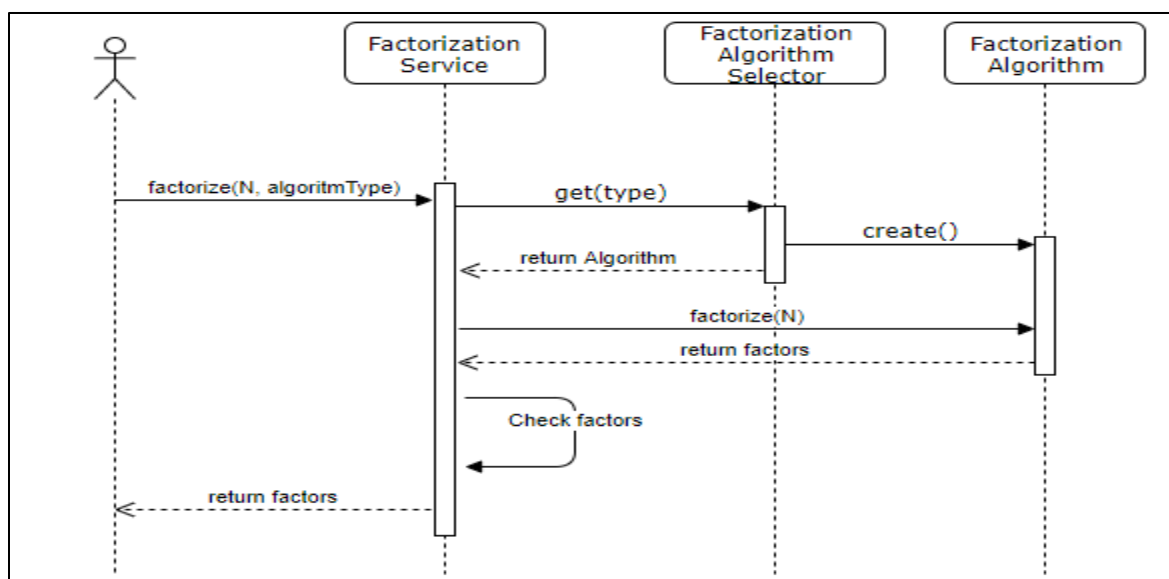


Рисунок 3.5 – Діаграма послідовності

Як бачимо з діаграми послідовності все починається з того, що клієнт ініціює увесь процес факторизації. Він повинен ввести число для факторизації та тип алгоритма, який повинен бути застосований. Далі ці дані потрапляють до FactorizationService, який спочатку отримує реалізацію вибраного клієнтом

алгоритма за допомогою класу `FactorizationAlgorithmSelector`. Також треба відмітити, що сам клас селектор є відповідальний за створення класу, реалізації `FactorizationAlgorithm`. Далі клієнтське число для факторизації передається до метода `factorize(BigInt)` класу `FactorizationAlgorithm`, у якому проводиться факторизації числа та у відповідь повертаються отримані фактори, які потім сервісом повертаються до клієнта.

Для реалізації зручного управління вибором алгоритму використовується патерн стратегія. Стратегія – це патерн поведінки, який визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Стратегія дозволяє змінювати алгоритми незалежно від клієнтів, які ними користуються. Цей патерн використовується у програмі, бо дозволяє використовувати єдиний клієнтський код для різних видів алгоритмів факторизації.

Розглянемо діаграму класів (рис. 3.6).

Клас `FactorizationService` інкапсулює у собі основну логіку системи, він вибирає та викликає алгоритм з отформатованими параметрами, оброблює результат в зручну для клієнта форму та повертає його. `FactorizationAlgorithm` – це інтерфейс, який задає функціонал алгоритму факторизації і який використовується `FactorizationService`'ом, реалізації якого є взаємозамінними у сервісі. Клас `AlgorithmSelector` є відповідальний за створення необхідної реалізації інтерфейса `FactorizationAlgorithm` то повернення його до сервісу.

На діаграмі класів наведено дев'ять різних об'єктів.

`ShorController` – це клас основне призначення, якого це обробка `http` запитів: приймання запиту, вибір з нього параметрів, виклик методу сервісу, відправка відповіді клієнту. `ClassicShor` – це класична реалізація алгоритму Шора. `ParallelClassicShor` – це теж класична реалізація алгоритму, але оптимізована через використання багатопоточності. `FermatFactorization` – це клас, який реалізує алгоритм факторизації Ферма. `ParallelFermatFactorization` – це реалізація алгоритма Ферма, але з використанням паралельних обчислень для більш швидкого знаходження факторів. `PollardFactorization` та `ParallelPollardFactorization` це класи, які реалізують алгоритм ρ -Полларда без використання паралельних обчислень та

оптимізований за допомогою паралельних обчислень відповідно. Код класів, які реалізують алгоритми факторизації наведені у додатку А.

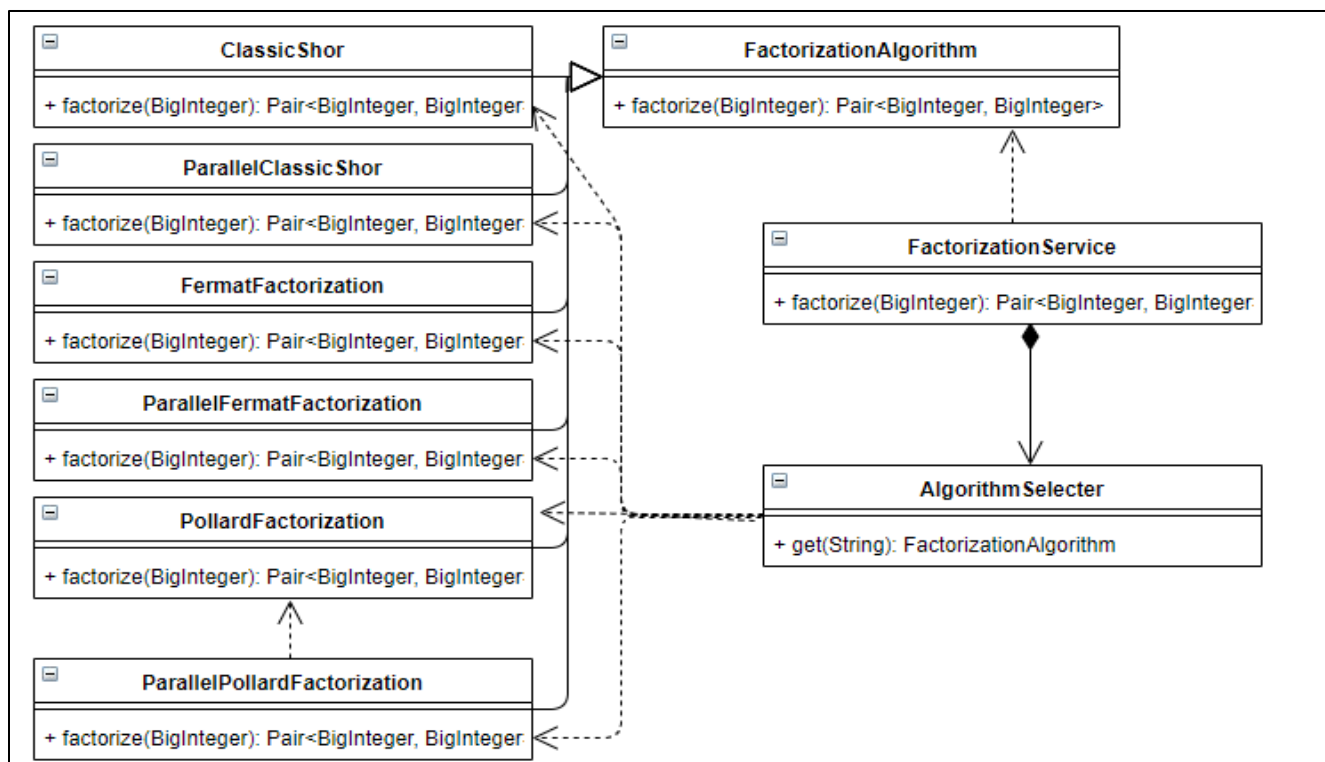


Рисунок 3.6 – Діаграма класів

Програма, яка реалізує квантовий алгоритм Шора та моделює його виконання написана на мові Q#. Структура цієї програми наведена на рисунку 3.7 та складається з двох основних файлів та залежних бібліотек.

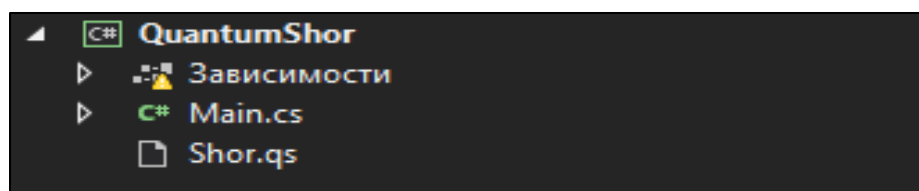


Рисунок 3.7 – Структура Q# програми для моделювання алгоритму Шора

Файл Main.cs написаний на мові програмування C# і є точкою входу до програми. Також цей файл відповідає за отримання вхідних даних від користувача, вимірювання часу, створення симулятора та усі інші підготовчі операції.

Файл `Shor.qs` написаний мові програмування `Q#` та містить у собі усю логіку алгоритма факторизації Шора. Головна частина цього файлу, операція `EstimatePeriod`, яка знаходить період функції $f(x):(a^x \bmod N)$ для алгоритму Шора. На вхід ця операція приймає два параметри a , випадково згенероване число, та N , число для факторизації. Спочатку визначаємо кількість бітів в модулі, щодо яких ми оцінюємо період та записуємо отримане значення до змінної `bitSize`. Дана змінна використовується, щоб підрахувати кількість кубітів, які знадобляться у алгоритмі. Так як алгоритм Шора є ймовірнісним, тому код знаходження періоду поміщений у цикл, який буде виконуватися, поки не буде знайдено вірне значення. В середині циклу спочатку створюються два регістри, які знаходяться у стані суперпозиції за допомогою команди `Qubit[qubitsize / 2]`. Далі до першої регістра ми застосовуємо гейт Адамара. Наступним кроком є створення оракула `DiscreteOracle(OrderFindingOracle(a, N, _, _))` та застосування його до регістрів `oracle.apply(register1, register2)`. Далі застосовуємо квантове перетворення Фур'є до першого регістра `QFT(register1)` и вимірюємо його значення `M(register1)`. Далі за допомогою отриманого значення знаходимо та перевіряємо період функції.

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Аналіз алгоритму Шора для класичного комп'ютера

Графічний інтерфейс користувача наведений на рисунку 4.1. Як бачимо з рисунку інтерфейс користувача уявляє собою текстове поле для введення числа, два випадаючих списки для вибору типу алгоритму та системи числення введеного числа та кнопка для запуску процесу факторизації у лівій частині екрана. У правій частині наведено панель, у якій відображаються результати факторизації: множники числа, випадково вибране число для алгоритму Шора (Basis), період функції у алгоритмі Шора та час виконання факторизації на сервері.

Factorising

calculation type:
Classic

numeral system:
10

number for factorize
9173503

factorize

Result

Factor1: 2579
Factor2: 3557
Period: 4583684
Basis (a): 2
Time: 4069 ms

Рисунок 4.1 – Графічний інтерфейс користувача

У попередньому розділі була описана класична реалізація алгоритму Шора. Проведемо тестування на правильність результатів, ефективність та проведемо дослідження роботи.

Спочатку згенеруємо випадковим чином вхідні дані (таблиця 4.1).

Таблиця 4.1 – Тестові дані для перевірки реалізації алгоритму Шора

№	Число для факторизації	Кількість бітів	Перший фактор	Другий фактор
1	360011	19	521	691
2	1434313	21	2753	521
3	9173503	24	2579	3557
4	84698077	27	14461	5857
5	889310809	30	11689	76081
6	1439949001	31	25391	56711

Так як класична реалізація алгоритму Шора складається з операцій, які є дуже затратними по часу, а також має експоненційну складність алгоритму, то вхідні числа мають невеликий розмір. Генерування чисел для факторизації виконувалося через знаходження двох випадкових простих чисел (першого та другого фактору), а потім отримання їх добутку.

Розглянемо результати (таблиця 4.2) роботи алгоритмічного модуля на тестових даних. Запуски програми проводилися на комп'ютері з 4 логічними ядрами та 2 фізичними.

Таблиця 4.2 – Результати роботи класичного алгоритму Шора

№	Число для факторизації	Випадковий параметр (a)	Період функції	Час виконання у послідовному режимі мс.	Час виконання у паралельному режимі мс.
1	360011	2	5980	26	9
2	1434313	2	89440	593	75
3	9173503	2	4583684	5375	2822
4	84698077	2	3528240	4340	1746
5	889310809	2	18525480	48506	22354
6	1439949001	11	143986690	942274	489336

З результатів бачимо, що для факторизації числа 1439949001 було затрачено майже 16 хвилин, тому запуск для більше великих чисел є занадто затратним по

часу і проводитися не буде. Також на основі таблиці результатів встановлено, що час виконання майже більше залежить від періоду функції та випадкового параметра ніж від кількості бітів у факторизуємому числі. Кількість бітів задає тільки межу для кількості операцій для найгіршого випадка. Побудуємо та розглянемо графік (рис 4.2) залежності часу виконання від періоду.

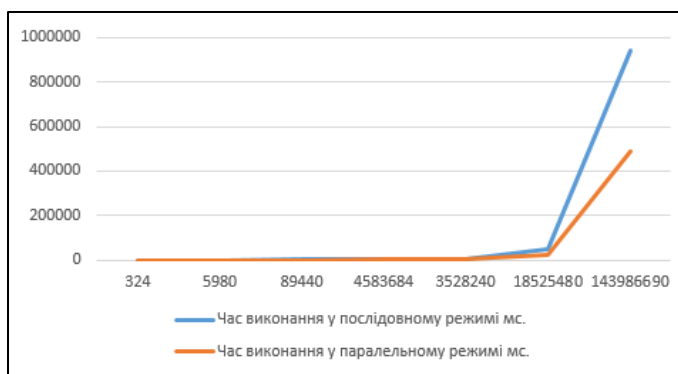


Рисунок 4.2 – Діаграма залежності часу виконання від періоду функції для класичного алгоритму Шора

З графіка бачимо, що час виконання експоненційно залежить від періоду і для послідовного режиму, і для паралельного. Також бачимо, що паралельний режим приблизно в два рази швидший за послідовний.

4.2 Аналіз алгоритму Ферма

У попередньому розділі було описано деталі реалізації алгоритму Ферма за допомогою мови програмування Java. Проведемо аналіз роботи отриманої програми.

Спочатку згенеруємо тестові дані (таблиця 4.3).

Дані були згенеровані шляхом отримання двох простих чисел та подальшого їх перемноження. Як бачимо, для аналізу алгоритму Ферма були взяті більше великі числа ніж для аналізу реалізації класичного алгоритму Шора. Це пов'язано

з тим, що алгоритм Шора є адаптованим квантовим алгоритмом, в той час, як алгоритм Ферма оснований на класичних обчисленнях.

Таблиця 4.3 – Тестові дані для перевірки реалізації алгоритму Ферма

№	Число для факторизації	Кількість бітів	Перший фактор	Другий фактор
1	2326505459	32	63277	36767
2	46766504593	36	204509	228677
3	483942319181	39	797273	606997
4	14722750530251	44	3691949	3987799
5	189314357834207	48	13743797	13774531
6	3390521606369323	52	61601773	55039351
7	50152109240242930	56	248899039	201495793
8	885719410306513319	60	952213337	930169087

Розглянемо результати (таблиця 4.4) роботи реалізованого алгоритму Ферма на тестових даних. Запуски програми проводилися на комп'ютері з 4 логічними ядрами та 2 фізичними. Перевірка правильності виконання факторизації проводилась шляхом порівняння отриманих факторів числа с факторами, які були отримані під час генерування випадкових даних.

Таблиця 4.4 – Результати роботи алгоритму Ферма

№	Число для факторизації	Кількість бітів	Час виконання у послідовному режимі, мс	Час виконання у паралельному режимі, мс
1	2326505459	32	30	64
2	46766504593	36	85	101
3	483942319181	39	221	336
4	14722750530251	44	54	76
5	189314357834207	48	4774	3549
6	3390521606369323	52	24581	19926
7	50152109240242930	56	101478	71308
8	885719410306513319	60	2626471	1784782

З отриманих результатів бачимо, що паралельні обчислення починають давати вигравш у часі починаючи з чисел розміром 48 біта та більше.

Факторизація числа розміром 60 біт зайняла майже 44 хвилини в той час, як у паралельному 30, тобто використання паралельних обчислень дало вигоду майже у 32% при факторизації даного числа. Також слід звернути увагу на час виконання факторизації числа 14722750530251, який складає 54 мілісекунди у послідовному режимі та 76 мілісекунд у паралельному і є значно меншим за час факторизації попереднього числа, 221 мілісекунд та 336 мілісекунд відповідно. Це пов'язано з особливістю алгоритму Ферма, який дозволяє дуже швидко проводити факторизацію числа, якщо його фактори знаходяться досить близько один до одного. Побудуємо графік (рис. 4.3) залежності часу факторизації числа від його розміру та проведемо його аналіз.

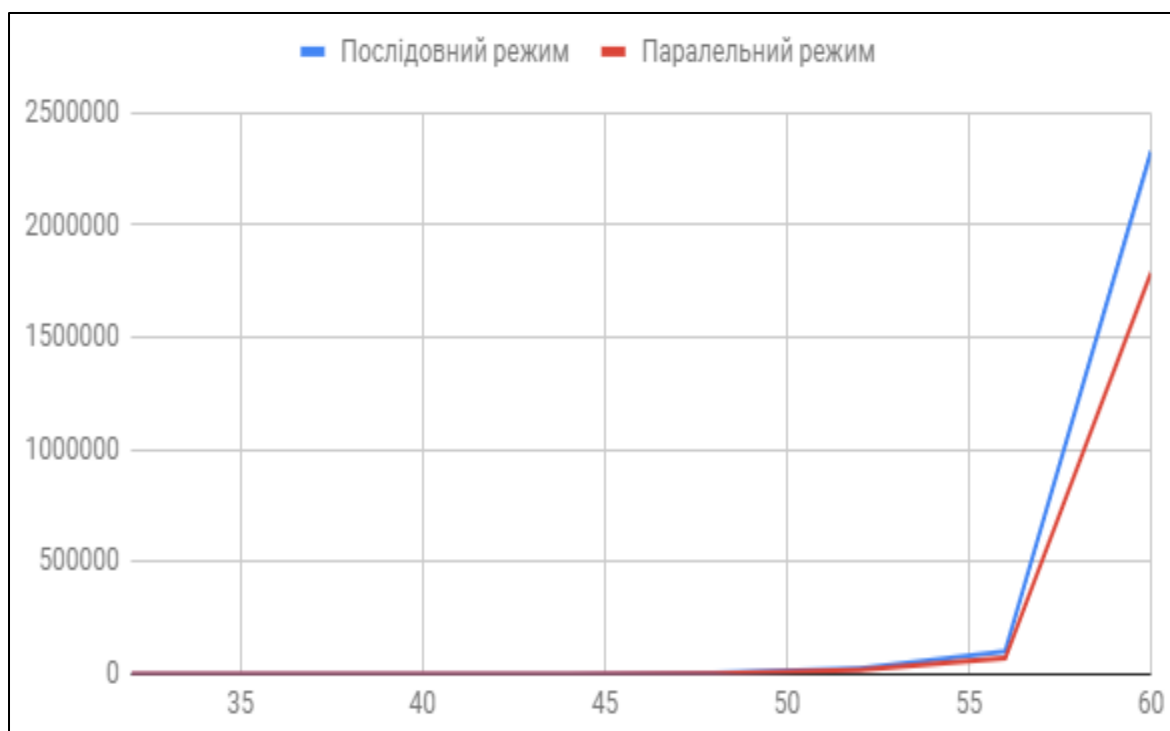


Рисунок 4.3 – Діаграма залежності часу виконання від розміру числа для алгоритму Ферма

З графіка бачимо, що час виконання експоненційно залежить від розміру числа і для послідовного режиму, і для паралельного.

4.3 Аналіз алгоритму ρ -Полларда

У попередньому розділі було описано деталі реалізації алгоритму ρ -Полларда за допомогою мови програмування Java. Проведемо аналіз роботи отриманої програми.

Спочатку згенеруємо тестові дані (таблиця 4.5).

Таблиця 4.5 – Тестові дані для перевірки реалізації алгоритму ρ -Полларда

№	Число для факторизації	Кількість бітів	Перший фактор	Другий фактор
1	3419400499457568584899	72	49939553869	68470785871
2	46067572131075492428033	76	243729076673	189011392321
3	726189210599864879544499	80	1041794164513	697056323923
4	9554785874640408115372229	83	2356576236929	4054520165701
5	179834419632725580066084077	88	12139680186583	14813769133019
6	2995240840957667566263635413	92	47690239617751	62806160442163
7	37257558670911150949610042479	95	241164844929319	154490007371641
8	814916356174804576740926024243	100	1059811251084941	768925934066623
9	7769294326016627540314714751953	103	3098361411244283	2507549409123491

Розглянемо результати (таблиця 4.6) роботи реалізованого алгоритму ρ -Полларда на тестових даних. Умови виконання були таким ж самими, як і при аналізі реалізацій класичного алгоритму Шора та алгоритму Ферма.

З отриманих результатів бачимо, що факторизація числа розміром 103 біти була виконана трохи більше ніж за 3 хвилини, що є набагато кращим результатом за алгоритм Ферма. Також треба відмітити, що дуже часто паралельна реалізація є

більш повільною за паралельну. Це пов'язано з тим, що на початку алгоритму р-Полларда нам треба згенерувати два випадкових числа, від яких дуже сильно залежить час подальшого виконання програми. Так як паралельний режим цього алгоритму уявляє собою лише запуск у кожному потоці послідовної версії з різними початковими випадковими числами, то існує можливість того, що при виконанні послідовної версії випадкові числа будуть "кращі" ніж в усіх потоках при паралельному виконанні. Також слід відмітити, що виконання програми у паралельному режимі накладає додаткові витрати часу на організацію паралельного виконання. Таким чином робимо висновок, що розпаралелювання даного алгоритму має сенс тільки при великій кількості доступних потоків.

Таблиця 4.6 – Результати роботи алгоритму р-Полларда

№	Число для факторизації	Кількість бітів	Час виконання у послідовному режимі, мс	Час виконання у паралельному режимі, мс
1	3419400499457568584899	72	433	736
2	46067572131075492428033	76	1912	3013
3	726189210599864879544499	80	251	3237
4	9554785874640408115372229	83	7435	10673
5	179834419632725580066084077	88	10168	16083
6	2995240840957667566263635413	92	30476	9350
7	37257558670911150949610042479	95	81326	41267
8	814916356174804576740926024243	100	108254	210845
9	7769294326016627540314714751953	103	261663	191769

Побудуємо графік (рис. 4.3) залежності часу виконання факторизації тестового числа від розміру цього числа у бітах та проведемо аналіз графіка.

З графіка бачимо, що час виконання експоненційно залежить від розміру числа i для послідовного режиму, і для паралельного.

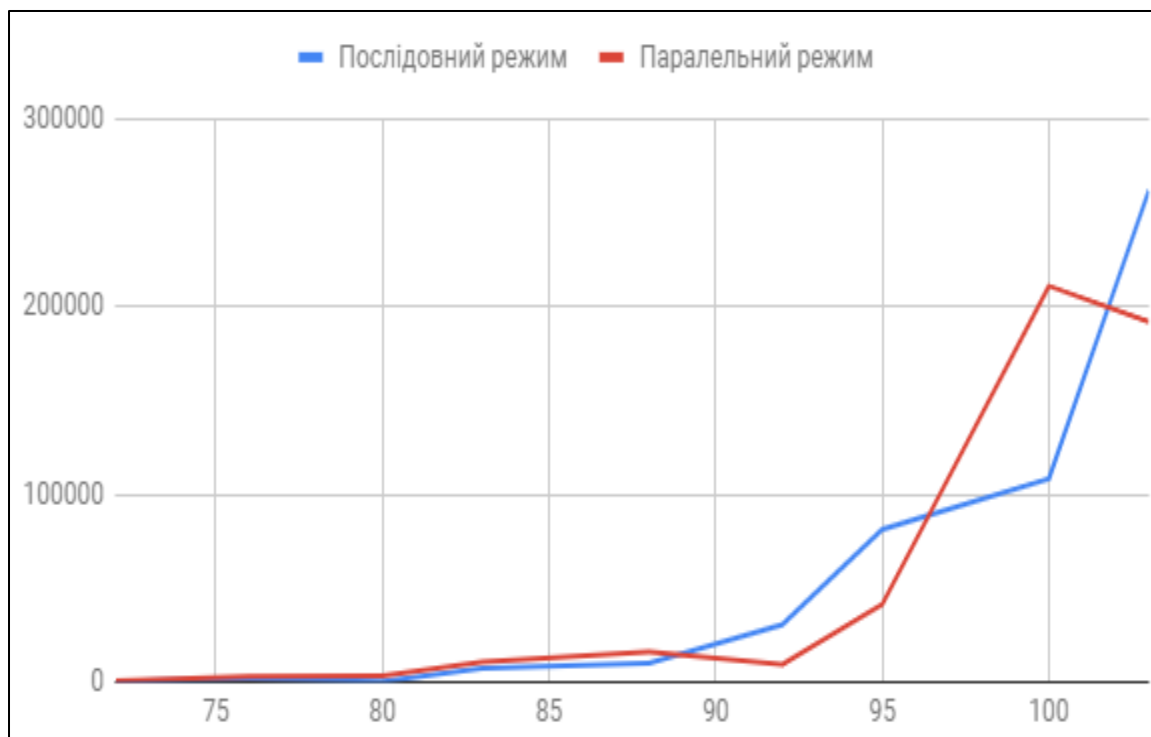


Рисунок 4.4 - Діаграма залежності часу виконання від розміру числа для алгоритму р-Полларда

Також треба відмітити, що графік не є постійно зростаючим, що є наслідком того, час виконання може бути досить малим при деяких випадкових числах, які дозволяють послідовності швидше зациклитись.

4.4 Аналіз змодельованого алгоритма Шора

У попередньому розділі було описано деталі реалізації квантового алгоритму Шора за допомогою мови програмування Q#. Проведемо аналіз роботи отриманої програми. Запуски програми проводилися на комп'ютері з 4 логічними ядрами та 2 фізичними. Тестовими даними були обрані числа 15, 21 та 35. Також були

проведені запуски програми для числа 65, але не вдалось дочекатися кінця її виконання, при кожному запуску програма працювала 30 хвилин (ліміт часу) та не могла за цей час факторизувати число. Це пов'язано з тим, що для числа 65 потрібно у два рази більше кубітів ніж для числа 35, що призводить до того, що квантовому симулятору потрібно набагато більше часу на виконання кожної операції.

Розглянемо результати вимірювань (таблиця 4.7) та проведемо їх аналіз.

Таблиця 4.7 – Результати змодельованого алгоритму Шора

Число	Розмір, біт	Номер запуску	Час виконання, мс	Випадковий параметр (a)	Кількість запусків оракула	Середній час, мс
15	4	1	45137	8	2	28137
		2	40462	8	2	
		3	17945	7	1	
		4	20391	7	1	
		5	16753	4	1	
21	5	1	136805	10	4	73878
		2	46294	13	1	
		3	47127	13	1	
		4	92065	10	2	
		5	47101	8	1	
35	6	1	141813	31	1	217344
		2	421745	31	4	
		3	150665	13	1	
		4	146714	29	1	
		5	225783	23	2	

З результатів вимірів бачимо, що існує залежність часу виконання від розміру числа у бітах та кількості запусків оракула у випадках, коли було знайдено

неправильний період функції. Також бачимо, що час виконання не залежить від випадкового параметра a .

Побудуємо графіки залежності мінімального та середнього часів виконання програми від розміру факторизуємого числа у бітах (рис. 4.5). Треба відмітити, що середній час розглядається тому, що квантовий алгоритм Шора є ймовірнісним і розглядання тільки мінімального часу є неправильним.

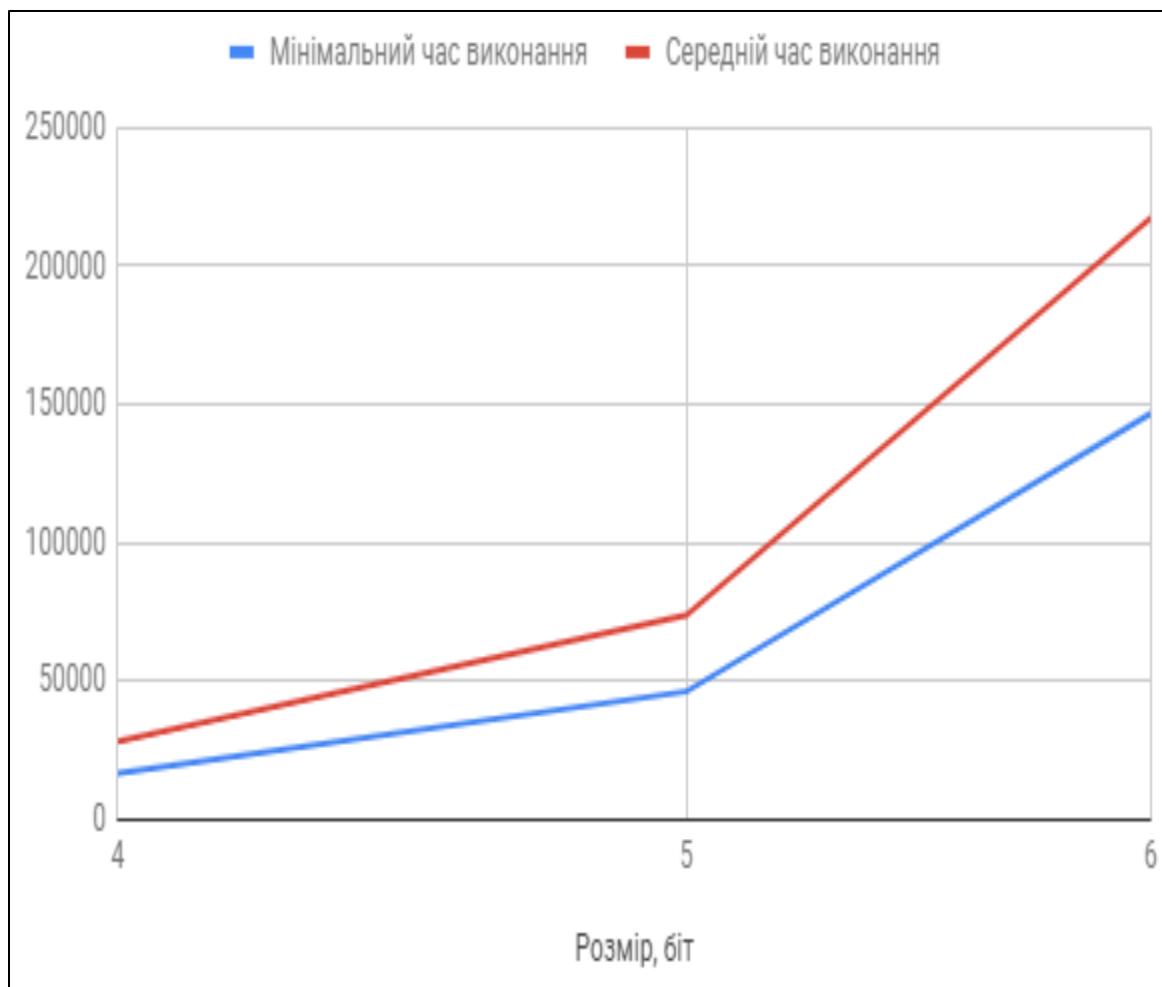


Рисунок 4.5 – Діаграма залежності часу виконання від розміру числа для змодельованого квантового алгоритму Шора

З графіка бачимо, що час виконання має тенденцію до експоненційної залежності від розміру числа.

4.5 Порівняльний аналіз алгоритмів факторизації

Проведемо порівняльний аналіз досліджуваних алгоритмів факторизації. Побудуємо графік залежності часу, який потрібен на виконання алгоритму, від розміру вхідного числа (рис 4.6).

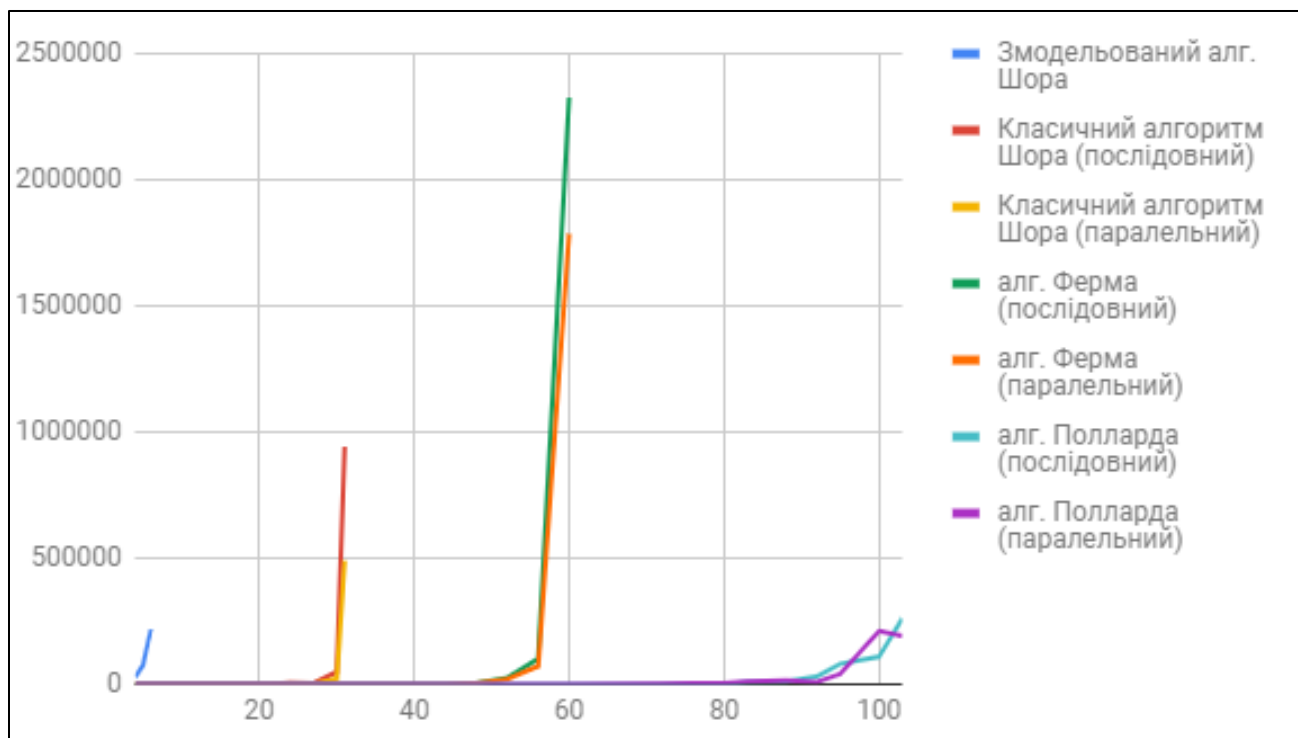


Рисунок 4.6 - Діаграма залежності часу виконання від розміру числа для досліджуваних алгоритмів

З графіку бачимо, що для усіх алгоритмів залежність часу виконання від розміру числа для факторизації є експоненціальною. Час виконання набагато більше залежить від типу алгоритму факторизації, ніж від режиму виконання програми (послідовний або паралельний). Найгіршою є реалізація змодельованого квантового алгоритму Шора за допомогою мови програмування Q#. Це пов'язано з тим, що багато ресурсів йде на модулювання роботи квантового комп'ютера і неможливості досягнути реального часу виконання квантових операцій на транзисторному комп'ютері. Передостаннє місце займає адаптована реалізація

алгоритму Шора для сучасного неквантового комп'ютера. Погані результати пов'язані з тим, що під час адаптації алгоритму операції, які повинні надавати найбільший виграш у часі, були замінені на класичні, які не можуть досягти гарної продуктивності. Наведені реалізації алгоритму Шора не є конкурентоспроможними порівняно з сучасними рішеннями, але можуть бути використанні при навчанні квантовим обчисленням для подальшого використання на справжніх квантових комп'ютерах. Другим по продуктивності є алгоритм Ферма. Отриманні результати пов'язані з тим, що цей алгоритм сильно залежить від відстані між факторами числа. Число, фактори якого розташовані достатньо близько, може бути швидко факторизоване, але якщо фактори розташовані на великій відстані, то і час факторизації буде великий. Найкращим досліджуваним алгоритмом для факторизації числа є алгоритм ρ -Полларда. Він зміг факторизувати числа розміром 103 біти за прийнятний час, який дорівнює 3 хвилини. Але оптимізація даного алгоритму, за допомогою паралельних обчислень не дала суттєвого виграшу.

ВИСНОВКИ

У ході виконання атестаційної роботи було досліджено проблему факторизації числа в умовах використання квантових комп'ютерів. Були розглянуті класичний та квантовий алгоритми Шора, алгоритм Ферма та алгоритм ρ -Полларда. Також були розглянуті особливості квантових алгоритмів, проблеми квантових обчислень, таких як когерентизація, клонування, перегляд поточного стану регістрів, імовірнісний характер обчислень. Було досліджено питання квантових комп'ютерів сьогодні, було відмічено посилений розвиток постквантових рішень та розглянуто успіхи у реалізації квантових комп'ютерів таких організацій, як D-Wave Systems, Google та IBM.

Були проведені детальні аналізи квантового та класичного алгоритму Шора, алгоритму Ферма та алгоритму ρ для факторизації числа та складені покрокові алгоритми для реалізації на комп'ютері. Також була розглянута оптимізація класичних алгоритмів шляхом використання паралельних обчислень. Були написані та опубліковані тези до 23-го міжнародного форуму "Радіоелектроніка та молодь у XXI столітті".

У результаті виконання атестаційної роботи було розроблено два додатки. Перший додаток дозволяє факторизувати число за допомогою алгоритмів класичного Шора, Ферма та ρ -Полларда. Інтерфейс клієнтської частини був розроблений простим та зручним. Кожен алгоритм може факторизувати число в послідовному та паралельному режимах. Для виконання паралельних обчислень використовувалася бібліотека `java.util.concurrent`. Другий додаток дозволяє моделювати факторизацію числа за допомогою алгоритму Шора на симуляторі квантового комп'ютера. Для розробки цього додатка використовувалась мова програмування Q#.

Аналіз класичного алгоритму Шора на тестових вхідних даних встановив, що час виконання сильніше залежить від періоду і з меншою силою залежить від розмірності числа. Залежність від періоду є експоненційною і для послідовного

режиму, і для паралельного. Також бачимо, що паралельний режим приблизно у два рази швидший за послідовний.

Аналіз реалізації алгоритму Ферма на тестових вхідних даних встановив, що час виконання залежить від розміру числа і залежність є експоненційною для послідовного та паралельних режимів. Також було встановлено, що алгоритм Ферма швидше знаходить результат, якщо фактори числа знаходяться близько один до одного. Було проведено оптимізацію шляхом використання паралельних обчислень, що дозволило прискорити роботу алгоритму майже у півтора рази.

Аналіз реалізації алгоритму ρ -Полларда на тестових вхідних даних встановив, що час виконання залежить від розміру числа і залежність є експоненційною для послідовного та паралельних режимів. Було проведено оптимізацію шляхом використання паралельних обчислень, але це не дало суттєвого прискорення тому, що алгоритм ρ -Полларда використовує випадкові числа на початку роботи і більше залежить від їх якості, а його оптимізація шляхом використання паралельних обчислень уявляє собою запуск усього алгоритма в кожному потоці з різними випадковими початковими параметрами.

Аналіз змодельованої роботи квантового алгоритму Шора встановив, що час виконання залежить від розміру числа у бітах та від кількості запусків оракула у випадках, коли було знайдено неправильний період функції. На даний час змодельований квантовий алгоритм Шора можна використовувати тільки на невеликих числах та з великими програшами у часі, що робить його неконкурентноспроможним у порівнянні з досліджуваними алгоритмами класичним Шора, Ферма та ρ -Полларда.

Було встановлено, що усі досліджувані алгоритми мають експоненційну складність. Тому мають дуже великі програши у часі при збільшенні розміру факторизуємого числа. Найкращим з досліджуваних алгоритмів є алгоритм ρ -Полларда, який показав найкращий час при факторизації чисел розміром у 100 біт.

ПЕРЕЛІК ДЖЕРЕЛ

1. Квантовые компьютеры: к чему готовится лично вам [Электронный ресурс] / Kaspersky lab – daily.- Режим доступа: — 02.12.2016 р. – Загл. з екрану.
2. Rivest R., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems / R. Rivest – New York City: ACM, 1978. — p.126.
3. A.K. Lenstra, H.W. Lenstra, Jr. The Factorization of the Ninth Fermat Number / A.K. Lenstra.- New York City: ACM, 1993. — p.246.
4. Тесленко Н.О. Криптоаналіз методів несиметричного шифрування [Текст]: диплом/ Н.О. Тесленко.-Харьків.:ХНУРЕ, 2016. – 71 с.
5. Функция односторонняя [Электронный ресурс] / Математическая криптография - Режим доступа : http://cryptography.ru/ref/функция_односторонняя/ - 09.01.2017 г. - Загл. с экрана.
6. Simmons G. I. "Cryptology", Encyclopedia Britannica, 16th edition / G. I. Simmons - Sandia National Laboratories 1986. - p.913.
7. Анохін М. А., Варновскій Н. П. Криптографія в банківській справі / М. А. Анохін.-М:МІФІ, 1997.- 174 с.
8. Ишмухаметов Ш.Т., Методы факторизации натуральных чисел [Текст] : учеб. пособие / Ш.Т. Ишмухаметов.— Казань:Казан.ун., 2011.- 201с.
9. Герман О.Н. Теоретико-числовые методы в криптографии[Текст] / О.Н. Герман, А.Ю. Нестеренко. — М.:Академия, 2012. — 300 с.
10. Bressoud D. M. Factorization and Primality Testing [Text] /D. M. Bressoud. — New York: Springer-Verlag, 1989. — 260 p.
11. Richard P.Brent Some parallel algorithms for integer factorisation [Text] / P. Richard. — Oxford:Oxford University, 1999. — 27p.
12. Lenstra A. K. Factoring polynomials with rational coefficients [Text] / A. Lenstra, H. Lenstra, S. Lovaz.-Luxembourg: Springer Science Business Media, 1982. — 208 p.
13. Василенко О. Н. Теоретико-числовые алгоритмы в криптографии / О. Н. Василенко. – М.: МЦНМО, 2003. – 328 с.
14. М.Г. Адигеев Введение в теорию сложности [Текст] / М.Г. Адигеев.- Ростов-на-Дону:РГУ, 2004.-35с.

15. Эббинхауз Г.Д. Машины Тьюринга и рекурсивные функции/ Г.Д. Эббинхауз. – М.: Мир, 1972. – 264 с.
16. Томас Х. Алгоритмы. Построение и анализ [Текст]/ Х. Томас - М.:Вильямс, 2016.-1328с.
17. Роман Душкин Квантовые вычисления и функциональное программирование/ Р. В. Душкин.-М.: ДМК Пресс, 2015. – 232 с.
18. В.Н. Ручкин Естественный параллелизм квантовых компьютеров и нейровычислителей/ В.Н. Ручкин, В.А. Романчук, В.А. Фулин.- Рязань.:Рязанский государственный университет им. С.А. Есенина, 2013 – 387 с.
19. С.А. Дуплий. Квантовая информация, кубиты и квантовые алгоритмы / С. Дуплий, В. Калашников, Е. Маслов. – Харьков.:Харківський національний університет ім. В. Н. Каразіна, 2005.- 217 с.
20. Дж.Прескилл Квантовая информация и квантовые вычисления / Д. Прескилл.-М.:Ижевск, 2008. - 464 с.
21. П.А. Правильщиков Квантовый параллелизм и решение уравнений в задачах управления на базе новой модели вычислений / П.А. Правильщиков.- М.:Институт проблем управления им. В.А. Трапезникова, 2014.- 179 с.
22. Wootters W.A Single Quantum Cannot Be Cloned [Text] / Wootters W., Zurek W.//Nature.-1982.-№5886.-803p.
23. Bennett C. Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels [Text] /Bennett C., Brassard G., Crépeau C.//Physical review letters.-1993.-№13.-1895-1899pp.
24. А.Г. Грозин Квантовый компьютер для чайников/ А.Г. Грозин.- Новосибирск.:Препринт, 2004.-240 с.
25. К.А. Валиев Квантовые компьютеры и квантовые вычисления / К.А. Валиев.-М.:Insitute of Physics and Technology, 2005.- 387с.
26. Shor P. Algorithms for quantum computation: discrete logarithms and factoring [Text] /Shor P.// Foundations of Computer Science.—1994.—№10. —134p.
27. Алгоритм Шора, его реализация на языке Haskell и результаты некоторых опытов [Электронный ресурс]/Записки программиста.- Режим доступа : <http://eax.me/shors-algorithm> - 29.12.2014 г. – Загл. з экрана.