Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

На правах рукопису

УДК 004.42

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: “Програмне забезпечення квантових обчислень в навчальному процесі”
Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
_____Іван Кирилов
(підпис) (розшифровка підпису) (дата)

Науковий керівник
_____к.т.н., доцент Курченко Олег
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

_____
(підпис) (розшифровка підпису) (дата)

Київ - 2022

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

професор, доктор техн. наук Бондарчук Андрій

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри програмних систем і технологій
_____(О.С.Бичков)
“____”_____ 20___р.

ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
СТУДЕНТУ

_____Кирилов Іван Ігорович_____
(прізвище, ім’я, по батькові)

1. Тема випускної кваліфікаційної магістерської роботи “Програмне забезпечення квантових обчислень в навчальному процесі”
затверджені наказом вищого навчального закладу від „__”_____20__р. №_____

2. Строк здачі студентом закінченої роботи _____
3. Вихідні дані до роботи теоретичні концепції алгоритму компіляції квантових схем, опис методів поступового підвищення результатів обчислень, опис методів розробки квантових обчислень.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
1. Аналіз існуючих рішень компіляції та розпаралелення квантових схем.
2. Дослідження алгоритму моделей компіляції, допоміжних технологій та інструментів вирішення поставленої задачі.
3. Розробка методів розпаралелення та оптимізації квантових схем.
4. Розробка програмного забезпечення.
5. Результати досліджень і розробки, аналіз та пошук можливих рішень вдосконалення розробленої системи.
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)
1. Діаграма класів складності при припущенні, що P = NP (рис. 1.1)
2. Ілюстрація закону Амдала(рис. 1.2)
3. Загальна квантова схема. Кожна горизонтальна лінія позначає «провід» (рис. 2.1)
4. Розрахунок глибини квантового контуру (рис. 2.2)
5. Квантова схема n-кубітного QFT-алгоритму (рис. 2.3)
6. Квантова схема для QTF, яка використовує лише вентилі J() і Z. (рис. 2.4)
7. Граф MBQC перед тим, як ми додамо вентиль J(α), який діє на провід i. (рис. 3.1)

6. Консультанти з роботи із зазначенням розділів роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Аналітико-теоретична частина: розділ 1,2,3	О.А. Курченко		
Практична частина: розділ 4	О.А. Курченко		
Аналіз та перспектива: розділ 5	О.А. Курченко		

7. Дата видачі завдання _____

Керівник _____ О.А. Курченко
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ І.І. Кирилов
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Вивчення на аналіз наявних рішень компіляції		виконано
2. Опис алгоритму розпаралелення		виконано
3. Пошук методів підвищення точності результатів		виконано
5. Побудова архітектури програмного забезпечення		виконано
6. Розробка першого наближення додатку		виконано
7. Розробка інструментів для авторизації студентів		виконано
8. Аналіз, пошук напрямків вдосконалення системи		виконано
9. Оформлення пояснювальної записки		виконано

Студент – магістр _____ І.І. Кирилов
(підпис) (розшифровка підпису)

Керівник роботи _____ О.А. Курченко
(підпис) (розшифровка підпису

АНОТАЦІЯ

Випускна кваліфікаційна магістерська робота: 74 сторінок, 35 рисунків, 2 таблиці, 2 додатки, 13 джерел.

Тема: програмне забезпечення квантових обчислень в навчальному процесі.

Об'єктом дослідження - процес авторизації та компіляції квантових схем.

Мета роботи полягає у підвищенні ефективності навчального процесу за допомогою розробки програмного забезпечення квантових обчислень.

Предметом дослідження - методи і алгоритми компіляції квантових схем та авторизації.

Результати дослідження:

Проаналізовано проблематику компіляції квантових схем для архітектур NISQ та паралельної обробки вентилів. Модернізовано алгоритм авторизації студентів та компіляції квантових схем.

Висновок

В ході дослідження було розроблено алгоритм який переводить квантовий контур в модель MBQC і виконує оптимізацію MBQC для більш ефективних квантових обчислень за допомогою паралельної обробки. Та алгоритм авторизації студентів за допомогою децентралізованої блокчейн-системи, що підвищить ефективність квантових обчислень в навчальному процесі.

КВАНТОВІ ОБЧИСЛЕННЯ, ГЕНЕРАЦІЯ КВАНТОВИХ СХЕМ, ПАРАЛЕЛЬНА ОБРОБКА, КВАНТОВІ ВЕНТИЛІ, MBQC, АВТОРИЗАЦІЯ

ANNOTATION

The final qualifying master's thesis: 74 pages, 35 figures, 2 tables, 2 appendices, 13 sources.

Topic: software for quantum computing in the learning process.

The object of research is the process of authorization and compilation of quantum schemes.

The purpose of the work is to increase the efficiency of the educational process through the development of quantum computing software.

The subject of research - methods and algorithms for compiling quantum schemes and authorization.

Results of the research:

The problems of compilation of quantum schemes for NISQ architecture and parallel processing of valves are analyzed. The algorithm of student authorization and compilation of quantum schemes has been modernized.

Conclusion

The study developed an algorithm that translates a quantum loop into an MBQC model and implements MBQC optimization for more efficient quantum computations using parallel processing. And an algorithm for student authorization using a decentralized blockchain system that will increase the efficiency of quantum computing in the learning process.

QUANTUM CALCULATIONS, QUANTUM CIRCUIT GENERATION, PARALLEL PROCESSING, QUANTUM VALVES, MBQC, AUTHORIZATION

ЗМІСТ

ВСТУП

Актуальність роботи

Сучасні квантові обчислення на декілька порядків продуктивніше виконують низку широко використовуваних алгоритмів у порівнянні з сучасними класичними обчислення, що побудовані на напівпровідникових логічних вентилях. Тому розробка алгоритмічно-програмного забезпечення для квантових комп'ютерів – це актуальна наукова і прикладна проблема.

Через фізичні обмеження поточної технології стан кубітів (квантових бітів) може залишатися стабільним лише короткий час. Протягом тривалого часу кубіти спонтанно змінюють свій стан. Це означає, що алгоритми, запущені на квантовому комп'ютері, повинні будуть закінчитися за короткий час. Це спонукало дослідження спрощення квантових алгоритмів та їх представлення у вигляді квантових схем.

Під час написання роботи, квантові обчислення є багатовартісною технологією через проблеми шумів та підтримки стану суперпозиції. Тому потрібно ефективно використовувати весь можливий час для компіляції та доступу до компонента, який реалізує квантові обчислення, щоб покращити навчальний процесу.

Порівняння роботи з відомими розв'язаннями проблеми

У дослідженні глибини квантового контуру Клів і Ватроус [2] показали, що обчислення приблизного значення QFT можна розпаралелювати до квантової схеми з послідовними кроками $O(log_n)$, де n — кількість кубітів. Це дослідження було зосереджено на одному конкретному алгоритмі, а не на загальних квантових схемах.

Для оптимізації вільних квантових схем Маслов, Дуек, Міллер і Негревернь [3] запропонували та реалізували метод, який використовує локальну оптимізацію. Їхній метод використовує визначені шаблони для заміни неоптимальних конструкцій у схемах на більш оптимальні.

Мета і задачі дослідження

Мета роботи полягає у підвищенні ефективності навчального процесу за допомогою розробки програмного забезпечення квантових обчислень.

Квантові обчислення – це сфера, яка швидко розвивається і, як очікується, зробить прорив у багатьох областях. Квантові обчислення використовують принципи квантової механіки для обробки інформації і мають потенціал для виконання конкретних завдань набагато швидше, ніж класичні обчислення.

Основною проблемою розробки програмного забезпечення квантових обчислень є процес компіляції та велика кількість помилок при обчисленнях. Також обладнання, яке реалізовує модель квантових обчислень, є дорогим та складним в обслуговуванні. Тому постає дві проблеми, які заважають підвищити ефективність та якість навчального процесу:

1) Швидкість компіляції квантових схем та обробка помилок.
2) Авторизація та доступ до обладнання великої кількості студентів.

Об’єктом дослідження: процес авторизації та компіляції квантових схем.

Предметом дослідження: методи і алгоритми компіляції квантових схем та авторизації.

Методи дослідження

Для дослідження існуючих алгоритмів компіляції квантових схем та формування підзадач використовуються загальнонаукові методи аналізу, синтезу та моделювання. Для модернізації ПЗ в рамках даної роботи використовуються евристичні методи програмної інженерії, що включають методи об’єктно-орієнтованого проектування (UML) та аспектно-орієнтованого програмування. Для аналізу якості ПЗ використовуються емпіричні методи програмної інженерії, а саме статистичний аналіз метрик (статичний аналіз коду).

Наукова новизна отриманих результатів

Реалізовано метод паралельної обробки вентилів та авторизацію за допомогою блокчейну. Запропоновано метод компіляції квантової схеми для архітектур NISQ (QCC-NISQ). Ключовими компонентами цього методу є синтезатор схем і візуалізаторів та метод паралельної обробки вентилів. Ці компоненти забезпечують важливу підтримку навчального процесу для подальшого розвитку практичних квантових комп’ютерів і дослідження квантових алгоритмів.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд літератури

1.1.1. Квантові обчислення

Сучасні обчислювальна техніка, використовує мікропроцесори на основі транзисторів, які підпорядковуються відкритому емпіричному закону Мура, згідно з яким кожні 2 роки кількість транзисторів на кристалі подвоюється. Виходячи з закону, технологічний процес постійно вдосконалюється і завдяки цьому зменшуються розміри транзисторів. На сьогодні розробляються основи 3-5 нанометрового процесу виробництва. З цього постає питання, чи можуть обчислювальні можливості мікропроцесорів, на основі транзисторів, зростати необмежено? На жаль, відповідь на це питання негативна. Основна проблема в розмірі транзистора, з одного атома його створити неможливо. Також є проблема в класичних алгоритмах та способах збереження інформації, які вирішують завдання зі складністю “NP” (рис. 1.1). Вони потребують великі розміри пам'яті та часу для їх вирішення, наприклад, розкладання чисел на множники, задача комівояжера, повне моделювання еволюції молекули ДНК (яка складається з кількох сотень мільярдів елементарних частинок, і ми будемо змушені керувати одночасно такою кількістю елементів пам’яті, що перевищує кількість елементарних частинок у Всесвіті). Виходячи з цього, якщо була б створена технологія транзистора розміром в один атом, ряд принципових завдань все одно буде недоступним для класичних обчислень. Через ці проблеми, постає необхідність у зміні принципів виконання обчислень.

Наразі успішно розвивається та використовується, розпаралелювання обчислень (багатоядерні процесори, розподілені обчислювальні системи тощо). Однак і в цьому способі є принципові обмеження, наприклад, закон Амдала стверджує, що час виконання обчислень паралельною системою не може бути

меншим за час виконання найдовшого фрагмента коду (якщо ми розпаралелимо 95% усього коду, то не досягнемо більше як 20-разового прискорення обчислень (рис. 1.2.)).

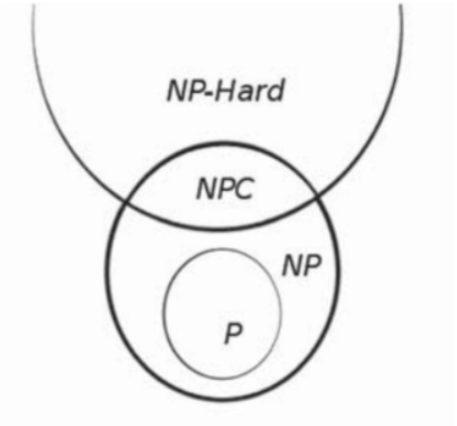


Рис. 1.1. Діаграма класів складності при припущенні, що P = NP

Альтернативою класичних обчислень є використання принципів квантових обчислень, які засновані на принципі квантової суперпозиції, явища квантової заплутаності, правилі Борна та можливості оборотності логічних операцій. Завдяки цим особливостям, є можливість в одному біті інформації кодувати велику кількість станів, ефективно та паралельно маніпулювати. Результатом особливостей є те, що потужність квантових комп'ютерів зростає експоненційно[4]. Основні переваги квантових обчислень над класичними полягають у мініатюризації базового елемента обчислювального процесу та теоретичної переваги моделі. Що дозволяє ефективніше вирішувати завдання моделювання хімічних елементів, машинному навчанні, кібербезпеці. У цього типу обчислень є принципові недоліки: доступ до результатів є обмеженим, що робить неможливим спостереження проміжних результатів і продовження обчислень. На даний час вже існують комерційні квантові комп'ютери від IBM, Google та D-Wave, але вони мають свої обмеження через недосконалість технологій чи створюються для однієї задачі.

Квантові обчислення – це сфера, яка швидко розвивається і, в недалекому майбутньому, зробить прорив у багатьох областях. Квантові обчислення використовують принципи квантової механіки для обробки інформації і мають

потенціал для виконання конкретних завдань набагато швидше, ніж класичні обчислення. Основою квантових обчислень є квантовий біт (кубіт). На відміну від класичного комп'ютерного біта, якому призначається або 0, або 1, кубіту можуть бути призначені стани, які є суперпозицією 0 і 1. У порівнянні з класичними алгоритмами, квантові алгоритми мають потенціал для вирішення конкретних задач з експоненційним прискоренням. За останні два десятиліття з'явилася велика кількість літератури, яка сприяла розробці квантових алгоритмів.

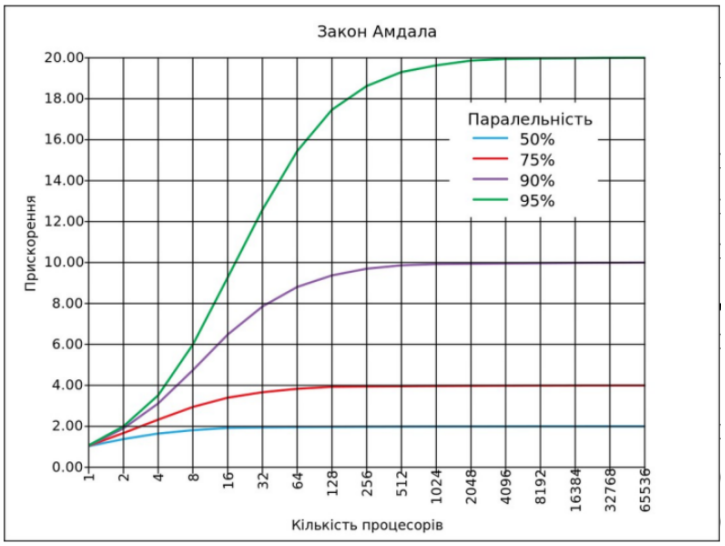


Рис. 1.2. Ілюстрація закону Амдала[3]

Так само, як і класичні обчислення, квантові обчислення застосовні до додатків у багатьох дисциплінах і будуть мати широкий вплив. Існують два типи додатків, де очікується, що квантові комп'ютери будуть перевершувати класичні комп'ютери. Перше застосування — це проблеми, які вимагають великої кількості паралельних обчислень, таких як оптимізація, шифрування, аналіз великих даних та машинне навчання. Іншим застосуванням є проблеми, які вимагають ефективного та точного моделювання квантових проблем у природі з таких областей, як фізика, хімія та матеріалознавство.

З огляду на швидкий розвиток квантового апаратного забезпечення, а також доступність універсальних квантових пристроїв для дослідників і професіоналів через Quantum-as-a-Service (QaaS), настав час зосередити увагу на розробці квантових програмних систем, щоб скористатися перевагами квантових технологій.

Між тим, різні сфери застосування квантових обчислень терміново потребують методологій та методів квантової розробки програмного забезпечення для підтримки розв'язання конкретних проблем. Тому існує нагальна потреба у створенні спільноти для квантової інженерії програмного забезпечення, яка б зосереджувалася на розробці методів, інструментів і процесів для ефективної розробки квантових програмних систем.

Існує ряд підходів до квантового програмування, наприклад, Scaffold, Qiskit, Q#, ProjectQ та Quipper. Крім того, концепції також застосовуються на ранніх етапах розробки квантового програмного забезпечення. Наприклад, на етапі проектування квантового програмного забезпечення забезпечує засоби для моделювання та специфікації квантових програмних систем, а вивчає питання модульності при проектуванні квантових систем. Завдяки різноманітним методам, доступним програмному інженеру на кожному етапі, завдання розробки квантової програмної системи створює значні проблеми. На кожному етапі розробки він повинен використовувати найбільш підходящу техніку квантового програмного забезпечення для програми, що розробляється. Вибір техніки може бути продиктований різними факторами, включаючи системні вимоги, організаційні методи та обмеження, накладені інструментами чи середовищем розробки, а також характером квантового програмування (механіки). Це означає, що на кожному етапі можна використовувати кілька методів у поєднанні одним з одним. Ускладнення методів розробки квантового програмного забезпечення потребує керівних принципів, які б підтримували розробку добре сконструйованих квантових програмних систем.

З ростом інтересу до квантових обчислень з'являються все більше бібліотек і інструментів для розробки в цій області. Існують середовища розробки і симулятори майже на всіх основних мовах програмування (Python, C/C++, Java і тд.).

Через фізичні обмеження поточної технології стан кубітів (квантових бітів) може залишатися стабільним лише короткий час. Протягом часу кубіти спонтанно змінюють свій стан, відповідно, алгоритми, запущені на квантовому комп’ютері, повинні будуть закінчитися за короткий час. Це спонукало дослідження спрощення квантових алгоритмів та їх представлення у вигляді квантових схем.

Для спрощення квантових схем Івама та Ямашіта [1] були надані правила перетворення для зниження вартості квантових схем, які складаються з n-кубітних вентилів Тоффолі. Їхнє рішення призначало вартість кожному вентилю та перетворювало схему на еквівалентну схему, загальна вартість якої була меншою. Це мотивовано тим, що деякі вентилі легше реалізувати у фізичній системі, і не націлені на глибину обчислень алгоритмів.

У дослідженні глибини квантового контуру Клів і Ватроус [2] показали, що обчислення приблизного значення QFT можна розпаралелювати до квантової схеми з послідовними кроками $O(log\, n)$, де n — кількість кубітів. Це дослідження було зосереджено на одному конкретному алгоритмі, а не на загальних квантових схемах.

Для оптимізації вільних квантових схем Маслов, Дуек, Міллер і Негревернь [3] запропонували та реалізували метод, який використовує локальну оптимізацію. Їхній метод використовує визначені шаблони для заміни неоптимальних конструкцій у схемах на більш оптимальні. Під час тестів їх основним завданням було зменшити кількість вентилів, а потім і глибину квантового контуру. Вони використовували свій алгоритм до квантових схем, відомих з літератури, і показали, що їхній метод був здатний зменшити кількість і глибину квантових схем. Недоліком цього методу є те, що він вимагає певних шаблонів, які призначені для оптимізації, відповідно, результат залежать від цих шаблонів.

Нещодавно Бродбентом і Кашефі була представилена нова технологія [4], яка використовує модель MBQC. Вона заснована на алгебраїчній структурі, розробленій в [5]. Використовуючи цю структуру, вони представили методику зменшення глибини квантового контуру. Квантові схеми перекладено на модель MBQC і застосовано ряд оптимізуючих перетворень. Після оптимізації модель MBQC повертається до квантової схеми, що призводить до створення нової квантової схеми, еквівалентної першій.

1.2. Авторизація

Авторизація — це процес, який перевіряє, чи має певний користувач доступ до певного ресурсу, або чи дозволено йому виконувати якусь операцію (наприклад,

видалити чи змінити певний ресурс). Авторизацію не слід плутати з контролем доступу, який є процесом застосування визначених правил безпеки для певного ресурсу.

Нова технологія під назвою “блокчейн” привертає нашу увагу з огляду на її властивості безпеки. Блокчейн вирішує проблему консенсусу розподілених баз даних і робить їх захищеними від несанкціонованого доступу за допомогою простих, але ефективних криптографічних алгоритмів. Ця дипломна робота зосереджена на дослідженні потенціалу платформи Ethereum, технології блокчейн, як основи рішення для контролю доступу, використовуючи переваги механізмів безпеки блокчейну та потенціалу смарт-контрактів.

1.2.1. Традиційна політика авторизації

Дискреційний контроль доступу (DAC) заснований на особистості користувача та наборі дозволів, що визначають, як користувачеві дозволено використовувати набір ресурсів. Політики DAC можуть бути представлені моделлю матриці контролю доступу (ACM), яка містить весь набір авторизацій, які є дозволами користувачам на використання ресурсів у системі. У матриці доступу A стовпці — це ресурси, рядки — користувачі, а записи в A[u, r] представляють права користувача u на ресурс r. Хоча ця модель проста, її впровадження стає непрактичним, з огляду на потенційно велику кількість порожнього простору. Таким чином, було запропоновано декілька варіацій моделі ACM: таблиця авторизації, списки контролю доступу (ACL) та модель на основі можливостей. У моделі таблиці авторизації таблиця має три стовпці, що представляють користувачів, дії та ресурси. Кожен рядок представляє один дозвіл, і для збереження місця фіксуються лише непорожні рядки. У моделі ACL таблиця зберігається для кожного ресурсу, кожен стовпець представляє користувача, а кожен рядок — дозволену дію. На відміну від цього, в моделі, що базується на можливостях, таблиця зберігається для кожного користувача, а стовпці представляють ресурси[17]. Пізніше буде детальніше розглянуто модель на основі можливостей, враховуючи її актуальність.

Контроль доступу на основі ролей (RBAC) заснований на принципі визначення ролей в організації або системі. Роль пов’язана з набором авторизацій, необхідних для виконання роботи. Оскільки багато користувачів, ймовірно, будуть мати однакові або подібні роботи, користувачам призначаються ролі замість окремих авторизацій. У цій схемі ролі користувачів важливіші за ідентичність [17].

Контроль доступу на основі атрибутів (ABAC) використовує атрибути для опису об’єктів, які беруть участь у прийнятті рішення про авторизацію, а також політики, щоб визначити, як надається доступ на основі атрибутів. Залученими об’єктами можуть бути користувачі, дії, ресурси та середовище, в якому виконується запит авторизації. Атрибути зазвичай виражаються у вигляді пар ключ-значення, напр. «відділ» користувача = «маркетинг», «тип» ресурсу = «загальнодоступний» або «дія» = «читати». Політикою (або правилом) може бути «користувачі з відділу маркетингу можуть читати публічні документи». У ABAC атрибути та політики можна перекладати за допомогою розширюваної мови розмітки керування доступом (XACML). Потужною особливістю ABAC є можливість адаптувати політики, які встановлюють відносини між об’єктами, напр. «співробітники відділу маркетингу можуть використовувати транспортні засоби, які належать відділу маркетингу», або, виражене псевдокодом, «надати доступ, якщо employee.deparment=vehicle.department». ABAC пропонує певну архітектуру для реалізації моделі. Як мінімум, архітектура визначає використання точки виконання політики (PEP) і точки прийняття рішень (PDP). PEP керує набором ресурсів, отримує запити на доступ і забезпечує виконання рішень щодо авторизації. PDP отримує запити від PEP, приймає рішення щодо авторизації та повертає результати PEP.

Керування доступом на основі можливостей (CapBAC), похідне від Discretionary Access Control, визначає можливість як пару *(r, p)*, де *p* — набір дозволів користувача діяти на ресурсі r. Доступ надається будь-якому користувачеві, який представляє цю можливість, і право власності на цю можливість має бути продемонстровано іншими способами. Реалізація моделі CapBAC зазвичай

використовує токен, що представляє набір можливостей користувача. Цей токен представляється користувачем, коли він запитує доступ до ресурсу [18].

1.3. Постановка задачі

Створити програмний продукт, який би використовував метод, описаний у [4], для розпаралелювання квантових схем. Таким чином, потрібно перевести квантові схеми в MBQC, оптимізувати їх у моделі MBQC і перевести їх назад до моделі квантової схеми. На додаток до оптимізацій в моделі MBQC, необхідно застосувати оптимізації до квантової схеми, яку ми отримуємо з моделі MBQC. Також для ефективної авторизації CapBAC студентів на основі блокчейну Ethereum для підвищення ефективності, захищеності та стійкого доступу до квантових обчислень.

1.4 Висновки до розділу 1

Отже, в класичних алгоритмах та способах збереження інформації, які вирішують завдання зі складністю “NP”, потребують великий розмір пам'яті та часу, є зараз використовуване рішення - розпаралелювання обчислень, але час не зменшується за законом Амдала. Більше переваг має квантове обчислення, оскільки фізичні можливості квантових апаратних забезпечень мають обмеження в часі, тому алгоритми робили простішими та більш швидкими.

Для авторизації користувачів обрано блокчейн, оскільки він вирішує проблему консенсусу розподілених баз даних і робить їх захищеними від несанкціонованого доступу за допомогою простих, але ефективних криптографічних алгоритмів.

РОЗДІЛ 2

ДОСЛІДЖЕННЯ

У цьому розділі наведено короткий вступ до квантових обчислень і двох моделей, які ми використовуємо в цій роботі: моделі квантової схеми та моделі MBQC. Поглиблений опис квантових обчислень та моделі квантової схеми можна знайти в підручниках [6, 7]. Аспекти MBQC, які мають відношення до цієї роботи, можна знайти в [4, 5].

2.1. Квантові обчислення

Аналогом класичного біта в квантових обчисленнях є кубіт. Кубіт відповідає двовимірній квантово-механічній системі. Стан кубіта може бути представлений одиничним вектором у двовимірному комплексному Гільбертовому просторі *H*. Ми можемо вибрати ортонормований базис у цьому просторі та позначити базисні вектори як $|0\rangle$ та $|1\rangle$. Тоді загальний стан кубіта такий:

$$\alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

де α і β — комплексні коефіцієнти та $|\alpha|^2 + |\beta|^2 = 1$. Базис $\{|0\rangle, |1\rangle\}$ для стану кубіта називається обчислювальним базисом. Стан кубіта змінюється шляхом застосування до нього унітарних операторів.

Стан *n*-кубіту — це одиничний вектор у просторі *n*-кратного тензорного добутку $H_1 \otimes H_2 \otimes \cdots \otimes H_n$. 2^n базисних станів цього простору є *n*-кратними тензорними добутками станів $|0\rangle$ та $|1\rangle$. Запишемо $|0\rangle \otimes \cdots \otimes |0\rangle$ як $|0 \cdots 0\rangle$. З цими базовими станами *n*-кубітний стан $|\phi\rangle$ є 2^n-вимірним комплексним одиничним вектором

$$|\phi\rangle = \sum_{i \in \{0,1\}^n} \alpha_i |i\rangle \tag{2.2}$$

2.2. Модель квантової схеми

У моделі квантової схеми, квантові обчислення представлені у вигляді квантових схем. Кубіти представлені у вигляді горизонтальних проводів, унітарні оператори представлені у вигляді вентилів, що діють на ряд проводів. Загальну квантову схему можна побачити на рис. 2.1. Обчислення, представлені квантовими схемами, виконуються шляхом застосування квантових вентилів зліва направо, поки не будуть застосовані всі вентилі.

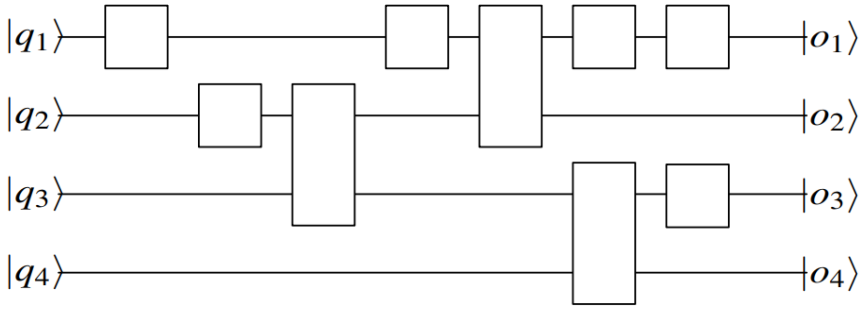


Рис. 2.1. Загальна квантова схема. Кожна горизонтальна лінія позначає «провід». Кожен провід представляє кубіт у обчисленні, виконаному квантовою схемою. $|q_1\rangle$, $|q_2\rangle$, $|q_3\rangle$ та $|q_4\rangle$ – початкові стани кубітів, представлених проводами. $|o_1\rangle$, $|o_2\rangle$, $|o_3\rangle$, $|o_4\rangle$ – вихідні стани цих кубітів. Прямокутники на проводах являють собою квантові вентилі. Вентилі можна застосовувати до будь-якої кількості проводів.

За нашим припущенням, застосування вентиля до кубітів у схемі займає один дискретний крок у часі, і що вентилі, що діють на різні кубіти, можна застосовувати паралельно. Тоді можемо розділити квантові схеми на декілька горизонтальних зрізів, так що виконання кожного кроку займає рівно один часовий крок. Це показано на рис. 2.2. Кількість таких зрізів у ланцюзі є глибиною цього квантового контуру. Також можемо оцінити глибину однорідного сімейства квантових схем, як функцію кількості вхідних кубітів. Наприклад, якщо ми говоримо, що квантовий контур має логарифмічну глибину, то ми маємо на увазі, що кількість дискретних

кроків часу, необхідних для оцінки схеми, збільшується логарифмічно відповідно до розміру проблеми, і ми пишемо, що глибина ланцюга $O(log\ n)$.

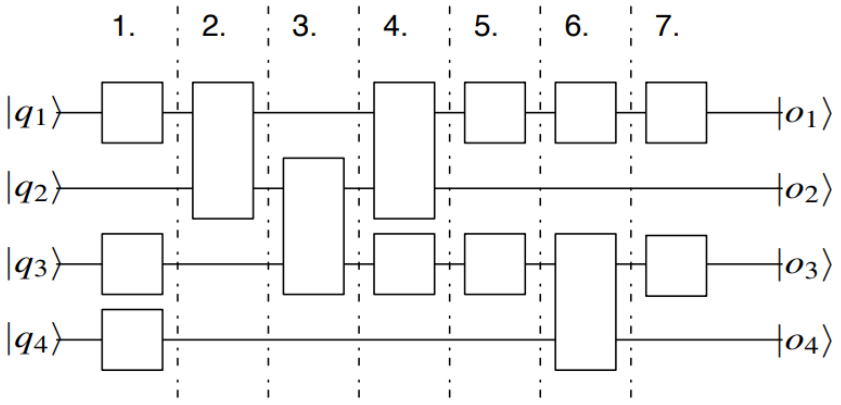


Рис. 2.2. Розрахунок глибини квантового контуру. Глибина цього конкретного контуру становить 7

Деякі з найбільш поширених однокубітних та двокубітних квантових вентилів представлені в додатку А. Усі ветилі, використані в цій роботі, або представлені на цьому малюнку, або представлені безпосередньо перед їх використанням. Будь-які обчислення, реалізовані за допомогою квантових схем, можуть бути реалізовані за допомогою лише невеликого набору різних вентилів. Такий набір ми називаємо універсальним набором вентилів.

Визначення 2.2.1. Набір вентилів називається універсальним, якщо для будь-якого цілого числа $n \geq 1$ будь-який *n*-кубітний унітарний оператор може бути виражений квантовою схемою, використовуючи лише скінченну кількість вентилів із цього набору.

Методи, які використовуються в цій роботі, вимагають, щоб усі квантові схеми були записані з використанням лише елементів $J(\alpha)$ і $\wedge Z$. Оскільки ці вентилі утворюють універсальний набір квантових вентилів [8], кожна квантова схема може бути реалізована за допомогою лише цих двох вентилів.

2.2.1. QFT

У цій роботі буде використано алгоритм QFT як приклад, тому визначимо його в цьому розділі. QFT є квантовим аналогом алгоритму дискретного перетворення

Фур'є (DFT). Для заданого розміру n DFT є лінійною функцією, яка відображає вектор $(a_0, a_1, ..., a_{n-1})$ та $(b_0, b_1, ..., b_{n-1})$ в C^n, де

$$b_x = \sum_{y=0}^{n-1} \left(e^{\frac{2\pi i}{n}}\right)^{x \cdot y} a_y \tag{2.3}$$

QFT є унітарним оператором, який відображає квантовий стан $\sum_{x=o}^{n-1} \alpha_x |x\rangle$ у квантовий стан $\sum_{x=o}^{n-1} \beta_x |x\rangle$, де

$$\beta_x = \frac{1}{\sqrt{n}} \sum_{y=0}^{n-1} \left(e^{\frac{2\pi i}{n}}\right)^{x \cdot y} a_y \tag{2.4}$$

У цій роботі використовується QTF_m для позначення алгоритму QTF на m кубітах. У рівнянні 2.4, n — кількість ортогональних базових станів. Для m-кубітної квантової системи це число дорівнює 2^m, тому для QTF_m маємо $n = 2^m$ у рівнянні 2.4. квантова схема QTF_m показана на рис. 2.4. Оскільки в цій роботі нами допускаються лише схеми, що складаються з елементів $J(\alpha)$ і $\wedge Z$, ми також представляємо алгоритм QTF_2, який написаний з використанням лише цих двох вентилів на рис. 2.5.

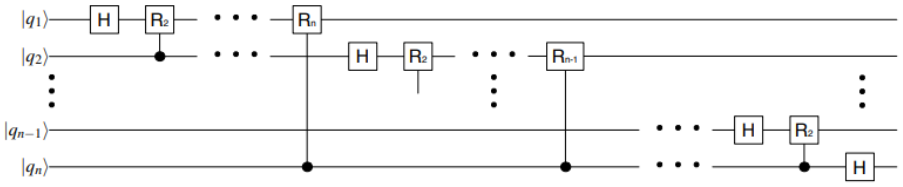


Рис. 2.3. Квантова схема n-кубітного QFT-алгоритму.

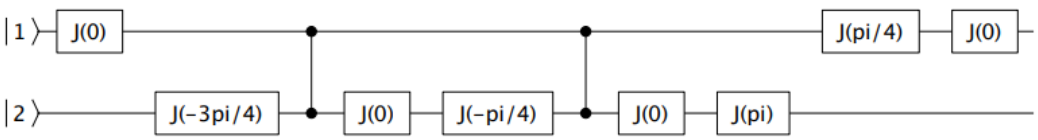


Рис. 2.4. Квантова схема для QTF_2, яка використовує лише вентилі $J(\alpha)$ і $\wedge Z$.

2.3. MBQC модель

Модель MBQC — це модель квантових обчислень, де квантовий стан змінюється шляхом виконання вимірювань для деяких кубітів і застосування поправок на основі результатів цих вимірювань. Детальний опис моделі MBQC доступний у [5]. Представляємо короткий виклад аспектів MBQC, які використовуються в даній роботі, і він базується на [5, 4]. У цій главі буде висвітлено, як обчислення в MBQC можуть бути представлені шаблонами вимірювання.

У MBQC вимірювання одного кубіта проводяться на попередньо підготовленому заплутаному квантовому стані. Результат цих вимірювань з одним кубітом є імовірнісним. Щоб зробити обчислення детермінованими, результати вимірювань використовуються для зміни вимірювань на деяких інших кубітах за допомогою локальних поправок.

2.3.1. Шаблон вимірювання

Шаблон MBQC визначається як $P = (V, I, O, A)$, де V - набір кубітів, $I \subset V$ - набір вхідних кубітів, $O \subset V$ — набір вихідних кубітів і A - кінцевий набір команд діючи на V. Шаблон записується, як послідовність команд, яких існує п'ять різних типів. Під час виконання обчислень команди застосовуються справа наліво, доки не будуть виконані всі команди. Таким чином, у цій роботі початком схеми є його правий кінець, а кінець схеми - його лівий кінець. Ми пояснюємо ці п’ять типів команд і описуємо кілька основних відносин між ними.

Команда підготовки N_i готує кубіт і у стан $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Команда підготовки застосовується до всіх невхідних кубітів. Пропускаємо ці команди з шаблону і припускаємо, що ця команда неявно застосовується до невхідних кубітів перед тим, як вони будуть використані.

Команда E_{ij} застосовує вентиль $\wedge Z$ до кубітів i та j. Оскільки $\wedge Z$ є симетричною операцією, маємо $E_{ij} = E_{ji}$. І оскільки $\wedge Z$ є його власною оберненою

операцією $E_{ij}E_{ji} = I$. Операція вимірювання M_{α}^{i} визначається як ортогональна проекція кубіта і на один із наступних станів: $|+\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\alpha}|1\rangle)$ та $|-\alpha\rangle = \frac{1}{\sqrt{2}}(|0\rangle - e^{i\alpha}|1\rangle)$ з подальшим частковим слідом системи над виміряним кубітом.

Вимірюються всі невихідні кубіти у шаблоні вимірювання. Класичний результат вимірювання на кубіті *i* позначимо як $s_i \in Z_2$. Де визначаємо $s_i = 0$, якщо результат вимірювання команди M_{α}^{i} був $|+\alpha\rangle$, і $s_i = 1$, якщо був $|-\alpha\rangle$. Результати вимірювань можна підсумувати за модулем 2, а суми результатів вимірювання називаються сигналами. Вимірювання може залежати від інших вимірювань за допомогою сигналів. Залежне вимірювання може залежати від двох сигналів, *t* і *s*:

$$^{t}[M_i^{\alpha}]^{s} = M_i^{(-1)^s\alpha + t\pi} \tag{2.5}$$

Для обчислення сигналів необхідно знати всі результати вимірювань, які з'являються в сигналах *t* і *s*. Це означає, що всі ці вимірювання необхідно виконати перед залежним вимірюванням.

Є дві однокубітні команди корекції Паулі: X_i^s та Z_i^s. Ці команди застосовують вентилі Паулі X і Z до кубіту *i*, якщо сигнал $s = 1$, і нічого не роблять, якщо $s = 0$: $X_i^1 = X_i$, $Z_i^1 = Z_i$, $X_i^0 = Z_i^0 = I_i$. Оператори корекції використовуються, щоб зробити обчислення детермінованим шляхом протидії небажаним результатам вимірювання. Результат вимірювання одного кубіта в MBQC є випадковим. Оператори корекції використовуються для зміни стану системи таким чином, щоб після вимірювання 1 це було так само, як якщо б результат був 0. Таким чином, хоча результати вимірювання є випадковими, обчислення залишаються детермінованими за допомогою поправок. Команди корекції відносяться до команди залежного вимірювання таким чином [5]:

$$ {}^{t}[M_i^{\alpha}]^s X_i^r =^{t} [M_i^{\alpha}]^{s+r} \tag{2.6} $$

$$ {}^{t}[M_i^{\alpha}]^s Z_i^r =^{t+r} [M_i^{\alpha}]^{s} \tag{2.7} $$

$$ M_i^{\alpha} X_i^s Z_i^t =^{t} [M_i^{\alpha}]^{s} \tag{2.8} $$

Було використано [[P]] для позначення квантових обчислень, реалізованих MBQC шаблоном P.

2.3.2. Від квантової схеми до моделі MBQC

Ми можемо створити шаблон, який реалізує обчислення, яке дорівнює обчисленню в моделі квантової схеми, додавши послідовності команд, які дорівнюють вентилям, до шаблону. Шаблони MBQC, що реалізують вентилі $J(\alpha)$ і $\wedge Z$, є такими:[5]:

$$ J(\alpha) = X_2 M_1^{-\alpha} E_{1,2} \tag{2.9} $$

$$ \wedge Z_{1,2} = E_{1,2} \tag{2.10} $$

У рівнянні 2.7 кубіт 1 у шаблоні є вихідним кубітом, який відповідає проводу, на який діє вентиль $J(\alpha)$, а кубіт 2 — новий кубіт для шаблону.

Як приклад наведемо схему вимірювання для квантової схеми QTF_2, показану на рис. 2.4. Зразок отримують за допомогою правил з формул 2.7 і 2.8:

$$ \begin{aligned} QFT_2 = X_{10}^9 M_9^{-\frac{\pi}{4}} E_{9,10} X_{10}^2 M_2^0 E_{2,9} X_8^7 M_7^{-\pi} E_{7,8} X_8^7 M_7^{-\pi} E_{7,8} X_7^6 M_6^0 E_{6,7} \\ E_{2,6} X_6^5 M_5^{\frac{\pi}{4}} E_{5,6} X_5^4 M_4^0 E_{4,5} \\ E_{2,4} X_4^3 M_3^{\frac{3\pi}{4}} E_{3,4} X_2^1 M_1^0 E_{1,2} \end{aligned} \tag{2.11} $$

2.4 Висновки до розділу 2

Отже, модель квантової схеми представляється у вигляді квантових схем, кубіти в схемах представляються проводами, а унітарні оператори у вигляді

вентилів, приклади схем наведені у розділі вище. Обчислення виконуються шляхом застосування квантових вентилів зліва направо, поки не будуть застосовані всі вентилі.

Застосування вентиля до кубітів у схемі займає один дискретний крок у часі, вентилі, що діють на різні кубіти, можна застосовувати паралельно. Квантову схему розділено на декілька горизонтальних зрізів, так що виконання кожного кроку займає рівно один часовий крок. Кількість таких зрізів у ланцюзі є глибиною цього квантового контуру. Оцінка глибини однорідного сімейства квантових схем, рівна функції кількості вхідних кубітів. Будь-які обчислення, реалізовані за допомогою квантових схем, реалізовані за допомогою лише невеликого набору різних вентилів. Такий набір називається універсальним набором вентилів.

Алгоритм QFT розглянутий як приклад, і він є квантовим аналогом алгоритму дискретного перетворення Фур'є (DFT).

За основу береться метод MBQC. У даному методі вимірювання одного кубіта проводяться на попередньо підготовленому заплутаному квантовому стані. Результат цих вимірювань з одним кубітом є імовірнісним. Щоб зробити обчислення детермінованими, результати вимірювань використовуються для зміни вимірювань на деяких інших кубітах за допомогою локальних поправок.

Тому є можливість створити шаблон, який реалізує обчислення, яке дорівнює обчисленню в моделі квантової схеми, додавши послідовності команд, які дорівнюють вентилям, до шаблону.

РОЗДІЛ 3

МЕТОДИ

Процедура, яка була реалізована в програмному забезпеченні для розпаралелювання квантових ланцюгів, така:

- Переведіть квантову схему в модель MBQC.
- Виконайте такі оптимізації обчислень у моделі MBQC:
  - стандартизація;
  - зміщення сигналу;
  - спрощення Паулі.
- Переведіть обчислення з моделі MBQC назад до моделі квантової схеми.
- Оптимізації, для зменшити глибину блоків CNOT і $\wedge Z$ затвора з $O(n)$ до $O(log\ n)$.

Оптимізації, які ми застосовуємо до обчислень, визначені для шаблону MBQC. Є три різні методи, які ми використовуємо в моделі MBQC для оптимізації шаблону: стандартизація, зсув сигналу та спрощення Паулі. Ці оптимізації визначені для шаблонів MBQC, як набір правил перезапису, і їх поглиблений опис можна знайти в [4] і [5]. Будемо використовуємо граф для представлення обчислень шаблону MBQC у нашій програмі, і таким чином опишемо вплив цих оптимізацій на базовий граф. Фактичні оптимізації шаблону MBQC реалізуються як модифікації графа і об’єднуються в один алгоритм зі створенням графа MBQC.

Спочатку опишемо три оптимізації, які виконуються на шаблоні MBQC. Потім надамо пояснення, як збудовано оптимізований граф MBQC, покажемо, як перетворити його назад у квантовий контур та опишемо оптимізації цієї квантової схеми.

Оскільки під час процесу розпаралелювання, який виконує наша програма, є три різні квантові схеми, будемо використовувати наступні позначення, щоб розрізняти їх:

- оригінальна схема - неоптимізована квантова схема, яка вводиться як вхід до програмного забезпечення.

- оптимізована схема - квантовий контур, який ми отримуємо після перекладу графа MBQC в модель схеми.
- розпаралелізована схема – квантовий контур, який ми отримуємо після застосування методів розпаралелювання, описаних у цьому розділі, до оптимізованої схеми. Це результат нашої програми.

3.1. Оптимізації в моделі MBQC

3.1.1. Стандартизація

Шаблон називається стандартним, якщо всі операції заплутування знаходяться на початку шаблону, за ними йдуть усі операції вимірювання, а операції корекції — в кінці шаблону.

Для переміщення команди заплутування на початок шаблону ми маємо такі правила перезапису [5] :

$$E_{ij}X_i^s \Rightarrow X_i^s Z_i^s E_{ij} \tag{3.1}$$

$$E_{ij}Z_i^s \Rightarrow X_i^s E_{ij} \tag{3.2}$$

$$E_{ij}A_{\vec{k}} \Rightarrow A_{\vec{k}}E_{ij} \tag{3.3}$$

де A не є командою заплутування, а $\vec{k}$ представляє набір кубітів, на які діє A, і не містить i та j.

Наступні правила комутативності дозволяють нам перемістити команди корекції в кінець шаблону[5]:

$$A_{\vec{k}}X_i^s \Rightarrow X_i^s A_{\vec{k}} \tag{3.4}$$

$$A_{\vec{k}}Z_i^s \Rightarrow Z_i^s A_{\vec{k}} \tag{3.5}$$

де A є довільною командою, а $\vec{k}$ представляє кубіти, на які діє A, і не містить i. У [4] та [5] була додаткова вимога, щоб A не було командою корекції. Це було потрібно,

щоб отримати завершення для алгоритму стандартизації, але для нашого алгоритму, представленого в розділі 3.3, ми повинні дозволити правилам корекції змінюватися.

Зазначаємо, що корекція X_i і Z_i змінюється в шаблоні, навіть, якщо застосовано до одного і того ж кубіта:

$$X_i^S Z_i^t \Rightarrow Z_i^t X_i^S \tag{3.6}$$

Це має місце тому, що

$$XZ = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{3.7}$$

$$ZX = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{3.8}$$

Оператори *XZ* і *ZX* відрізняються лише глобальною фазою, а оскільки в квантових обчисленнях глобальна фаза однокубітного унітарного оператора фізично незначна, виконується правило комутації (3.6).

Рівняння (2.8) можна використовувати для об'єднання поправок *X* і *Z* на кубіті з його вимірюванням. Після об'єднання всіх поправок до команди вимірювання на її кубіті залишаються лише поправки на вихідних кубітах, які можна перемістити в кінець шаблону за допомогою комутативності команд корекції (3.4), (3.5).

Шаблон вимірювання *P = (O, A, V, I)* у стандартній формі можна записати так:

$$P = CME \tag{3.9}$$

де C — оператор, який виконує всі поправки X і Z, M — оператор, який виконує всі команди вимірювання, а E — операція, яка виконує всі команди заплутування. C, M і E також можуть бути порожніми. У стандартизованому шаблоні всі виправлення вносяться на вихідні кубіти.

Нашим завданням є написання стандартизованого шаблону MBQC так, щоб виправлення певного вихідного кубіта були останніми командами шаблону. Це буде використано в розділі 3.3, щоб допомогти нам визначити, як додати вентилі до шаблонів. Завдяки комутативності команд корекції (3.4), (3.5), ми можемо перемістити поправки на деякому вихідному кубіті *o* ліворуч від шаблону, а завдяки рівнянню (3.6) можемо записати їх так, щоб корекція Z_{o_i} з'являється перед корекцією X_{o_i}. Пишемо C(O), щоб позначити оператора, який виконує всі виправлення X і Z виправлення кубітів у множині O. Позначаємо виправлення для всіх вихідних кубітів, крім кубіта oi, як $C(O\backslash\{o_i\})$ і маємо наступна еквівалентність:

$$C(O) = X_{o_i}^{S_{o_i}} Z_{o_i}^{t_{o_i}} C(O\backslash\{o_i\}) \tag{3.11}$$

Таким чином, можемо написати стандартизований шаблон MBQC як:

$$P = CME = C(O)ME = X_{o_i}^{S_{o_i}} Z_{o_i}^{t_{o_i}} C(O\backslash\{o_i\})ME \tag{3.12}$$

Наведене вище рівняння виконується, навіть, якщо кубіт o_i не належить шаблону *P* або *P* є порожнім шаблоном, тобто $o_i \notin O$:

$$P = C(O)ME = I_{o_i} C(O)ME = X_{o_i}^{0} Z_{o_i}^{0} C(O\backslash\{o_i\})ME \tag{3.13}$$

Ми будемо використовувати цю форму для зручності при описі алгоритму, який переводить квантові схеми в обчислення MBQC.

3.1.2. Спрощення Паулі

В цьому розділі вимірювання – це вимірювання Паулі X, коли кут вимірювання дорівнює α = 0, і вимірювання Паулі Y, якщо кут вимірювання дорівнює $\alpha = \frac{\pi}{2}$. Спрощення Паулі дозволяє виконувати вимірювання Паулі X і Y на першому кроці

обчислення. При наявності X у шаблоні вимірювання Паулі, ми можемо видалити залежності X корекції з кубіта відповідно до рівняння (3.14). Коли вимірювання Паулі Y виконується на кубіті, залежності X корекції можна змінити на залежності Z корекції, відповідно до рівняння (3.15). Згодом залежності Z корекції можуть бути переміщені в кінець шаблону за допомогою процесу, який називається зсувом сигналу, який визначено в наступному розділі. Рівняння, що описують спрощення Паулі, такі [5]:

$$M_i^0 X_i^s = M_i^0 \tag{3.14}$$

$$M_i^{\frac{\pi}{2}} X_i^s = M_i^{\frac{\pi}{2}} Z_i^s \tag{3.15}$$

З рівняння (2.13) отримуємо:

$${}^t\left[M_i^0\right]^s = {}^t\left[M_i^0\right] \tag{3.16}$$

$${}^t\left[M_i^{\frac{\pi}{2}}\right]^s = {}^{s+t}\left[M_i^{\frac{\pi}{2}}\right] \tag{3.17}$$

Якщо всі можливі спрощення Паулі були виконані MBQC на шаблоні P, то не буде X-поправок для кубітів, кут вимірювання яких дорівнює $\alpha = 0$ або $\alpha = \frac{\pi}{2}$.

3.1.3. Зміщення сигналу

Можна перемістити всі поправки Z на виміряні кубіти до кінця шаблону. Цей процес називається зсувом сигналу, і правила перезапису, які використовуються для зсуву сигналу, є такими [5]:

$${}^t[M_i^\alpha]^s \Rightarrow S_i^t[M_i^\alpha]^s \tag{3.18}$$

$${}^t\left[M_j^\alpha\right]^s S_i^r \Rightarrow S_i^{rt[(r+s_i)/s_i]}\left[M_j^\alpha\right]^{s[(r+s_i)/s_i]} \tag{3.19}$$

$$X_j^s S_i^r \Rightarrow S_i^r X_j^{s[(r+s_i)/s_i]} \tag{3.20}$$

$$Z_j^s S_i^r \Rightarrow S_i^r Z_j^{s[(r+s_i)/s_i]} \tag{3.21}$$

де $s[(r + s_i)/s_i]$ означає заміну s_i на $r + s_i$ в сигналі s. S_i^r є командою зсуву сигналу і використовується для переміщення сигналу ліворуч від шаблону MBQC. У шаблоні, зі зміщенням сигналу, поправки Z застосовуються лише до вихідних кубітів.

3.2. Графічне представлення шаблону MBQC

Цей проект вимагає реалізації квантових обчислень у моделі MBQC. Зокрема, обчислення повинні бути представлені як шаблони MBQC, оскільки оптимізації, які ми повинні використовувати, визначені для шаблонів MBQC. Для представлення шаблонів використовується граф. Для наших цілей буде достатньо, щоб граф представляв лише стандартизовані моделі MBQC.

Визначення 3.2.1. Стандартизований граф шаблону MBQC являє собою кортеж G , де V — множина вершин у G. Кожна вершина являє собою кубіт, $I \subset V$ — множина вхідних вершин, $O \subset V$ — множина вихідних вершин, E — множина ребер у G, M — множина, що представляє кути вимірювання для вершин у V, S — це набір множин вершин, який представляє X залежностей корекції для кожної вершини $v \in V$, T являє собою набір множин вершин, який представляє залежності Z корекції для кожної вершини $v \in V$. Ми говоримо, що вершина v залежить від іншої вершини w, якщо для кубіта, представленого v, потрібно виконати корекцію X або Z, залежно від вимірювання, виконаного для кубіта, представленого w.

У цій роботі стандартизований граф шаблонів MBQC називається просто графом MBQC. Множини S і T визначають, від яких інших вершин залежить вершина $v \in V$. v можна виміряти, якщо виміряно кожну вершину в $S \cup T$. У моделі MBQC можна виконувати паралельно будь-яку кількість вимірювань, якщо виконано вимірювання, від яких вони залежать. Найперший обчислювальний крок, на якому можна виконати вимірювання, можна знайти за допомогою алгоритму l - *CalculateLayer*. *Layer* — це набір вершин у графі MBQC, де найперший крок, який вершина може бути виміряна, є однаковою для кожної вершини шару. Таким чином, алгоритм 1 рекурсивно знаходить шар однієї вершини, від якої залежить вершина, і

встановлює її шар на максимальне значення плюс один. Якщо вершина не залежить від інших вершин, її можна виміряти на першому кроці і, таким чином, належить до першого шару.

Симетрична різниця двох множин A і B записується як $A \Delta B$ і визначається як множина елементів, які знаходяться в A або B, але не в обох одночасно. Його можна записати, використовуючи операції об'єднання, перетину та різниці:

$$A \Delta B = (A \cup B) \backslash (A \cap B) \tag{3.22}$$

Ми говоримо, що графі MBQC G еквівалентний стандартизованому шаблону MBQC $P = (V, I, O, A)$, якщо наступні співвідношення між ними вірні:

- Набори вершин, вхідних і вихідних вершин G і P рівні:

$$V = V' \tag{3.23}$$

$$I = I' \tag{3.24}$$

$$O = O' \tag{3.25}$$

- Команди заплутування P відповідають ребрам в G:

$$E = E_{i_1 j_1}, E_{i_2 j_2}, \ldots, E_{i_k j_k} \Leftrightarrow E' = E'_{i_1 j_1} \Delta E'_{i_2 j_2} \Delta \ldots \Delta E'_{i_k j_k} \tag{3.26}$$

Команда заплутування є власною оберненою (рівняння (2.6)), і використання симетричної різниці неявно видалить ребра, які представлені парну кількість разів у графі, залишаючи не більше одного ребра між двома вершинами.

- Кути вимірювання P зберігаються в наборі M':

$$M_q^{\alpha} \Leftrightarrow M'_q = \alpha, \qquad \forall q \in V \backslash O \tag{3.27}$$

- Сигнал корекції X на кубіті q у шаблоні P представлений у вигляді симетричної різниці на графу G.

$$X_q^{s_1+s_2+\ldots+s_m} \Leftrightarrow S_q = s_1 \Delta s_2 \Delta \ldots \Delta s_m \tag{3.28}$$

- Сигнал корекції Z на кубіті q у шаблоні P представлений у вигляді симетричної різниці на графу G.

$$Z_q^{t_1+t_2+t_3+\ldots+t_m} \Leftrightarrow T_q = t_1 \Delta t_2 \Delta \ldots \Delta t_m, \qquad \forall q \in V \tag{3.29}$$

Рівняння (3.23) - (3.29) дозволяють нам безпосередньо створити граф MBQC із шаблону. Для створення шаблону з графа потрібна додаткова інформація, оскільки ці рівняння не дають порядку виконання для команд шаблону. Оскільки шаблон є стандартизованим шаблоном, команди заплутування повинні бути першими. Порядок інших команд можна отримати, упорядкувавши вершини графів за їхнім шаром у порядку зростання та використовуючи рівняння (3.27) - (3.29) для створення команд. Порядок команд для кубітів в одному шарі не має значення, оскільки ці вимірювання не залежать один від одного і, отже, комутують.

граф може представляти лише стандартизовані шаблони MBQC, оскільки в рівнянні (3.26) ми припускаємо, що всі команди заплутування переміщено на початок шаблону. Для нашої мети цього достатньо, як ми побачимо в наступному розділі.

3.3. Створення оптимізованого графа MBQC

Ми визначаємо оптимізований шаблон як стандартизований шаблон із зміщеним сигналом і замінений Паулі, а граф, що представляє оптимізований шаблон, як оптимізований граф. Зауважте, що порожній шаблон, тобто шаблон без команд, також тривіально є оптимізованим шаблоном.

Як приклад оптимізованого шаблону ми знову використовуємо QFT_2:

$$\begin{gathered} QFT_2 = X_{10}^{1+9} Z_{10}^{2+5} X_8^{2+5+7} Z_8^{4+6} \\ M_9^0 \left[M_7^{-\pi}\right]^{4+6} M_6^0 \left[M^{\frac{\pi}{4}}\right]^{1+4} M_4^0 M_3^{\frac{3\pi}{4}} \left[M_2^{\frac{-\pi}{4}}\right]^1 M_1^0 \\ E_{9,10} E_{2,9} E_{7,8} E_{6,7} E_{2,6} E_{5,6} E_{4,5} E_{2,4} E_{3,4} E_{1,2} \end{gathered} \tag{3.30}$$

Легко перевірити, що шаблон (3.30) є оптимізованим шаблоном. По-перше, всі команди заплутування знаходяться на початку, вимірювання в середині, а виправлення в кінці. Таким чином схема має стандартну форму. По-друге, не існує жодних спрощень Паулі, які можна виконати. Нарешті, до шаблону більше не можна застосовувати правила зсуву сигналу. Таким чином, шаблон (3.30) є оптимізованим шаблоном. граф MBQC, що відповідає цьому шаблону, представлений далі в розділі 3.3.4.

Визначення 3.3.1. Якщо існує квантовий контур, який реалізує такі ж обчислення, що і шаблон MBQC P, і якщо P_g є шаблоном MBQC квантового вентиля g, то ми говоримо, що шаблон P' отримано шляхом додавання елемента g до шаблону P, якщо $[[P']] = [[P_g P]]$.

Визначення 3.3.2. Якщо P є шаблоном MBQC оптимізованого графа G, а P′ є шаблоном MBQC оптимізованого графа G′ , то ми говоримо, що G′ отримано шляхом додавання елемента g до G, якщо P′ отримано шляхом додавання вентиля g до P.

Було розроблено ітераційний алгоритм для створення оптимізованого графа MBQC G з квантової схеми QC. Починаємо з порожнього графа і додаємо вентилі з схеми QC до графа G в тому порядку, в якому вони повинні застосовуватися в схемі. Після того, як усі вентилі з QC будуть додані до G, граф G реалізує квантове обчислення, яке еквівалентне обчисленню, здійсненому схемою QC.

Допускаємо лише елементи $J(\alpha)$ і $\wedge Z$ у нашій вхідній схемі, ці вентилі утворюють універсальний набір, і їх достатньо для створення всіх можливих квантових обчислень. Таким чином, достатньо вивчити додавання лише цих двох вентилів до шаблону. Цей метод дасть нам алгоритм для побудови оптимізованого шаблону, але його потрібно представити у вигляді графа, який нами використовується. Модифікації графа, які необхідні для побудови оптимізованого графа, можна легко прочитати зі змін у шаблоні після додавання елемента.

3.3.1. Додавання вентиля $\wedge Z$ до графа MBQC

Нехай P — довільний оптимізований шаблон MBQC, а QC — квантова схема, яка виконує обчислення, еквівалентне [[P]]. Нехай QC' – це квантовий контур, який отримують шляхом додавання вентиля $\wedge Z_{i,j}$ до кінця QC. Тоді нехай O — набір вихідних кубітів у P, oi вихідний кубіт у шаблоні P, що відповідає проводу i в QC', і o_j вихідний кубіт у шаблоні P, який відповідає проводу j у QC'.

Використовуючи рівняння (3.13), можемо записати шаблон P як:

$$P = X_{o_i}^{s_i} Z_{o_i}^{t_i} X_{o_j}^{s_j} Z_{o_j}^{t_j} C(O \backslash \{o_i, o_j\}) ME \tag{3.31}$$

Зауважте, що це справедливо, навіть, якщо шаблон P є порожнім шаблоном або спочатку не містить кубітів o_i або o_j. Ми можемо створити шаблон P', який виконує такі ж обчислення, що й QC', додавши команду заплутування Eo_io_j до кінця шаблону P (2.15):

$$P' = E_{o_i o_j} P = E_{o_i o_j} X_{o_i}^{s_i} Z_{o_i}^{t_i} X_{o_j}^{s_j} Z_{o_j}^{t_j} C(O \backslash \{o_i, o_j\}) ME \tag{3.32}$$

А за допомогою рівнянь (3.1), (3.2) і (3.3) команду заплутування $E_{oi\,oj}$ можна перемістити поруч із оператором заплутування E.

$$P' = X_{o_i}^{s_j} Z_{o_i}^{t_i + s_j} X_{o_j}^{s_j} Z_{o_j}^{t_j + si} C(O \backslash \{o_i, o_j\}) ME_{o_i o_j} E \tag{3.33}$$

Легко помітити, що якщо шаблон P оптимізований, то і шаблон P' в (3.33) також оптимізований. Ми не змінювали команди вимірювання, це означає, що якщо P було оптимізовано, P' є зміщеним сигналом, а Паулі спрощеним. Оскільки вимірювання знаходяться в середині, виправлення в кінці, а заплутування на початку, шаблон P' має стандартну форму і, таким чином, оптимізований.

Нехай G це граф MBQC, як визначено в розділі 3.2, який представляє обчислення, виконані за шаблоном P, а G' граф для шаблону P'. Оскільки програма

використовує графічне представлення для зберігання обчислень, нас цікавить, як потрібно змінити граф *G*, щоб отримати граф *G′*.

Спочатку ми повинні переконатися, що обидві вершини o_i і o_j знаходяться в графі. Тоді, як ми бачимо з (3.33), додаючи команду заплутування $E_{oi\,oj}$ до шаблону P, ми повинні змінити залежності Z корекції від вершин o_i та o_i. Сам оператор заплутування $E_{oi\,oj}$ представляється як ребро між o_i та o_i. Якщо рівняння (3.31) показує шаблон *P*, а (3.33) шаблон *P′*, то відповідно до рівнянь (3.23) - (3.29) граф *G* слід модифікувати таким чином, щоб отримати граф *G′*:

$$V' = V \cup \{o_i, o_j\} \tag{3.34}$$

$$I' = I \cup (\{o_i, o_j\} \backslash V) \tag{3.35}$$

$$O' = O \cup \{o_i, o_j\} \tag{3.36}$$

$$E' = E \Delta \left\{E_{o_i o_j}\right\} \tag{3.37}$$

$$T'_{o_i} = T_{o_i} \Delta S_{o_i} \tag{3.38}$$

$$T'_{o_j} = T_{o_j} \Delta S_{o_i} \tag{3.39}$$

Представлено лише ті елементи, які в G′ відрізняються від елементів G. Рівняння (3.34) гарантує, що вершини, що представляють кубіти o_i та o_j, знаходяться в графі G′, а рівняння (3.36) — у множині вихідних вершин. Рівняння (3.35) має ефект додавання oi та oj до множини вхідних вершин у G′, якщо вони раніше не належали до G. Рівняння (3.38) і (3.39) розглядають зміни залежностей корекції Z та рівняння (3.37) додає ребро до графа.

3.3.2. Додавання елемента J(α) до графа MBQC

Нехай P — довільний оптимізований шаблон MBQC, а QC — квантовий контур, який виконує обчислення, еквівалентні P. Нехай O — набір вихідних кубітів у P, oi вихідний кубіт у шаблоні P, який відповідає проводу i в QC, і QC′ а квантовий контур, який можна отримати, додавши елемент J(α), що діє на кубіт i, до кінця КК.

Ми дозволяємо P бути будь-яким оптимізованим шаблоном, навіть порожнім, і, таким чином, QC може бути порожнім контуром. Відповідно до рівняння (3.13), схему *P* можна записати так:

$$P = X_{o_i}^{s_{o_j}} Z_{o_i}^{t_{o_i}} C(O \backslash \{o_i\}) ME \tag{3.40}$$

граф MBQC, що відповідає вихідному шаблону P, можна намалювати, як показано на рис. 3.1. Оскільки модифікації, які виконуються, торкнуться лише однієї існуючої вершини, будемо зосереджуватись на цій вершині.

Можемо створити шаблон P′, який виконує такі ж обчислення, що й QC′, спочатку додавши новий кубіт ni до P, а потім додавши команди, що відповідають вентилю J(α) (2.14), до кінця шаблону P.

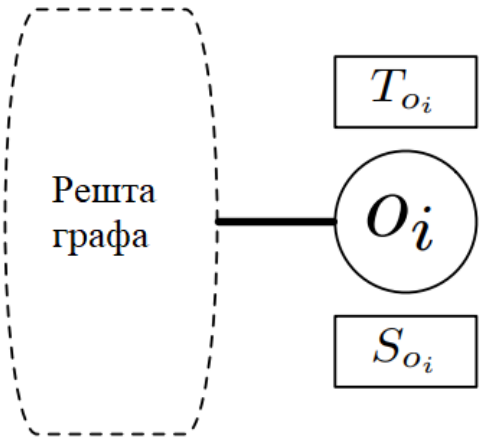


Рис. 3.1. Граф MBQC перед тим, як ми додамо вентиль J(α), який діє на провід i.

$$P' = X_{n_i}^{o_i} M_{o_i}^{\alpha} E_{n_i o_i} P = X_{n_i}^{o_i} M_{o_i}^{\alpha} E_{n_i o_i} X_{o_i}^{s_{o_i}} Z_{o_i}^{t_{o_i}} C(o \backslash \{o_i\}) ME \tag{3.41}$$

Як ми повинні змінити граф, що відповідає шаблону P, щоб отримати граф MBQC шаблону P′, показано на рис. 3.2.

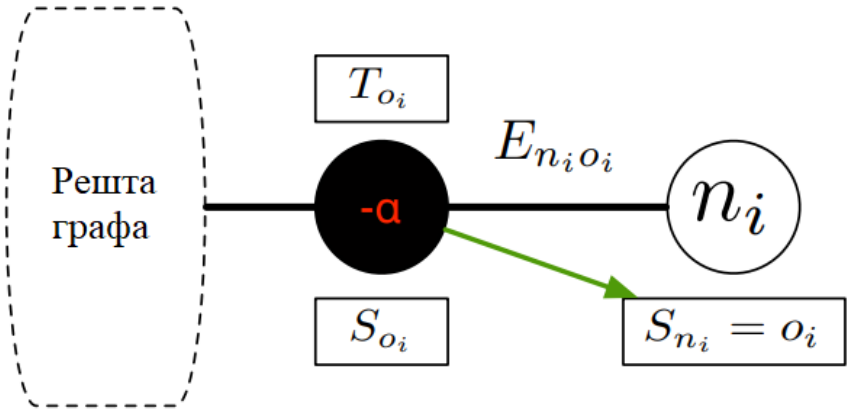


Рис. 3.2. Граф MBQC після того, як ми додамо вентиль J(α), який діє на провід i. Ми повинні додати нову вершину n_i і ребро між новою і старою вершиною. Вихідні вершини представляємо у вигляді білих кругів, а вимірювані вершини — у вигляді чорних кругів. Чорна вершина o_i і −α — кут вимірювання цієї вершини.

Рівняння (3.1), (3.2) і (3.3) можна використовувати для переміщення команди заплутування на початок шаблону:

$$P' = X_{n_i}^{o_i} M_{o_i}^{\alpha} X_{o_i}^{s_{o_i}} Z_{n_i}^{s_{o_i}} Z_{o_i}^{t_{o_i}} C(O \backslash \{o_i\}) M E_{n_i o_i} E \tag{3.42}$$

Як видно з рівняння (3.42), переміщення команди заплутування на початок шаблону введе Z поправок до вершини n_i. Це означає, що також потрібно змінити базовий граф, як показано на рис. 3.3.

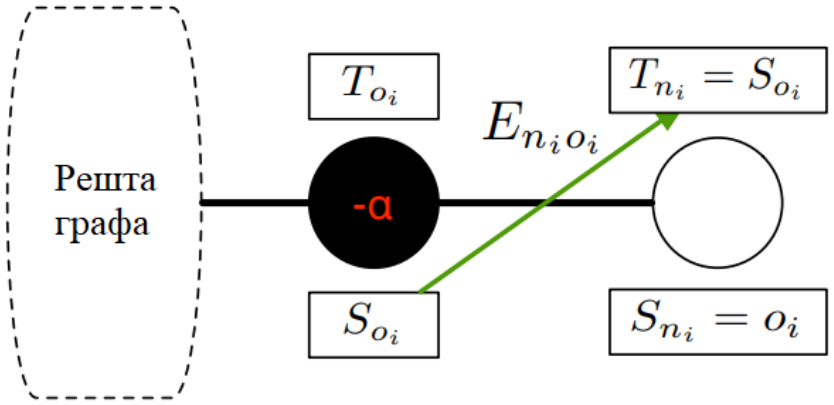


Рис. 3.3. Граф MBQC з доданим елементом J(α) після переміщення команди заплутування у формі шаблону на початок.

Тепер ми використовуємо комутативність корекції Z (рівняння (3.5)), щоб перемістити корекцію Z на кубіті ni до кінця шаблону, повз вимірювання на кубіті o_i, і рівняння (2.13) для об'єднання команд корекції на кубіті o_i з його команда вимірювання.

$$P' = X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}\ t_{o_i}} \left[M_{o_i}^{\alpha}\right]^{s_{o_i}} C(O\backslash\{o_i\}) M E_{n_i o_i} E \tag{3.43}$$

Далі виконуємо зсув сигналу за допомогою рівнянь (3.18) - (3.21). Зміни в базовому графі показані на рис. 3.4 і є похідними з наступного рівняння:

$$\begin{aligned} P' &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}} S_{o_i}^{t_{o_i}} \left[M_{o_i}^{\alpha}\right]^{s_{o_i}} C(O\backslash\{o_i\}) M E_{n_i o_i} E \\ &= S_{o_i}^{t_{o_i}} X_{n_i}^{t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} \left[M_{o_i}^{\alpha}\right]^{s_{o_i}} C(O\backslash\{o_i\}) M E_{n_i o_i} E \end{aligned} \tag{3.44}$$

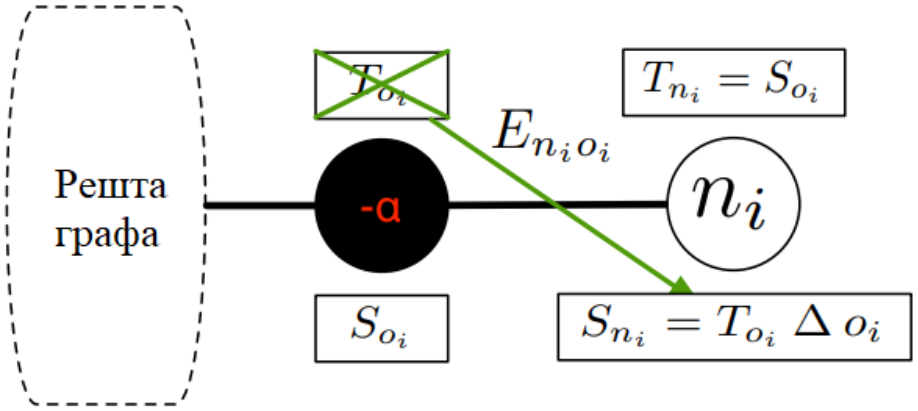


Рис. 3.4. Виконання зсуву сигналу на граф MBQC після додавання елемента J(α).

Додавання нового елемента до шаблону не внесе додаткові залежності через сигнали від кубіта o_i . Таким чином, після додавання іншого елемента $\wedge$Z або J(α), сигнал S_{oi}^{toi} завжди міг бути зміщений до кінця шаблону, без будь-якого впливу на інші команди шаблону. Тому ми опустимо S_{oi}^{toi} і перемістимо команду на кубіт oi

поруч з іншими командами вимірювання. Це можна зробити, оскільки жодна з поправок в C(O\{o_i}) не діє на кубіт o_i і, таким чином, можливе використання комутативності команд корекції (3.4) і (3.5).

$$P' = X_{n_i}^{t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} C(O\backslash\{o_i\}) \left[M_{o_i}^{\alpha}\right]^{s_{o_i}} ME_{n_i o_i} E \tag{3.45}$$

Якщо α = 0, ми маємо вимірювання Паулі X, і повинні використовувати правило спрощення Паулі X (3.14). Зробимо це після переміщення команди заплутування на початок шаблону в рівнянні (3.42). Таким чином, виконаємо таку послідовність перезаписів, якщо α = 0:

$$\begin{aligned} P' &= X_{n_i}^{o_i} M_{o_i}^{0} X_{o_i}^{s_{o_i}} Z_{n_i}^{s_{o_i}} Z_{o_i}^{t_{o_i}} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}\ t_{o_i}} \left[M_{o_i}^{0}\right] C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}} S_{o_i}^{t_{o_i}} M_{o_i}^{0} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= S_{o_i}^{t_{o_i}} X_{n_i}^{t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} M_{o_i}^{0} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} C(O\backslash\{o_i\}) ME_{n_i o_i} E \end{aligned} \tag{3.46}$$

Аналогічно, якщо $\alpha = \frac{\pi}{2}$, потрібно використовувати правило спрощення Паулі Z (3.17), внаслідок чого матимемо послідовність перезапису:

$$\begin{aligned} P' &= X_{n_i}^{o_i} M_{o_i}^{\frac{\pi}{2}} X_{o_i}^{s_{o_i}} Z_{n_i}^{s_{o_i}} Z_{o_i}^{t_{o_i}} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}} M_{o_i}^{\frac{\pi}{2}} Z_{o_i}^{s_{o_i}} Z_{o_i}^{t_{o_i}} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}\ s_{o_i}+t_{o_i}} \left[M_{o_i}^{\frac{\pi}{2}}\right] C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{o_i} Z_{n_i}^{s_{o_i}} S_{o_i}^{s_{o_i}+t_{o_i}} M_{o_i}^{\frac{\pi}{2}} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= S_{o_i}^{s_{o_i}+t_{o_i}} X_{n_i}^{s_{o_i}+t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} M_{o_i}^{\frac{\pi}{2}} C(O\backslash\{o_i\}) ME_{n_i o_i} E = \\ &= X_{n_i}^{s_{o_i}+t_{o_i}+o_i} Z_{n_i}^{s_{o_i}} C(O\backslash\{o_i\}) M_{o_i}^{\frac{\pi}{2}} ME_{n_i o_i} E \end{aligned} \tag{3.47}$$

Оскільки кубіт o_i тепер вимірюється, він більше не є вихідним кубітом; замість цього доданий кубіт n_i є новим вихідним кубітом, що відповідає дроту ланцюга і в QC'.

Легко перевірити, що, якщо початковий шаблон P був оптимізований, то кінцеві форми шаблону P′ в рівняннях (3.45), (3.46) і (3.47) дійсно є оптимізованими шаблонами. По-перше, усі вони мають команди корекції в кінці, вимірювання в середині та команди заплутування на початку, тому вони мають стандартну форму. По-друге, додане вимірювання не має t-сигналу, а оскільки P було оптимізовано, не мають також вимірювання в M. Таким чином, усі три шаблони є зміщеними сигналами. Отже, коли $\alpha = 0$ або $\alpha = \frac{\pi}{2}$ (шаблони (3.46) і (3.47)), нещодавно додані вимірювання матимуть сигнал s рівним нулю, а отже, якщо P було оптимізовано, шаблони Паулі (3.45), (3.46) та (3.47) є спрощеними Паулі. Це означає, що всі три можливі кінцеві форми шаблону P′ є оптимізованими шаблонами.

Нехай G є графом, що представляє шаблон *P*. Зміни шаблону *P*, необхідні для отримання *P′*, легко читаються з оптимізованих кінцевих форм рівнянь (3.45), (3.46) і (3.47). З цих змін ми можемо отримати модифікації для базового графа G, необхідні для отримання графа G′, який відповідає шаблону *P′*.

Потрібно переконатися, що вершини oi та ni знаходяться в графі, додати ребро між ними, встановити вимірювання для o_i та змінити множини S_{oi}, S_{ni}, T_{oi}, T_{ni} відповідно до сигналів корекції для o_i та n_i у шаблоні *P′*. Усі ці модифікації узагальнено в таблиці А.2. Алгоритм використовує ці правила модифікації та описує, як додати елемент *J(α)* до графа MBQC.

3.3.3. Алгоритм побудови графа MBQC

У попередніх розділах ми показали, як додати елементи $J(\alpha)$ і $\wedge Z$ до графа MBQC. Тепер ми можемо побудувати алгоритм для створення графа MBQC, що реалізує такі ж обчислення, що й квантовий контур. У наступних розділах ми проаналізуємо просторову та часову складність цього алгоритму.

Space Complexity.

Нехай QC — це квантовий контур з n проводами, k J(α) вентилів і l $\wedge Z$ вентилів, а G — граф MBQC, створений за допомогою алгоритму 4. Кількість вхідних вершин $|I|$ і вихідні вершини $|O|$ G дорівнює кількості проводів у QC,

оскільки вхідний і вихідний розміри обчислення, реалізовані *G*, повинні бути такими ж, як і в схемі *QC*. Кількість вершин дорівнюватиме *n+k*, оскільки нам потрібно додати виміряну вершину без вихідних даних для кожного елемента *J(α)*, і нам потрібно n вихідних кубітів, щоб зчитувати вихідні дані. Кількість ребер у графа MBQC буде максимум *k+l*, але може бути меншою, оскільки, додаючи вентиль, ми використовуємо симетричну різницю, щоб додати ребро до графа. Таким чином ми маємо:

$$|I| = n \tag{3.48}$$

$$|O| = n \tag{3.49}$$

$$|V| = n + k \tag{3.50}$$

$$|E| = O(k + l) \tag{3.51}$$

$$|M| = k \tag{3.52}$$

$$|S| = O(|V| - |I|) = O(k) \tag{3.53}$$

$$|T| = O(|O|) = O(n) \tag{3.54}$$

$$|S_q| = O(|M|) = O(k) \tag{3.55}$$

$$|T_q| = O(|M|) = O(k) \tag{3.56}$$

де *n* - кількість проводів у вхідному колі QC, *k* - кількість елементів *J(α)* у вхідному ланцюзі *QC*; *l* – кількість вентилів $\wedge Z$ у вхідному ланцюзі *QC*; |*I*| - кількість вхідних вершин у графі MBQC *G*; |*O*| - кількість вихідних вершин у графі MBQC *G*; |*V*| - кількість вершин у графі MBQC G; |*E*| - кількість ребер у графі MBQC *G*; |*M*| – кількість виміряних ребер у графі MBQC *G*; |*S*| - кількість вершин у графі MBQC *G*, які можуть мати *X* поправок (кожен не вхідний кубіт може мати поправку *X*); |*T*| це кількість вершин у графі MBQC *G*, які можуть мати *Z* виправлення (після зсуву сигналу лише вихідні кубіти матимуть *Z*-поправки); $\left|S_q\right|$ – максимальна кількість вершин, від результату вимірювання яких може залежати *X*-поправка вершини; $\left|T_q\right|$ – максимальна кількість вершин, від результату вимірювання яких може залежати *Z*-поправка вершини.

Знаючи розміри кожного елемента, який ми використовуємо для представлення нашого графа, ми можемо визначити загальний простір, необхідний для утримання цього графа.

$$\begin{aligned} &|V| + |I| + |O| + |E| + |M| + |S||S_i| + |T||T_i| = \\ &= (n+k) + n + n + O(k+l) + k + O(k)O(k) \ + O(n)O(k) = \\ &= O\big(n + k + l + k^2 + n \cdot k\big) = \\ &= O\big(l + n \cdot k + k^2\big) \end{aligned} \tag{3.57}$$

Не використовують жодних тимчасових змінних, тому загальна потреба в просторі алгоритму дорівнює розміру кінцевого графа MBQC, і тобто відповідно до рівняння 3.57 $O(l + n \cdot k + k^2)$.

Якщо глибина ланцюга QC дорівнює d, то в гіршому випадку кількість елементів J(α) k і $\wedge$Z вентилів l:

$$k = O(n \cdot d) \tag{3.58}$$

$$l = O(n \cdot d) \tag{3.59}$$

Це відповідає випадку, коли вентиль застосовується до кожного проводу на кожному кроці обчислення. Таким чином, ми виражаємо просторову складність нашого алгоритму відповідно до глибини схеми *QC*:

$$O\big(l + n \cdot k + k^2\big) = O\big(n \cdot d + n^2 \cdot d = n^2 \cdot d^2\big) = O\big(n^2 \cdot d^2\big) \tag{3.60}$$

Time Complexity.

Нехай QC — це квантовий контур з *n* проводами, *k J(α)* вентилів і *l* $\wedge Z$ вентилів, а *G* — граф MBQC, створений за допомогою алгоритму. Це описує ту саму ситуацію, що й попередній розділ, тому ми будемо використовувати ті самі позначення для цього розділу.

Нехай множини, використані в алгоритмі, є відсортованими множинами. Об'єднання, різницю та симетричну різницю двох відсортованих множин можна здійснити за $O(log_n)$ часу, якщо одна з множин має постійний розмір. Тут n — кількість елементів у наборі непостійних розмірів. Якщо жоден з двох наборів не має постійний розмір, то ці операції займають $O(min(n_1, n_2))$ часу, де n_1, n_2 — кількість елементів у двох наборах.

Спочатку потрібно перевірити, скільки часу потрібно, щоб додати $\wedge Z$ вентиль до графа. Створення ребра та вершин займає *O(l)* часу, усі операції набору матимуть один операнд із постійним розміром, отже, ці операції займають такий час:

Таким чином, загальний час виконання становить:

$$\begin{aligned} &O(log|V| + log|I|) + O(log|V|) + O(log|O|) + O(log|E|) = \\ &= O(log|V| + log|I| + log|O| + log|E|) \end{aligned} \quad (3.61)$$

Жоден з операндів не має постійного розміру, і найгірший час для обчислення цих операцій:

$$\begin{aligned} &O(min(|Tq|, |Sq|)) + O(min(|Tq|, |Sq|)) = \\ &= O(min(O(|M|), O(|M|))) + O(min(O(|M|), O(|M|))) = \\ &= O(|M|) \end{aligned} \quad (3.62)$$

Таким чином, загальний час виконання T_Z для додавання вентилів $\wedge Z$ до графа *G* становить:

$$Tz = O(log|V| + log|I| + log|O| + log|E| + |M|) \quad (3.63)$$

Додавши все це разом, отримаємо час виконання T_j для додавання вентилів *J(α)* до графа *G*:

$$TJ = O(log|V| + log|I| + log|O| + log|E| + |M|) \quad (3.64)$$

Це той самий час виконання, який був отриманий в рівнянні (3.63) для додавання вентилів $\wedge Z$ до графа. Оскільки ми маємо k $J(\alpha)$ і l $\wedge Z$ вентилів, загальний час виконання T створення графа G дорівнює:

$$\begin{aligned} T &= (k+l)\cdot O(log|V| + log|I| + log|O| + log|E| + |M|) = \\ &= (k+l)\cdot O(log(n+k) + logn + logn + log(k+l) + k) = \\ &= (k+l)\cdot O(log(n+k) + log(k+l) + k) = \\ &= (k+l)\cdot O(logn + logl + k) = \\ &= O(k\,log_n + k\,log_l + k^2 + l\,\log_n + l\,\log_l + kl) = \\ &= O(k\,\log_n + k^2 + l\,\log_n + l\,\log_l + kl) \end{aligned} \tag{3.65}$$

Тут було використане рівняння (3.48) - (3.52), щоб підставити розміри елементів у графі MBQC параметрами вихідної схеми.

Якщо глибина ланцюга QC дорівнює d, то можемо замінити кількість елементів $J(\alpha)$ k і $\wedge Z$ вентилів l відповідно до рівнянь (3.58) і (3.59):

$$\begin{aligned} T &= O(k\log_n + k^2 + l\,\log_n + l\,\log_l + kl) = \\ &= O(n\cdot d\cdot \log_n + n^2d^2 + n\cdot d\cdot \log_n + n\cdot d\cdot \log_{nd} + n^2\cdot d^2) = \\ &= O(n^2d^2) \end{aligned} \tag{3.66}$$

Це дає нам час виконання, яке залежить від кількості кубітів n і глибини вхідного контуру d як $O(n^2d^2)$.

Також можливо виразити час виконання, яке потрібно для побудови графа MBQC залежно від кількості вершин у кінцевому графі MBQC. У цьому випадку було б корисно припустити, що два $\wedge Z$ вентилі, які компенсують один одного, не додаються до графа. Їх можна було замінити на вентилі ідентифікації в оригінальній схемі, перш ніж почати будувати граф. Тоді кількість вентилів, доданих до графа, буде $|E|$, оскільки кожен вентиль буде вводити, але ніколи не видалятиме, ребро. Розмір елементів графа залежить від кількості вершин $|V|$ як:

$$|I| = O(|V|) \tag{3.67}$$
$$|O| = O(|V|) \tag{3.68}$$
$$|E| = O\left(|V|^2\right) \tag{3.69}$$
$$|M| = O(|V|) \tag{3.70}$$

Таким чином, час виконання алгоритму 4:

$$\begin{aligned} T &= |E| \cdot O(log|V| + log|I| + log|O| + log|E| + |M|) = \\ &= O(|V|^2) \cdot O(log|V| + log|V| + log|V| + log|V| + |V|) = \\ &= O(|V|^3) \end{aligned} \tag{3.71}$$

Отриманий результат досить цікавий, оскільки, використовуючи правила перезапису, на яких базувався наш алгоритм (описані в [4, 5]), оптимізація шаблону MBQC займає $O(n^5)$ часу. Ці правила перезапису є потужнішими, ніж наш алгоритм, оскільки вони можуть оптимізувати будь-який дійсний шаблон MBQC, розроблений алгоритм працює лише на шаблонах, які ми можемо отримати з квантових схем. Крім того, алгоритми не виконують однакові обчислення. Наш алгоритм приймає квантову схему як вхід і виводить граф MBQC, що відповідає оптимізованому шаблону. Алгоритм у [5] приймає шаблон MBQC як вхід і виводить оптимізований шаблон MBQC. Тим не менш, враховуючи шаблон MBQC, що реалізує квантову схему як вхід до алгоритму в [5], вихідний шаблон буде еквівалентним шаблону, представленому графом MBQC, записаним нашим алгоритмом.

3.3.4. Приклад графа MBQC.

Як приклад графа, який створює наш алгоритм, ми представляємо граф MBQC QFT_2. На малюнку 3.5 показані ребра і вершини графа, на рис. 3.6 представлені поправки X і Z.

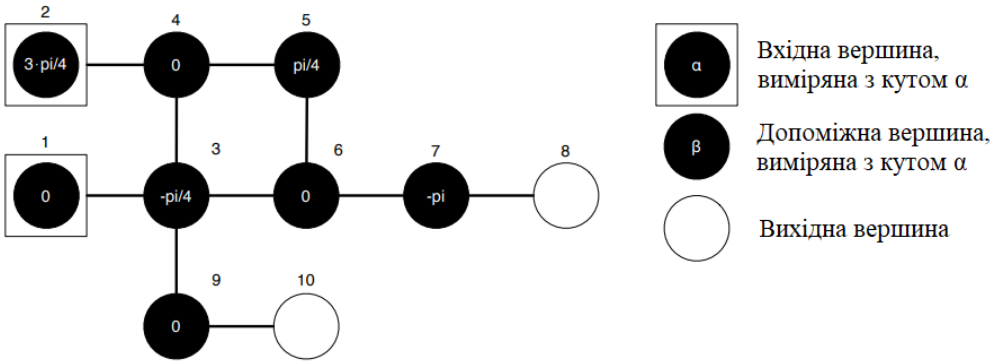


Рис. 3.5. Граф MBQC QFT_2, розробленим алгоритмом, без залежностей сигналу.

3.4. Від графа MBQC до квантової схеми.

Щоб перетворити граф MBQC назад у квантову схему, використали алгоритм, який перетворює шаблони MBQC в квантові схеми, описаний у [4], і

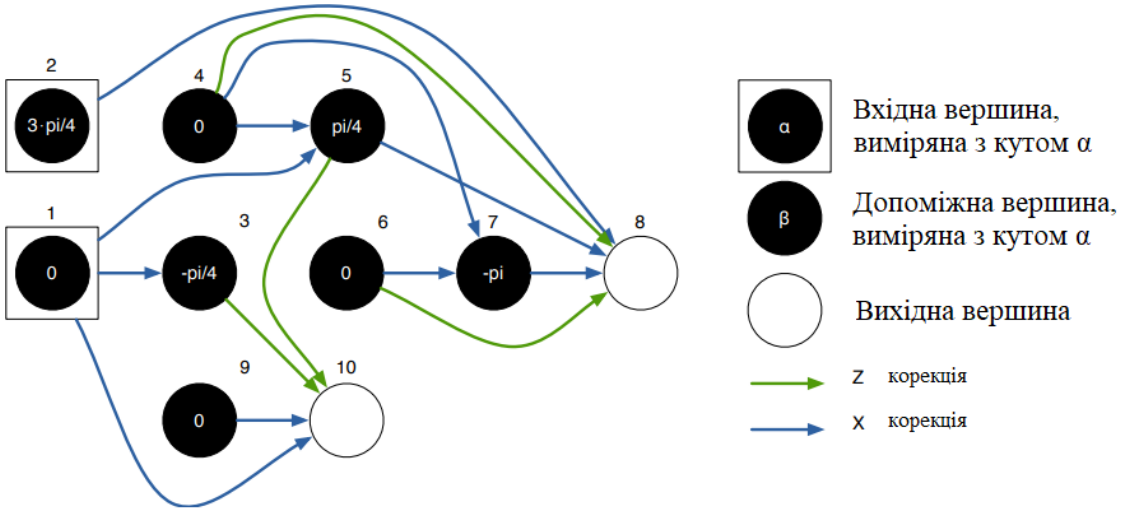


Рис. 3.6. Корекція X і Z графа MBQC QFT_2.

модифікували його, щоб він працював з нашим графом MBQC. Якщо є граф MBQC G і порожня схема *C*, то алгоритм перетворення графа в схему складається з нижче описаних кроків.

Як приклад цього алгоритму на рис. 3.7 показано схему, яка буде отримана, якщо переведемо граф MBQC QFT_2 назад до моделі квантової схеми. На рис. 3.7

бачимо структуру квантових схем, які ми отримуємо з графа MBQC. Зазначимо, що алгоритм 5 передбачає, що всі не вхідні кубіти мають початковий стан $|0\rangle$. Ми бачимо початкові вентилі Адамара (створені на кроці 2 алгоритму) і $\wedge Z$ вентилі (створені на кроці 3) на початку. Потім ми отримуємо деякі блоки вентилів J(α), переплетені блоками CNOT (створеними на кроці 4), і закінчуємо блоком вентилів $\wedge Z$ і CNOT (створеним на кроці 5) перед остаточними вимірюваннями (створеними на кроці 6).

3.5. Оптимізація квантової схеми

Схема, яку ми отримаємо після зворотного перекладу з моделі MBQC, матиме структуру, показану на рис. 3.8 (і на рис. 3.7). Для подальшого зменшення глибини цієї схеми ми можемо використовувати методи, описані Муром і Нільссоном у [9], щоб зменшити глибину підсхем, що складаються лише з вентилів $\wedge Z$ і CNOT, до логарифмічної глибини. Це можна зробити за допомогою наступних твердженнь.

Твердження 3.5.1 (твердження 5 з [9]). Контур будь-якого розміру на n кубітах, що складається з діагональних або взаємно комутуючих вентилів, кожен з яких поєднує не більше k кубітів, можна розпаралелювати на глибину $O(n^{k-1})2^{O(k)}$ без допоміжних елементів, а на глибину $O(k\,log_n) + 2^{O(k)}$ з $O(n^k)$ допоміжних елементів.

Твердження 3.5.2 (твердження 6 з [9].) Контур будь-якого розміру на n кубітах, що повністю складається з керованих-NOT вентилів, можна розпаралелювати на глибину $O(log_n)$ за допомогою $O(n^2)$ допоміжних елементів.

Нехай *QC* — це квантовий контур з n проводами, k *J(α)* вентилів, l $\wedge Z$ вентилів і глибиною d. Нехай *G* — граф MBQC, що реалізує такі ж обчислення, що й QC. З рівнянь (3.50), (3.58), (3.59) видно, що кількість вершин $|V|$ в *G* є:

$$|V| = k + l = O(n \cdot d) + O(n \cdot d) = O(n \cdot d) \tag{3.72}$$

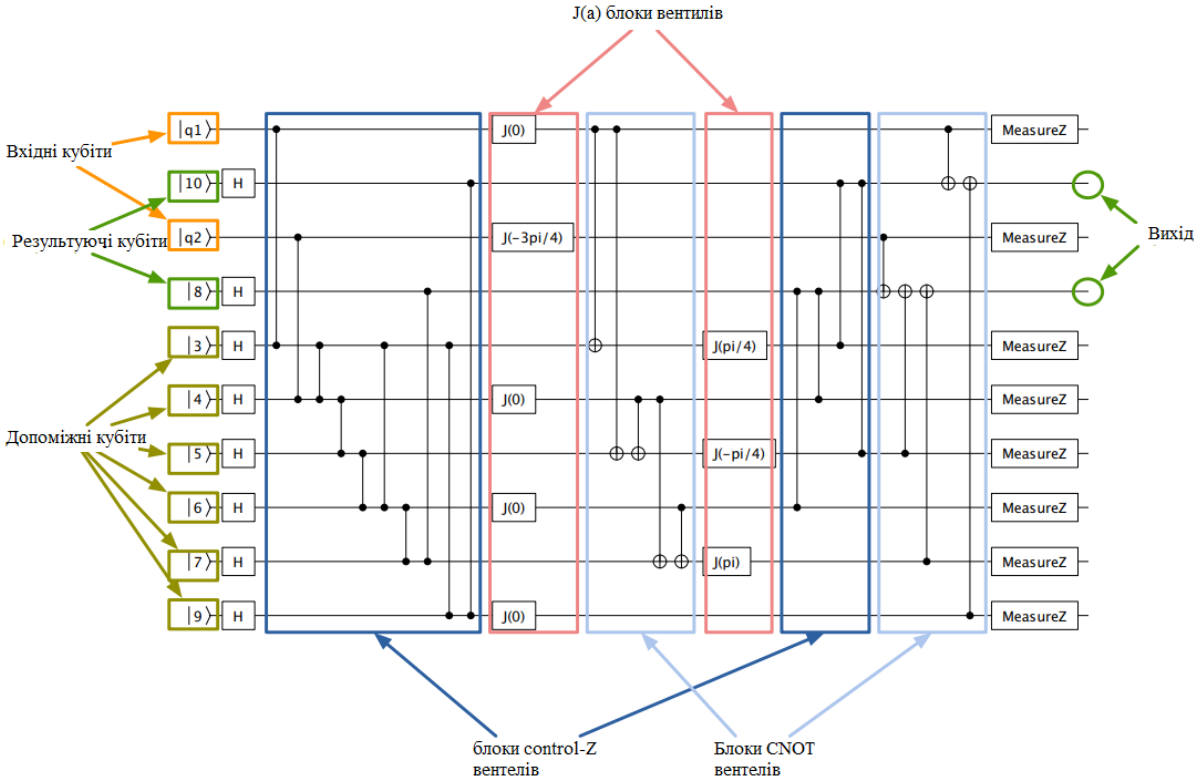


Рис. 3.7. Схема QFT_2 відразу після перекладу її назад із моделі MBQC

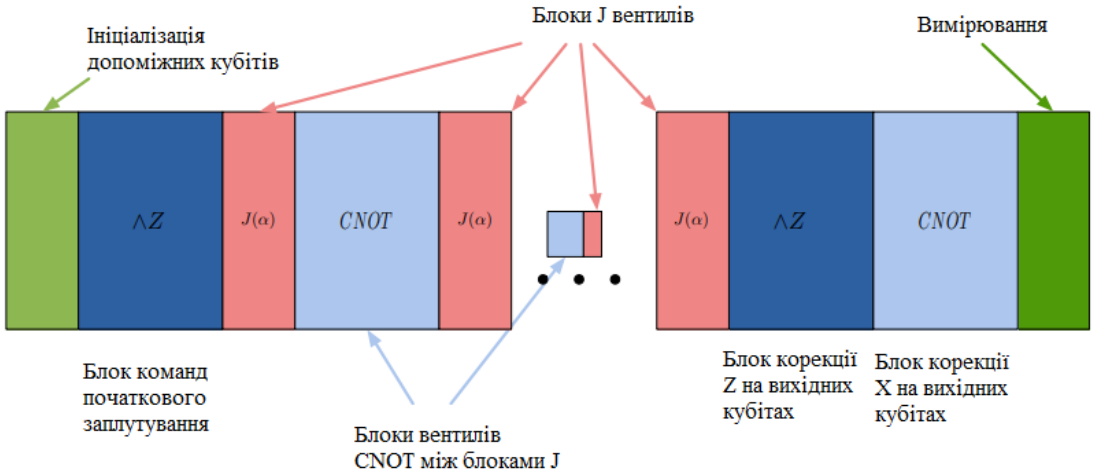


Рис. 3.8. Структура оптимізованих схем

Алгоритм переводить кожну з вершин G в провід оптимізованої схеми. Таким чином, оптимізована схема матиме $O(n \cdot d)$ кубітів. Використовуючи твердження 3.5.1, підсхеми $\wedge Z$ в оптимізованій схемі розпаралелюються до глибини $O(log_{nd}) = O(log_n + log_d)$. Крім того, підсхеми CNOT розпаралелюються на

однакову глибину $O(log_n + log_d)$ за допомогою твердження 3.5.2. Нехай d' — глибина обчислення MBQC. Бродбент і Кашефі довели в [4], що d' не буде більше $O(d)$. Оскільки кожен з блоків $J(\alpha)$ між підсхемами CNOT представляє крок у обчисленні MBQC, загальна глибина паралельного контуру після зменьшення глибини підсхем CNOT і $\wedge Z$, буде $O(d'log_n + d' log_d) = O(d log_n + d log_d)$.

Глибина $O(d log_n + d log_d)$ є лише верхньою межею розпараллеленої схеми. Існують схеми, які після розпаралелювання матимуть меншу глибину, ніж раніше. Деякі з цих ланцюгів описані в розділі 6 результатів. Оскільки глибина паралельних ланцюгів $O(d'log_n + d' log_d)$ залежить від глибини графів MBQC, якщо ми можемо зменшити глибину вихідної схеми в моделі MBQC, глибина паралельних контурів також зменшиться. Це причина, чому оптимізації в моделі MBQC можуть зменшити глибину квантових схем. Наприклад, якщо контур глибини O(n) можна оптимізувати до глибини $O(log_n)$ у моделі MBQC, то глибина паралельного контуру буде $O(log_n \cdot log_n + log_n \cdot loglog_n) = O(log_n^2)$.

3.6. Модель авторизації

Було обрано контроль доступу, на основі атрибутів (ABAC), як модель авторизації, оскільки варіант використання вимагає, щоб контроль доступу надавався на основі прав, а не ідентичності. Крім того, ABAC дозволяє визначати правила доступу з великим рівнем деталізації.

Щоб використовувати ABAC, визначимо атрибути для користувачів і ресурсів, а також політики, які записуються у функціях цих атрибутів. Можна визначити багато політик одночасно, і доступ надається за умови, що принаймні одна політика задовольняється:

$$GrantAccess(user, resource) : \bigvee_{j=1}^{k} Policy_j(user, resource) \qquad (3.73)$$

де *user* і *resource* — це кортежи, *k* — кількість політик, заданих у системі контролю доступу.

Наша архітектура реалізує ключові компоненти моделі ABAC (Контроль доступу на основі атрибутів), а саме, точку виконання політики (PEP) і точку прийняття рішення про політику (PDP). PEP відображається на ресурси, а PDP — на систему контролю доступу.

3.6.1. Атрибути

Визначаємо атрибути користувача та ресурсу як кортежі, де кожен елемент має значення типу даних Solidity(Типи даних Solidity). І будемо використовувати точкове позначення для представлення окремих елементів кортежів, наприклад *userAttrib.userId* — це елемент *userId* атрибутів р користувача *userAttrib*.

Атрибути користувача:

$$userAttrib = (userId, userName, userManagerId, rights) \quad (3.74)$$

де *userId* та *userName* – це bytes32, *userManagerId* – address.

Атрибути ресурсу:

$$resourceAttrib = (resourceId, objects) \quad (3.75)$$

де *resourceId* - це bytes32, *objects* - це масив bytes32.

Resource.objects можна інтерпретувати як сукупність ресурсів, наприклад підресурси уніфікованого ідентифікатора ресурсу (URI) у ресурсі *q*. Користувач виявляє підресурси, описані об’єктами, і включає в запит доступу елемент масиву, який представляє підресурс, до якого він хоче отримати доступ, як параметр об’єкта, як показано в тілі запиту.

Важливим вибором дизайну є те, де буде зберігатися інформація про атрибути та як вона поводить себе під час взаємодій. У нашій архітектурі інформація про атрибути зберігається в системі контролю доступу. Користувачам потрібно лише

вказати свою особу як частину запиту на доступ, а додаткові атрибути, необхідні для рішення про авторизацію, а саме *userAttrib.rights*, витягуються зі сховища даних у системі контролю доступу. У разі ресурсів інформація про атрибути зберігається не в системі контролю доступу, а в самих ресурсах.

3.6.2. Політики

Політики написані з точки зору атрибутів користувача та ресурсу. У реалізованій архітектурі ми використовуємо лише одну політику. Ця політика має значення true, якщо є відповідність між userAttribp.rights користувача p і запитаним об’єктом.

$$Policy(userAttribp.rights, object : \bigvee_{j=1}^{k} userAttribt_p.rights_j = object \quad (3.76)$$

де *k* — розмір *userAttribt.rights*, *object* — значення bytes32.

3.6.3. Смарт контракт

Функції, необхідні для задоволення варіанту використання, реалізовані в смарт-контракті. Будемо класифікувати їх, як критичні чи некритичні в залежності від того, чи змінює функція стан, напр. знімати токени з рахунку. Критичні функції мають бути викликані за допомогою підписаних транзакцій, тоді як некритичні функції можуть бути викликані за допомогою викликів.

Серед найважливіших функцій у нас є: реєстрація користувачів, зміна прав, видалення користувачів і реєстрація менеджерів користувачів. Серед некритичних функцій у нас є: перевірка прав, прав на перегляд і перегляд користувачів. Найактуальнішими функціями є наступні:

- RegisterUserManager: реєструє менеджера користувачів, даючи йому повноваження реєструвати користувачів і керувати їхніми правами.
    - Метод виклику: транзакція підписана адміністратором.
    - Параметри: address userManagerId, bytes32 userManagerName, uint balance, bytes32 pricelist.

  - Повертає: логічне значення; true, якщо userManagerId не був зареєстрований раніше; інакше повертає false.

- RegisterUser: реєструє нового користувача із зазначеним набором прав і віднімає комісію з балансу менеджера користувачів, що запитує.
  - Метод виклику: транзакція підписана менеджером користувачів, що запитує.
  - Параметри: address userManagerId, bytes32 userId, bytes32 userName, bytes32 права.
  - Повертає: логічне значення; true, якщо userManagerId зареєстровано і він відповідає підпису транзакції, userId не був зареєстрований раніше, а менеджер користувачів має достатній баланс маркерів, інакше повертає false.
- SetUserRights: встановлює права існуючого користувача та віднімає комісію з балансу менеджера користувачів, що запитує.
  - Метод виклику: транзакція підписана менеджером користувачів, що запитує. Параметри: address userManagerId, bytes32 userId, bytes32 права.
  - Повертає: логічне значення; true, якщо userM anagerId і userId зареєстровані, userM anagerId, пов’язаний з наявним користувачем, відповідає підпису транзакції, а менеджер користувачів має достатній баланс маркерів; інакше повертає false.
- DeleteUser: видаляє існуючого користувача, який належить менеджеру користувачів, що запитує.
  - Метод виклику: транзакція підписана менеджером користувачів, що запитує.
  - Параметри: address userManagerId, bytes32 userId.
  - Повертає: логічне значення; true, якщо userManagerId та userId зареєстровані, а userManagerId, пов’язаний із наявним користувачем, відповідає підпису транзакції; інакше повертає false.
- Перевірити: визначає, чи надається користувачеві доступ до ресурсу. Отримує права користувача з блокчейну та викликає політику, визначену в 5.4.

- Метод виклику: виклик. –
- Параметри: address userManagerId, bytes32 userId, bytes32 object. – Повертає: логічне значення; true, якщо політика, визначена в (3.76), задоволена, реєструються userManagerId та userId; інакше повертає false.

- GetUserRights: отримує список прав даного користувача.
  - Метод виклику: виклик.
  - Параметри: address userManagerId, bytes32 userId.
  - Повертає: масив bytes32; заповнено userAttribp.rights користувача p, якщо userManagerId і userId існують, а userAttribp.rights не порожній; інакше, один елемент масиву bytes32 з кодом помилки.
- GetUserList: отримує список користувачів певного менеджера користувачів.
  - Метод виклику: виклик.
  - Параметри: address userM anagerId.
  - Повертає: два масиви bytes32; якщо userManager існує, масиви заповнюються userAttrib.userId і userAttrib.userName всіх користувачів, що відповідають параметру userManagerId; в іншому випадку два масиви bytes32 з одним елементом з кодом помилки.

3.7. Висновки до розділу 3

Отже, є три різні методи, які ми використовуємо в моделі MBQC для оптимізації шаблону: стандартизація, зсув сигналу та спрощення Паулі. Ці оптимізації визначені для шаблонів MBQC, як набір правил перезапису. Шаблон називається стандартним, якщо всі операції заплутування знаходяться на початку шаблону, за ними йдуть усі операції вимірювання, а операції корекції — в кінці шаблону. Виправлення певного вихідного кубіта останніми командами шаблону. Це для того щоб допомогти нам визначити, як додати вентилі до шаблонів.

Вимірювання Паулі X, коли кут вимірювання дорівнює $\alpha = 0$, і вимірювання Паулі Y, якщо кут вимірювання дорівнює $\alpha = 2$. Спрощення Паулі дозволяє виконувати вимірювання Паулі X і Y на першому кроці обчислення. При наявності X у шаблоні вимірювання Паулі, ми можемо видалити залежності X корекції з

кубіта. Коли вимірювання Паулі Y виконується на кубіті, залежності X корекції можна змінити на залежності Z корекції. Згодом залежності Z корекції переміщені в кінець шаблону за допомогою процесу, який називається зсувом сигналу. У шаблоні, зі зміщенням сигналу, поправки Z застосовуються лише до вихідних кубітів.

Вентилі утворюють універсальний набір, і їх достатньо для створення всіх можливих квантових обчислень. Таким чином, достатньо вивчити додавання лише двох вентилів до шаблону. Цей метод дає нам алгоритм для побудови оптимізованого шаблону, але його потрібно представити у вигляді графа, який використовується. Модифікації графа, які необхідні для побудови оптимізованого графа, можна легко прочитати зі змін у шаблоні після додавання елемента. Щоб перетворити граф MBQC назад у квантову схему, використали алгоритм, який перетворює шаблони MBQC в квантові схеми.

Оскільки глибина паралельних ланцюгів залежить від глибини графів MBQC, якщо зменшити глибину вихідної схеми в моделі MBQC, глибина паралельних контурів також зменшиться. Це причина, чому оптимізації в моделі MBQC можуть зменшити глибину квантових схем.

Обрано контроль доступу, на основі атрибутів (ABAC), як модель авторизації, оскільки варіант використання вимагає, щоб контроль доступу надавався на основі прав, а не ідентичності. Крім того, ABAC дозволяє визначати правила доступу з великим рівнем деталізації.

РОЗДІЛ 4

РЕАЛІЗАЦІЯ

4.1. Архітектура квантового компілятора

Модель і алгоритми MBQC були реалізовані на неграфічному рівні, а інтерфейс користувача та квантова схема на графічному рівні. Було вирішено реалізувати Quantum Circuit у графічній частині, тому що використання підпрограми вводу/виводу Qt Framework, і деякі класи в Qt, здавалося, дуже добре підходить для реалізації логіки моделі Quantum Circuit.

Остаточна архітектура програми дещо відрізняється від початкової, що можна побачити, порівнявши початкову архітектуру на рис. 4.1 з кінцевою на рис. 4.2. Щоб отримати робочу версію програми, перша архітектура вимагала б спочатку створити графічний інтерфейс, оскільки файловий ввод-вивод і квантовий ланцюг мали бути реалізовані на графічному рівні. На початку роботи було прийняте рішення, що оскільки графічний інтерфейс не повинен бути основною особливістю нашого проекту, було вирішено спочатку розробити інтерфейс проекту за допомогою простого інтерфейсу командного рядка. Таким чином буде можливість додати графічний інтерфейс наприкінці проекту. Було переміщено файл введення/виводу на неграфічний шар і розділили модель квантової схеми на графічну та неграфічну частини, зробивши неграфічну частину незалежною від графічної частини. Також було додано новий шар під назвою Console Implementation, який реалізує інтерфейс користувача командного рядка, і створений невеликий модуль допоміжних програм до шару Non-Graphical Implementation.

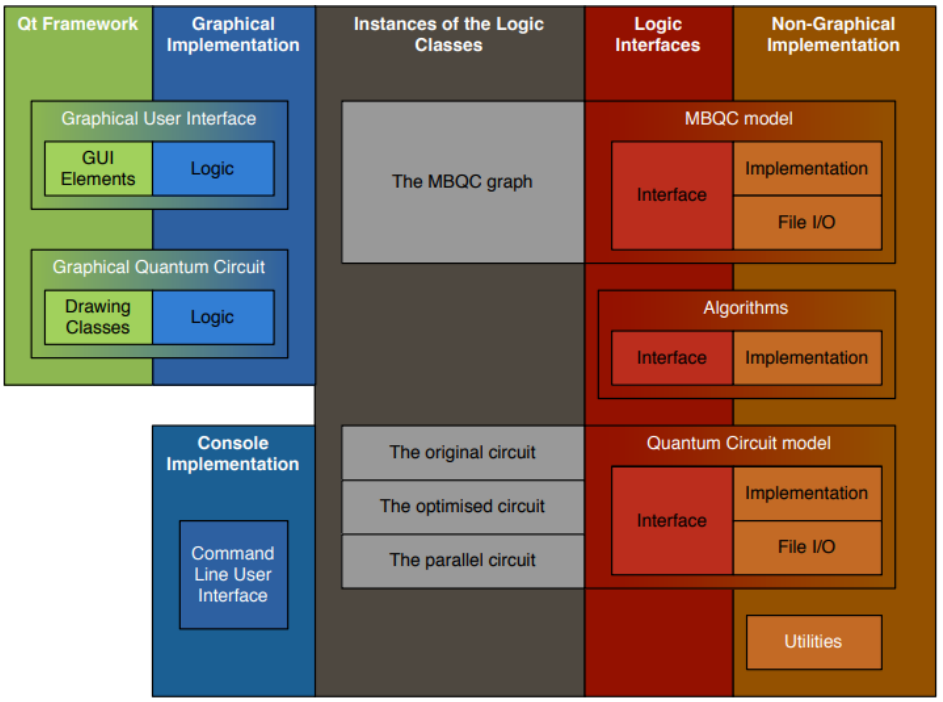


Рис. 4.2. Остаточна архітектура програми. Розділ під назвою «Екземпляри логічних класів» представляє екземпляри елементів з логічної частини, які використовують як графічна, так і консольна реалізація.

4.2. Дизайн квантового компілятора

Остаточний дизайн програми повністю відрізняється від початкового. Тому початковий проект не представлено, його можна знайти в документі з підготовки проекту [10]. Остаточний дизайн представлений у наступних двох розділах, але спочатку окреслимо причини змін дизайну. Існували три основні проблеми, які вимагали переосмислення дизайну:

- Поділ коду Quantum Circuit на графічні та неграфічні частини.
- Використання нового алгоритму для оптимізації шаблонів MBQC.
- Погана продуктивність початкової реалізації графічної квантової схеми.

Для зручності будемо посилатися на інтерфейсний та неграфічний шари, описані в розділі архітектури 4.1 та рис. 4.2, як back-end нашої програми. Графічна реалізація буде називатися інтерфейсом графічного інтерфейсу, а консольна реалізація — інтерфейсом консолі.

У розділі 4.1 було пояснено, що було прийняте рішення розділити модель квантової схеми на окремі графічні та неграфічні частини, спочатку все це було реалізовано як графічне. Це означало, що довелося перенести логіку квантових схем із інтерфейсу GUI на внутрішній. Тому довелося змінити дизайн back-end.

Початковий дизайн передбачав, що ми виконаємо оптимізацію MBQC, описану в розділі 3.1, на шаблоні MBQC. Під час проекту розроблено новий і швидший алгоритм (розділ 3.3), який працює на спеціальному типі графів MBQC (розділ 3.2). Щоб реалізувати цей алгоритм і граф MBQC, довелося повністю переробити частину моделі MBQC.

Початкова версія графічної схеми, яку розробили для інтерфейсу графічного інтерфейсу, використовувала класи Qt для автоматичного розташування вентилів у схемі. Самі вентилі також були отримані від класів Qt, які несуть багато функціональних можливостей, непотрібних для цього проекту. Цей дизайн був обраний, тому що він полегшив би редагування квантових схем за допомогою графічного інтерфейсу. Недоліком було те, що було неможливо відобразити схеми з більш ніж двадцятьма вентилями. Редагування схем через GUI не було вимогою для програми, тому ми переробили частину графічної схеми, нехтуючи можливістю редагування схем. Згідно з новим дизайном, вентилі складаються з класів Qt нижнього рівня, і розташування елементів у схемі розраховується вручну лише один раз, коли схема створюється. Новий дизайн дозволив нам візуалізувати квантові схеми, які можливо розпаралелювати з нашою програмою.

4.2.1. Back-end

Внутрішня частина складається з неграфічного та інтерфейсного шарів, показаних на рис. 4.2. Він розділений на чотири логічні частини, показані на рис. 4.3:

- Квантовий ланцюг
- Граф MBQC
- Алгоритми, що перетворюють квантові схеми в графи MBQC і навпаки.
- Функції утиліти.

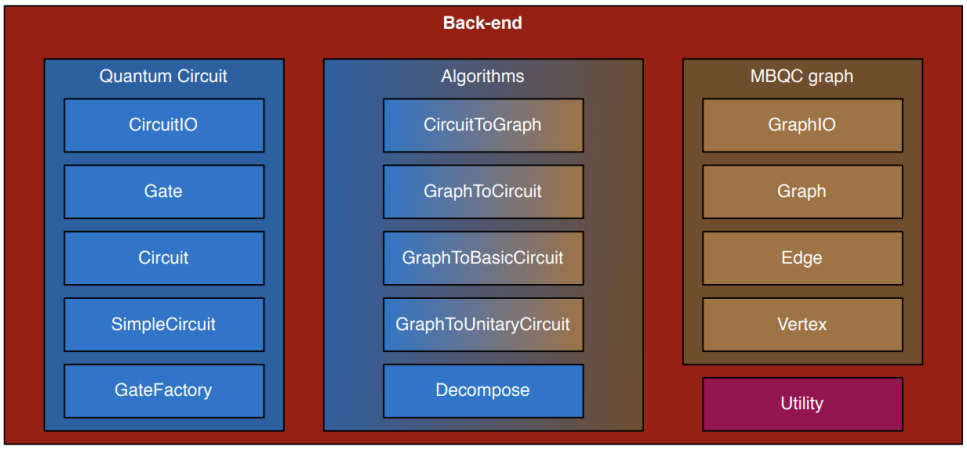


Рис. 4.3. Дизайн серверної частини, що показує класи та модулі, що використовуються у back-end. Алгоритм декомпозиції синього кольору, оскільки він переводить квантовий контур в інший ланцюг і не використовує граф MBQC.

Файли Quabtum Circuit містять класи для опису квантової схеми та функції для вводу/виводу. Найосновнішим елементом квантової схеми є Gate, клас Circuit є підкласом Gate. Це дозволяє нам додавати підсхеми як вентилі до ланцюгів. Такий тип підсхеми може, наприклад представляти складний квантовий вентиль, який складається з кількох більш примітивних елементів. Підкласом класу Circuit є клас SimpleCircuit. Цей клас може складатися тільки з вентилів, що діють на один кубіт, і він використовується в графічному інтерфейсі для ідентифікації наборів вентилів, які можуть бути намальовані на тому ж етапі обчислення в схемі.

Файли MBQC graph містять класи для графа, його ребра та вершини, а також функції введення/виводу для графа. Граф має структуру, описану в розділі 3.2, і функції для додавання елементів до графа, описаного в розділі 3.3.

Підпрограми вводу-виводу для графа та схеми записані як окремі функції, щоб зробити граф і класи схем незалежними від будь-якого вхідного та вихідного формату та функцій. Ці функції для графа Quantum Circuit і MBQC знаходяться в окремих файлах із коду Quantum Circuit і MBQC Graph. І квантові схеми, і граф MBQC можна записати у вихідний потік, але оскільки наша програма призначена для розпаралелювання квантових схем, з вхідного потоку можна зчитувати лише екземпляри квантових схем. Немає потреби вводити граф MBQC до нашої програми.

Algorithms є окремою колекцією функцій, оскільки вони не належать ні класу Graph, ні класу Circuit, а обом. Збереження їх в окремому файлі зменшує залежності між схемою та графом. Схемі не потрібно знати про граф і навпаки. Також ми можемо легко додати або змінити функції алгоритму, не змінюючи код схеми або графа. Ми реалізували п'ять алгоритмів у Algorithms:

- CircuitToGraph - перетворює квантовий контур у граф MBQC.
- GraphToCircuit – перетворює граф MBQC у квантовий контур. Використовує алгоритми, описані в розділах 3.4 і 3.5.
- GraphToBasicCircuit – перетворює граф MBQC у квантовий контур, але не застосовує остаточні оптимізації, описані в розділі 3.5.
- GraphToUnitaryCircuit – перетворює граф MBQC у квантовий контур, але не створює ініціалізацію, схему та блок вимірювань. Ця схема використовується програмами перевірки для розрахунку унітарного оператора оптимізованої схеми.
- Decompose – перетворює схему на нову схему, де всі вентилі розкладаються на елементи J(α) і $\wedge Z$.

Модуль Utilities містить лише допоміжну функцію для перетворення об’єктів C++ (або елементів вбудованого типу) у рядки C++.

4.2.2. Front-end

Цей розділ описує інтерфейс графічного інтерфейсу. Інтерфейс консолі є легким, і його реалізація складається лише з одного файлу, який відповідно до аргументів командного рядка викликає відповідні внутрішні функції.

Діаграма класів, що описує дизайн інтерфейсу, представлена на рис. 4.4. Як видно з цього малюнка, інтерфейс залежить від back-end лише через класи CircuitView і CircuitScene. Це було розроблено, щоб мінімізувати залежність інтерфейсу від back-end. На рис. 4.4 представлені основні класи Qt, які використовуються разом із класами, які були реалізовані в цьому проекті.

Як видно з рис. 4.4, у головному вікні є три класи, які відповідають за взаємодію з користувачем та відображення схеми та її властивостей. На рис. 4.5

показано знімок екрана програми та виділені частини графічного інтерфейсу, які реалізовані цими трьома класами. Спілкування та обов'язки цих класів також показано на рис. 4.6. Інтерфейс містить три екземпляри схем: вихідні, оптимізовані та паралельні схеми. Вхідні дані з ControlPanel використовуються для вибору схеми, яка має відображатися, клас CircuitView намалює вибрану схему, а клас Circuit info покаже глибину схеми та кількість використовуваних кубітів.

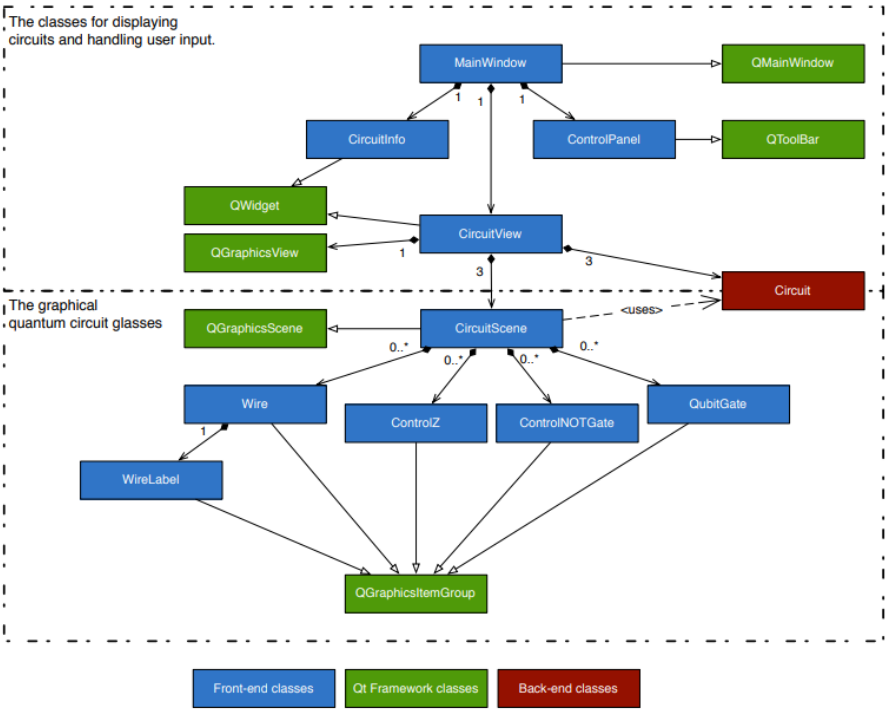


Рис. 4.4. Діаграма класів, інтерфейсу програмного забезпечення

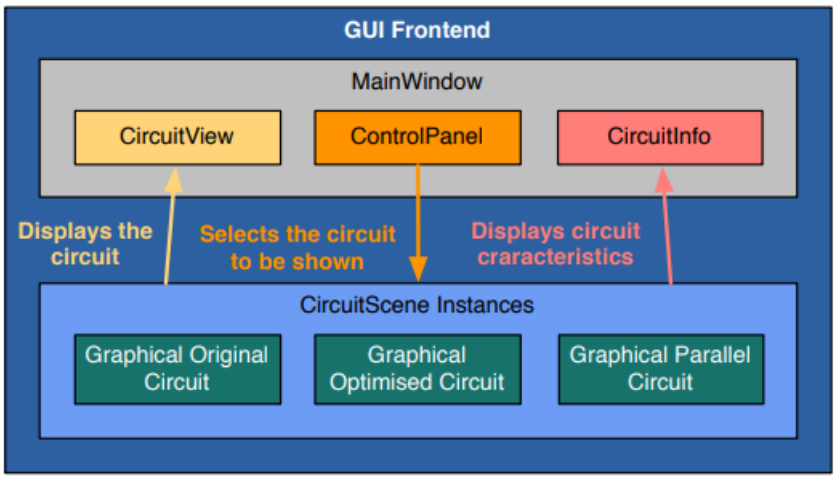


Рис. 4.5. Взаємодії в інтерфейсі

4.3. Архітектура авторизації

На рис. 4.7 зображена реалізована архітектура. Кожна фігура з єдиною суцільною рамкою на малюнку є процесом, що виконується в окремій машині, якщо не вказано інше.

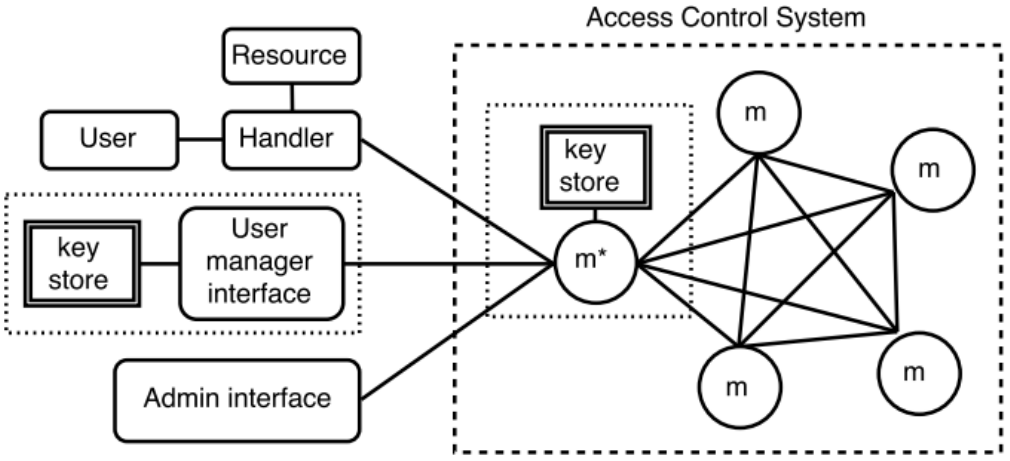


Рис. 4.7. Архітектура

Access control system - система контролю доступу складається з приватної мережі Ethereum. Мережа має п’ять вузлів з можливістю майнінгу. Лише один вузол забезпечує кінцеву точку HTTP для викликів RPC, а саме, виклик функцій смарт-контракту. Додаток А описує деталі апаратного та програмного забезпечення, що використовуються для реалізації вузлів.

User manager interface - це інтерфейс гаманця користувача, який взаємодіє зі смарт-контрактом через javascript API, наданий вузлом майнера, позначеним як m* на рис. 5.4. Інтерфейс менеджера користувачів генерує ключ облікового запису менеджера користувачів локально з початкового рівня, підписує транзакцію також локально і відправляє вже підписану транзакцію на вузол, єдиною функцією якого є просування підписаної транзакції в мережу Ethereum. Як альтернатива використанню гаманця, інтерфейс менеджера користувачів може покладатися на вузол майнера, щоб керувати своїми ключами та підписувати транзакції від його імені. У цьому випадку інтерфейс менеджера користувачів не має локального доступу до ключів. Під час виклику функції через транзакцію транзакція надсилається вузлу без підпису. Потім вузол отримує доступ до приватного ключа

менеджера користувачів, підписує транзакцію і просуває її в мережу. Закриті ключі знаходяться в сховищі ключів, яке знаходиться у файловій системі машини, де був створений або імпортований обліковий запис менеджера користувачів, а саме, вузол майнера, що забезпечує кінцеву точку інтерфейсу менеджера користувачів, як показано на рис. 5.4.

Administrator interface - це програма JavaScript, яка взаємодіє зі смарт-контрактом через API JavaScript, що надається вузлом майнера, позначеним як m* на рис. 5.4. Функції реєстрації менеджерів користувачів і написання прайс-листів викликаються за допомогою транзакцій, підписаних закритим ключем адміністратора. Інтерфейс адміністратора покладається на вузол майнера для підписання транзакцій за допомогою приватного ключа адміністратора. Оскільки інтерфейс адміністратора є локальним для вузла майнера, він миттєво розблокує свій обліковий запис перед викликом функцій через підписані транзакції.

Handler - виконує функцію точки виконання політики. Він перехоплює запити від користувачів, запитує систему контролю доступу щодо рішення про авторизацію і виконує таке рішення. У типовій роботі обробник отримує запит на доступ від користувача і, у свою чергу, надсилає запит Verify до системи контролю доступу. Аргументи, необхідні для виклику Verify, беруться безпосередньо з корисного навантаження запиту доступу. Обробник спілкується зі смарт-контрактом через інтерфейс JSON RPC, наданий вузлом майнера, позначеним як m* на рис. 5.4, на відміну від інтерфейсу менеджера користувачів, який спілкується за допомогою Javascript API (останній призначений лише для програм Javascript, як і Корпус для інтерфейсу менеджера користувачів).

User - користувач надсилає запит на доступ до обробника з корисним навантаженням, як описано в розділі 5.2.4 (Типовий етап операції). userId та userM anagerId — це атрибути користувача, завантажені до користувача його менеджером користувачів після етапу ініціалізації, тоді як resourceId та object — це атрибути ресурсу, виявлені користувачем. Механізм виявлення доступних ресурсів виходить за рамки нашої реалізації.

4.4. Висновки до розділу 4

Отже, модель і алгоритми MBQC реалізовані на неграфічному рівні, а інтерфейс користувача та квантова схема на графічному рівні. Вирішено реалізувати Quantum Circuit у графічній частині, тому що використання підпрограми вводу/виводу Qt Framework, і деякі класи в Qt, тому що він підходить для реалізації логіки моделі Quantum Circuit.

Існували три основні проблеми, які вимагали переосмислення дизайну:

- Поділ коду Quantum Circuit на графічні та неграфічні частини.
- Використання нового алгоритму для оптимізації шаблонів MBQC.
- Погана продуктивність початкової реалізації графічної квантової схеми.

Внутрішня частина складається з неграфічного та інтерфейсного шарів. Він розділений на чотири логічні частини:

- Квантовий ланцюг
- Граф MBQC
- Алгоритми, що перетворюють квантові схеми в графи MBQC і навпаки.
- Функції утиліти.

Інтерфейс консолі є легким, і його реалізація складається лише з одного файлу, який відповідно до аргументів командного рядка викликає відповідні внутрішні функції.

РОЗДІЛ 5

РЕЗУЛЬТАТИ ТА АНАЛІЗ

5.1. Квантовий компілятор

5.1.1. Сходи Тоффолі

Були проведені експерименти зі схемами, які складаються тільки з вентилю Тоффолі (див. рис. 5.1 для вентиля Тоффолі). Ці схеми цікаві тим, що їх також можна реалізувати в класичній моделі оборотних обчислень. Зокрема, вентиль Тоффолі є універсальним для класичних реверсивних обчислень, і, таким чином, оборотні класичні обчислення можна реалізувати за допомогою лише вентилів Тоффолі. Ми намагалися знайти класичні обчислення, які можна було б розпаралелювати в квантових обчисленнях, але не можна розпаралелювати або нелегко розпаралелювати в класичній моделі.

$$TOFFOLI = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} =$$

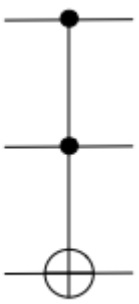


Рис. 5.1. Вентиль Тоффолі

Однією цікавою конструкцією, була сходи Тоффолі. Ми побудували сходи, з’єднавши цільовий кубіт вентиля Тоффолі з першим контрольним кубітом наступного вентиля Тоффолі, як показано на рис. 5.1. Наша програма вводить деякі початкові накладні витрати до глибини малих оптимізованих схем. Після 5 вентилів Toffoli глибина оптимізованих схем стабілізується до лінійної глибини, як показано на рис. 5.3. Додавання ще одного вентиля Тоффолі до початкового контуру додасть 13 кроків до глибини, але глибина оптимізованого контуру збільшиться лише на 10

кроків. Це означає, що наша програма оптимізує сходи Toffoli, які мають лінійну глибинну складність, до іншої еквівалентної схеми з лінійною складністю глибини. Нова схема матиме менший постійний коефіцієнт глибинної складності.

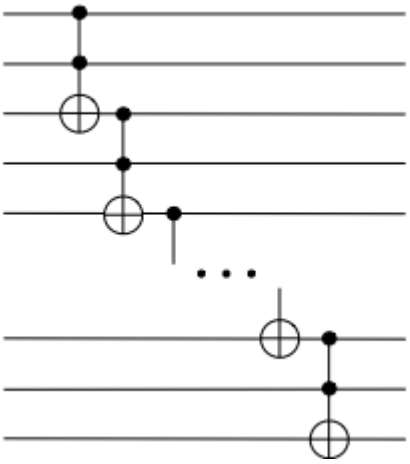


Рис. 5.2. Схема сходів Тоффолі

Цікаво, що кількість кубітів в оптимізованій схемі сходів Тоффолі є лінійною за розміром входу квантової схеми (рис. 5.4). У наших тестах реалізований метод зменшує глибину контуру і розширює ширину на постійний коефіцієнт.

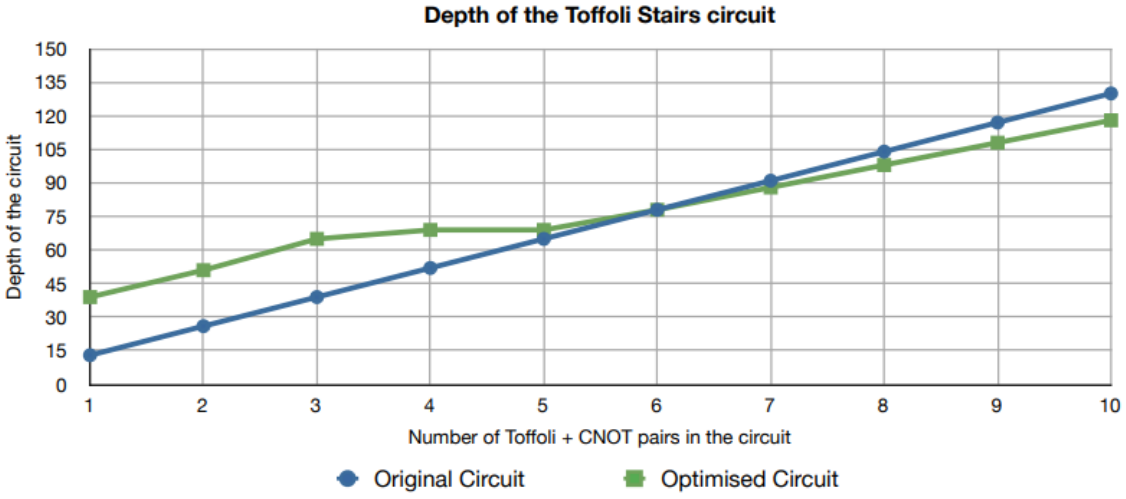


Рис. 5.3. Глибина контуру сходів Toffoli, в залежності від кількості «сходових ступенів» у вихідній схемі.

Коли замінюємо кожен другий вентиль Toffoli в ланцюзі Toffoli вентилем CNOT, ми отримуємо схему Toffoli + CNOT, як показано на рис. 5.5. Кількість

кубітів, що використовуються в оптимізованій схемі, є лише постійним коефіцієнтом, більшим, ніж у вихідній схемі (див. рис. 5.7). Те саме відбувається зі сходами Toffoli, але відмінність цієї схеми полягає в тому, що наша програма оптимізує цю схему на постійну глибину. Знову ж таки, є постійні накладні витрати, але, як показано на рис. 5.6, для більших схем це створить квантовий контур глибиною 47. Оскільки вихідна схема є схемою лінійної складності, це значне покращення.

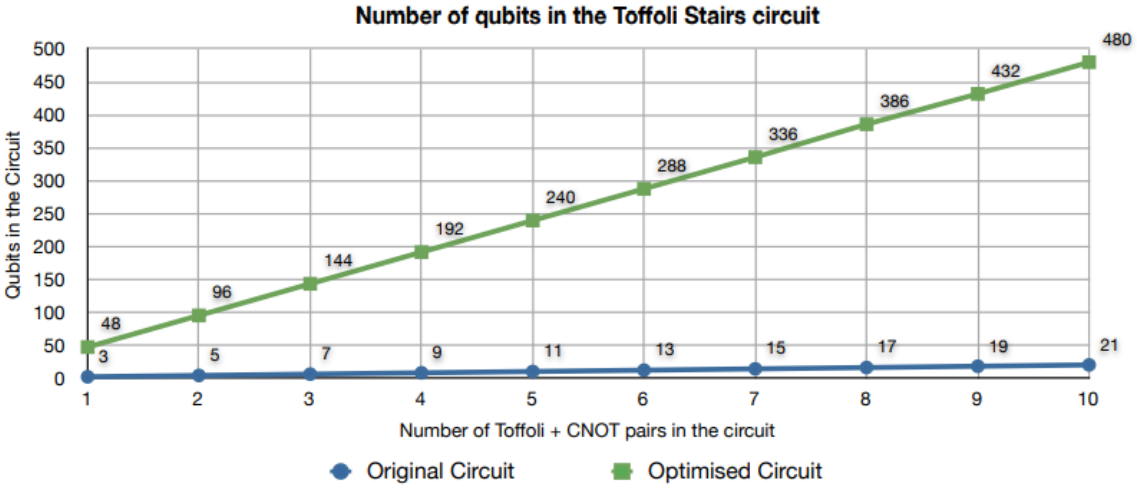


Рис. 5.4. Кількість кубітів у схемі сходів Тоффолі, залежно від кількості «сходових ступенів» у вихідній схемі

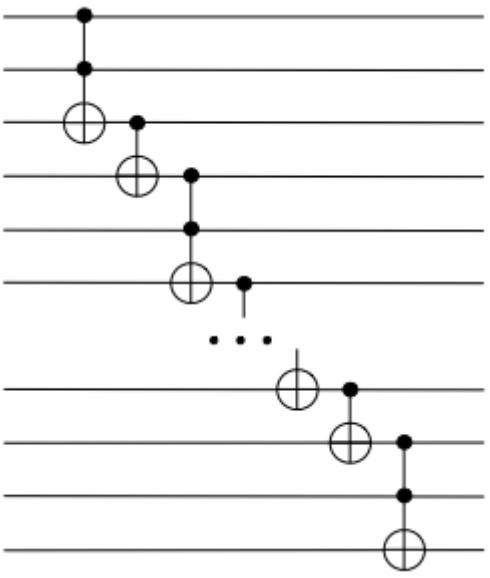


Рис. 5.5. Схема сходів Toffoli + CNOT

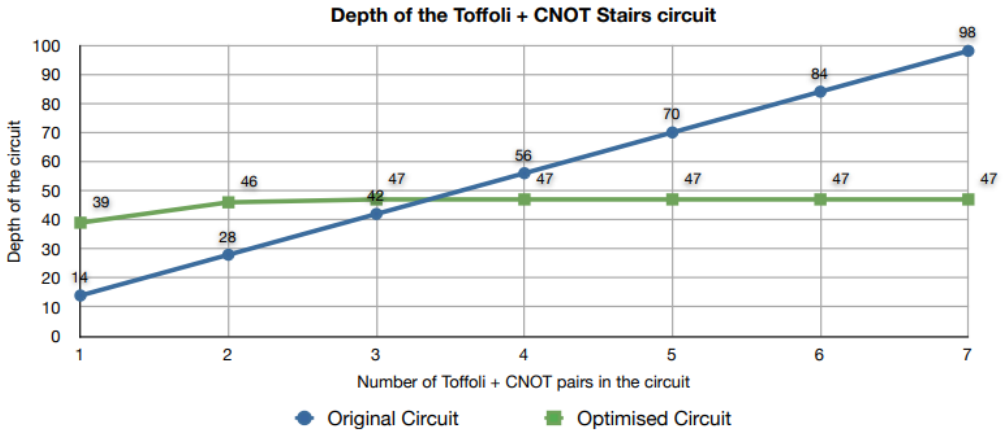


Рис. 5.6. Глибина схеми Toffoli + CNOT Sairs, в залежності від кількості «сходових ступенів» у вихідній схемі.

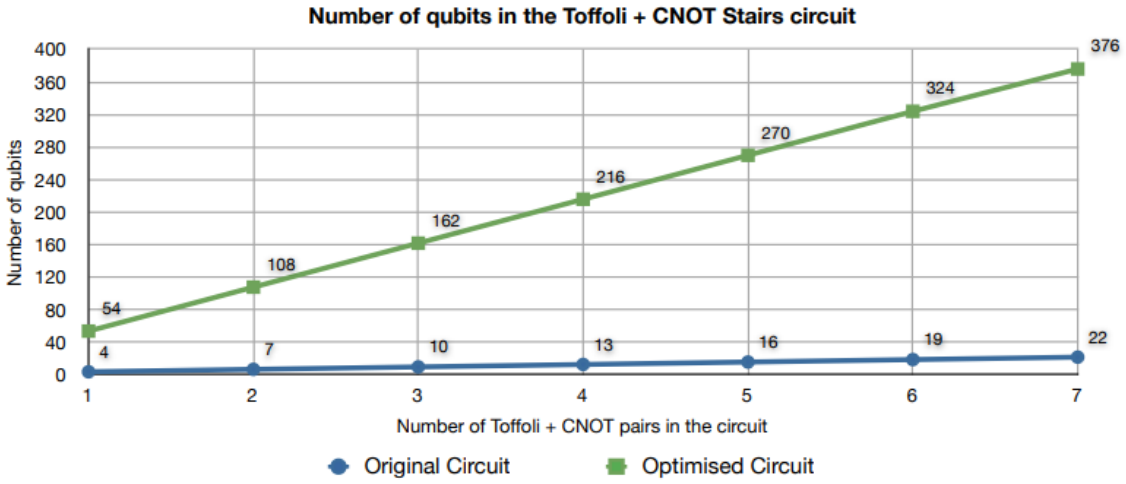


Рис. 5.7. Кількість кубітів у схемі Toffoli + CNOT Sairs, залежно від кількості «сходинок» у вихідній схемі

5.1.2. QFT

Протестувавши нашу програму на алгоритмі QFT, описаному в розділі 2.2.1., спробували відтворити результати Кліва та Уотруса в [2], де вони показали, що QFT можна обчислити на глибині $O(log_n)$. Зробити це виявилось неможливим, тому що алгоритм, який ми використовували, був для розрахунку точного QFT, але результат з [2] застосовується для приблизного QFT. Тим не менш, цікаво побачити, що робить програма з нетривіальним квантовим контуром. Створили схеми QFT до десяти кубітів і оптимізували ці схеми. Результати показані на рис. 5.8. Використані нами

вхідні схеми QFT відповідають алгоритму QFT з лінійною глибинною складністю. Це також найвідоміша глибина для точного QFT, тому не дивно, що ми не змогли зменшити глибину схем QFT. Аналіз у розділі 3.5 показує, що в найбільш загальному випадку глибина ланцюгів збільшиться на коефіцієнт $O(log_n)$. Тести, які були проведені (рис. 5.8), свідчать про те, що це стосується алгоритму QFT. Цікавим майбутнім експериментом було б запустити нашу програму на наближеному алгоритмі QFT, який має глибину O(n), і перевірити, чи зможемо ми відтворити результати в [2] і чи зможемо ми паралелізувати їх до глибини $O(log_n)$.

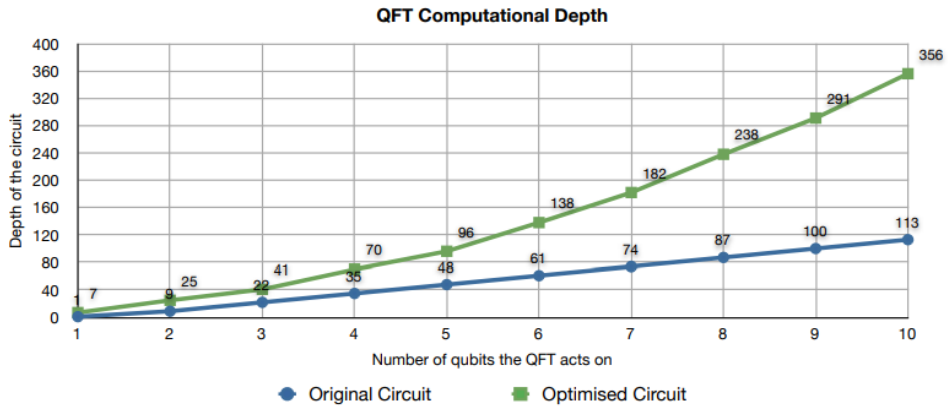


Рис. 5.8. Глибина квантових схем, що реалізують алгоритм QFT

5.2. Системи авторизації

Якісна оцінка зосереджена на оцінці функцій адміністрування та підтримки, безпеки та використання ресурсів нашого рішення та найсучаснішого стану. У таблиці 6.3 узагальнено якості оцінених рішень.

Делегування адміністративних завдань: рішення надає можливість призначити менеджерам користувачів повноваження реєструвати власних користувачів, а рівень повноважень обмежується смарт-контрактом. Ми вважаємо цей метод дуже безпечним і простим, оскільки такі адміністративні завдання ініціюються лише транзакціями, а Ethereum піклується про аутентифікацію менеджерів користувачів, а саме — перевірку підписаних транзакцій.

Делегування дозволів користувачів: наше рішення не дозволяє цю функцію, тому що в ньому користувачі не пов’язані з обліковим записом Ethereum, тому вони не можуть самостійно змінити стан блокчейну, а саме дозволи.

Визначення складних правил доступу: рішення, засновані на CapBAC, мають обмежену виразність у своїх правилах доступу, оскільки вони повинні бути закодовані в маркері авторизації. Ці правила повинні бути досить простими, щоб вони не мали великих витрат на обчислення, коли ресурси, які отримують маркер, інтерпретують правила авторизації та дозволи, вбудовані в нього. Наше рішення використовує функції розумного контракту для визначення правил авторизації, використовуючи переваги мови логіки першого порядку, і теоретично немає обмежень у сховищі, яке може мати контракт.

Аудит: стосовно запитів на доступ, враховуючи використання транзакцій для отримання маркерів доступу, тоді як у нашому рішенні запити на доступ не передбачають транзакції. Однак вважаємо, що можна запрограмувати наш розумний контракт на виклик події після кожного рішення про авторизацію, ініціюючи об’єкт, зовнішній для блокчейну, для реєстрації даних події. Крім того, кожен ресурс може реєструвати запити на доступ локально, однак для консолідації журналів усіх ресурсів знадобиться додатковий механізм. З точки зору запиту стану системи, усі дані, які знаходяться в блокчейні, підлягають аудиту, а їх автори — відстежуються, оскільки зміни стану викликаються підписаними транзакціями, що дозволяє не відмовлятися.

ВИСНОВКИ

Як було представлено, головна мета цього проекту, написання програми, яка автоматично перекладає квантові схеми за допомогою моделі MBQC, була повністю досягнута. Реалізовано дві версії цієї програми: графічну та консольну. Ця програма є корисним інструментом, має дві сфери застосування. Найочевидніший спосіб використовувати це для зменшення глибини квантових схем, коли глибина ланцюга важлива. Це корисно при реалізації квантової схеми на експериментальних квантових комп'ютерах, де час життя кубіта короткий. Інше використання цієї програми полягає у вивченні паралельності квантових схем. Цю програму можна використовувати як корисний інструмент для пошуку нових класів ланцюгів, які можна розпаралелювати в моделі квантової схеми, а також після реалізації програми.

Використовуючи реалізовану програму розпаралелювання, було знайдено пару класів схем, де наша програма зменшувала б глибину схем. Ці результати важливі, оскільки раніше не існувало жодного методу розпаралелювання цих схем. Оскільки наші тести були обчислювальними, теоретичне доведення цих результатів потребує подальшої роботи.

Два класи паралельних ланцюгів – це Toffoli і Toffoli+CNOT. Схема Toffoli можна розпаралелювати на глибину, яка є постійним фактором, меншим за глибину вихідного контуру. Схема сходів Toffoli+CNOT може бути паралелізована на постійну глибину. Для обох цих схем кількість кубітів, що використовуються паралельною версією, є лише постійним коефіцієнтом, більшим, ніж використовуваний в оригінальній схемі. Оскільки обидві ці схеми можуть бути реалізовані в класичній реверсивній обчислювальній моделі, було цікаво дізнатися, чи можна знайти метод розпаралелювання класичної версії, чи якась властивість квантових обчислень робить їх паралельними в моделі квантової схеми.

Нами був знайдений набір вентилів, які можна використовувати для побудови схем, які завжди мають логарифмічну глибину. Це набір вентилів: $\wedge$Z, CNOT, ω, Z(α) і J($-\frac{\pi}{2}$) вентилі. Раніше було відомо, що ланцюг, що складається лише з групових вентилів Кліффорда, паралельний до логарифмічної глибини.

Запропонований набір вентилів цікавий, тому що за допомогою цих вентилів стало можливим побудувати схеми, які були неможливими з груповими вентилями Кліффорда. З іншого боку, ми не можемо побудувати всі схеми, які могли б за допомогою групових воріт Кліффорда. Це означає, що класи схем, які реалізуються нашим набором вентилів і елементами групи Кліффорда, різні, вони мають непорожній перетин, але жодна з них не є підмножиною іншого. Відкрите питання щодо цих вентилів – наскільки потужний цей набір вентилів. Які квантові схеми ми можемо побудувати за допомогою нього, і чи є будь-яка з цих схем з якоїсь причини цікава чи корисна.

На додаток до пошуку нових квантових схем, які можна паралелізувати, експериментували з деякими добре відомими квантовими схемами. Було підтверджено експериментально результат з [4], сказавши, що схеми зі спеціальною структурою можна розпаралелювати на логарифмічну глибину. Це показало, що можливо відтворення попередніх результатів за допомогою програми. Також була запущена програма на схемах для точного алгоритму QFT. Результати співпали з теорією, яка передбачала збільшення глибини в загальному випадку на коефіцієнт $O(log_n)$.

Як побічний продукт реалізації програми розпаралелювання був розроблений новий ітераційний алгоритм для оптимізації обчислень у моделі MBQC. Цей алгоритм будує оптимізований граф MBQC, який потребує $O(n^3)$ часу для виконання. Цей граф еквівалентний оптимізованому шаблону MBQC. Попередній найвідоміший алгоритм, який можна було використовувати для оптимізації шаблонів MBQC, має час виконання $O(n^3)$. Хоча обидва ці алгоритми можна використовувати для створення оптимізованого шаблону MBQC, вони не є еквівалентними, оскільки наш алгоритм буде використовувати квантову схему як вхід, тоді як попередній алгоритм використовує шаблон MBQC як вхід. Таким чином, наш алгоритм має обмеження на роботу тільки з шаблонами MBQC, отриманими з квантових схем, тоді як попередній алгоритм працює з кожним шаблоном MBQC.

Подальша робота

Реалізовану програму можна було б покращити кількома способами, щоб зробити її більш зручною у використанні та представити більше інформації про схеми:

- GUI може бути змінений, щоб дозволити користувачеві малювати або змінювати вхідні схеми через GUI.
- Спосіб відображення квантових схем можна було б покращити, наприклад, показуючи для кожного елемента, на якому кроці вони можуть бути виконані.
- Можна відобразити більше інформації про квантові схеми. Наприклад, кількість допоміжних кубітів в оптимізованих схемах, кількість блоків затвора J(α) і глибина кожного з блоків затвора CNOT і $\wedge Z$ у схемі.
- Можна додати більше форматів файлів (.png, .jpg тощо), куди можна експортувати схеми.

Ці покращення стосуються графічного інтерфейсу програми.

Алгоритми, які реалізовані програмою, також можуть бути покращені. Поточний алгоритм побудови й оптимізації графа MBQC дозволяє додавати до нього лише елементи J(α) та $\wedge Z$. Цей алгоритм можна покращити щонайменше двома способами. По-перше, можна було б покращити його, зробивши можливим додавання довільних ребер і вершин до графа. Другий спосіб покращити алгоритм полягає в додаванні додаткових методів оптимізації. Одним із них може бути, наприклад, техніка локальної комплементації [16]. Цей метод перетворює шаблон вимірювання на еквівалентний, змінюючи основний граф і кути вимірювання.

Нарешті, сама програма може бути використана для дослідження квантових схем. Його можна використовувати для ідентифікації більшої кількості класів квантових схем, які можна паралелізувати за допомогою реалізованого методу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kazuo Iwama and Shigeru Yamashita. Transformation rules for cnot-based quantum circuits and their applications. New Generation Computing, 2003. – 317 с.

2. Richard Cleve and John Watrous. Fast parallel circuits for the quantum fourier transform. Annual IEEE Symposium on Foundations of Computer Science, 2000. –526–536 с.

3. Dmitri Maslov, Gerhard W Dueck, D Michael Miller, and Camille Negrevergne. Quantum circuit simplification and level compaction. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2008. – 444 с.

4. Anne Broadbent and Elham Kashef. Parallelizing quantum circuits. Theoretical Computer Science, 2009. – 2510 с.

5. Vincent Danos, Elham Kashefi, and Prakash Panangaden. The measurement calculus. Journal of the ACM, 2007.

6. Michael A. Nielsen and Isaac L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.

7. Phillip Haye, Raymond Laflamme, and Michele Mosca. An Introduction to Quantum Computing. Oxford University Press, 2006.

8. Vincent Danos, Elham Kashefi, and Prakash Panangaden. Parsimonious and robust realizations of unitary maps in the one-way model. Phys. Rev. A, 2005.

9. Cristopher Moore and Martin Nilsson. Parallel quantum computation and quantum codes. SIAM Journal on Computing, 2001. 799–815 с.

10. Einar Pius. Msc project preparation report, 2010.

11. Vincent Danos and Elham Kashefi. Determinism in the one-way model. Phys. Rev. A, 2006.

12. Eric W. Weisstein. Undirected graph. – MathWorld-A Wolfram Web Resource – [Електронний ресурс]. – Режим доступу: http://mathworld.wolfram.com/UndirectedGraph.html.

13. Mehdi Mhalla and Simon Perdrix. Finding optimal flows efficiently. Proc. of 35th ICALP, 2008. – 868 с.

14. Niel de Beaudrap, Vincent Danos, and Elham Kashefi. Phase map decompositions for unitaries, 2006.

15. Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits, 2004.

16. M Hein, W Dür, J Eisert, R Raussendorf, M. Van den Nest, and H. J Briegel. Entanglement in graph states and its applications, 2006.

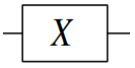
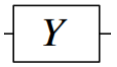
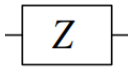
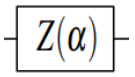
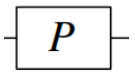
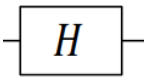
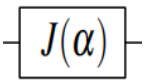
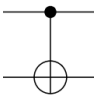
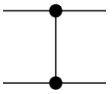
17. Pierangela Samarati and Sabrina Capitani de Vimercati. Access Control: Policies, Models and Mechanisms, 2001. 137–196 c.

18. Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A capability-based security approach to manage access control in the internet of things. Mathematical and Computer Modelling, 2013. 1189–1205 c.

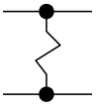
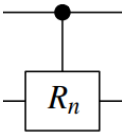
Додаток А

Таблиці

Таблиця 1. Основні вентилі

Назва вентиля	Символ вентиля та його унітарна матриця	Символ, що використовується в квантових схемах
Ідентифікації	$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	
Паулі X	$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	
Паулі Y	$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	
Паулі Z	$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	
Довільний поворот фази	$Z(\alpha) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$	
Фазове обертання	$P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	
Адамара	$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	
J	$J(\alpha) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{pmatrix}$	
CNOT	$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	
Контрольований Z	$\wedge Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	

Продовження таблиці 1

ω	$\omega = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ -1 & 1 & 1 & 1 \end{pmatrix}$	
Контрольно-фазовий	$R_n = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{\frac{2\pi i}{2^n}} \end{pmatrix}$	

Таблиця 2 - Модифікації графа залежно від кута α доданого елемента J(α).

	Загальний випадок	Вимірювання Паулі X	Вимірювання Паулі Y
M'_{o_i}	-α	0	$\frac{\pi}{2}$
S'_{o_i}	S_{o_i}	$\varnothing$	
S'_{n_j}	$T_{o_i}\Delta\{o_i\}$		$S_{o_i}\Delta T_{o_i}\Delta\{o_i\}$
T'_{o_i}	$\varnothing$		
T'_{n_j}	S_{o_i}		
E'	$E\Delta\{E_{o_i n_i}\}$		
V'	$V \cup \{o_i, n_i\}$		
O'	$(O \backslash \{o_i\}) \cup \{n_i\}$		
I'	$I \cup (\{o_i\} \backslash V)$		

Додаток Б

Програмний код

1. Лістінг функцію обробки транзакцій аунтифікацій:

```
func (s *service) ProcessingInternalTransactions(traceBlockRange, confirmations uint64, timeCheckNewBlock time.Duration) error {
done := make(chan struct{})
go gracefulShutdown(done)
errorStream := make(chan error, 5)
go s.errorsWather(done, errorStream)
writeAddress := make(chan string)
readAddress := make(chan []string)
transactionStream := make(chan []model.EthereumTransaction, 1024)
go addressMonitor(done, writeAddress, readAddress)
s.loadUsersAddresses(writeAddress)
go s.listenNewAddresses(done, errorStream, writeAddress)
go s.insertTransaction(done, errorStream, transactionStream)
lastProcessedBlock, err := s.mongo.GetFirstFailProcessedBlockNum()
if err != nil {return fmt.Errorf("couldn't get first fail processed block number w error %w", err)}
if lastProcessedBlock == 0 {
    lastProcessedBlock, err = s.mongo.GetLastSuccessfulProcessedBlockNum()
    if err != nil {return fmt.Errorf("couldn't get last successful processed block number w error %w", err)}
}
if lastProcessedBlock == 0 {return fmt.Errorf("couldn't get last processed block number")}
lastBlock, err := s.blockchain.GetLastLoadedBlockNum()
if err != nil {return fmt.Errorf("couldn't get last load block number w error %w", err)}
if lastBlock > lastProcessedBlock {
```

```
            if lastBlock-lastProcessedBlock > traceBlockRange {
                lastProcessedBlock = lastBlock - traceBlockRange
            }

            for i := lastProcessedBlock; i <= lastBlock; i++ {
                go s.catchInternalTransactions(done, errorStream, i, readAddress, transactionStream)
            }
        }
        newHeadNumberStream := make(chan uint64)
        err = s.blockchain.SubscribeToNewHeadNumber(done, errorStream, newHeadNumberStream, timeCheckNewBlock)
        if err != nil {return fmt.Errorf("couldn't subscribe to new head number w error %w", err)}
        for {
            select {
            case blockNumber := <-newHeadNumberStream:
                go s.catchInternalTransactions(done, errorStream, blockNumber, readAddress, transactionStream)
                go s.updateConfirmations(errorStream, blockNumber, confirmations)
            case <-done:
                s.logger.Infof("Stop...")
                return nil
            }
        }
}
```

2. Лістінг смарт-контракт ауторизації:

```
pragma solidity ^0.4.4;
contract Rights {
mapping (address => mapping(uint => bool)) users;
```

```
function set(address user, uint[] rights) returns(bool){
if(user != msg.sender) return false;
for(uint i = 0; i < rights.length; i++){
users[user][rights[i]] = true;
}
return true;
}
function get(address user, uint[] rights) constant returns(bool){
bool flag = true;
for(uint i = 0; i < rights.length; i++){
flag = flag && users[user][rights[i]];
}
return flag;
}
}
```

Демонстраційні матеріали

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра програмних систем і технологій

Програмне забезпечення квантових обчислень в навчальному процесі

Доповідач: Кирилов Іван Ігорович

Науковий керівник: Курченко Олег Анастасійович

Київ - 2022

Актуальність роботи

Сучасні квантові обчислення на декілька порядків продуктивніше виконують низку широко використовуваних алгоритмів у порівнянні з сучасними класичними обчислення, що побудовані на напівпровідникових логічних вентилях. Тому розробка алгоритмічно-програмного забезпечення для квантових комп'ютерів – це актуальна наукова і прикладна проблема.

Мета роботи полягає у підвищенні ефективності навчального процесу за допомогою розробки програмного забезпечення квантових обчислень..

Об’єкт досліджень - процес авторизації та компіляції квантових схем.

Предмет дослідження - методи і алгоритми компіляції квантових схем та авторизації.

Наукова новизна. Запропоновано метод компіляції квантової схеми для архітектур NISQ (QCC-NISQ). Ключовими компонентами цього методу є синтезатор схем, візуалізатор та метод паралельної обробки вентилів. Ці компоненти забезпечують важливу підтримку навчального процесу для подальшого розвитку практичних квантових комп’ютерів і дослідження квантових алгоритмів.

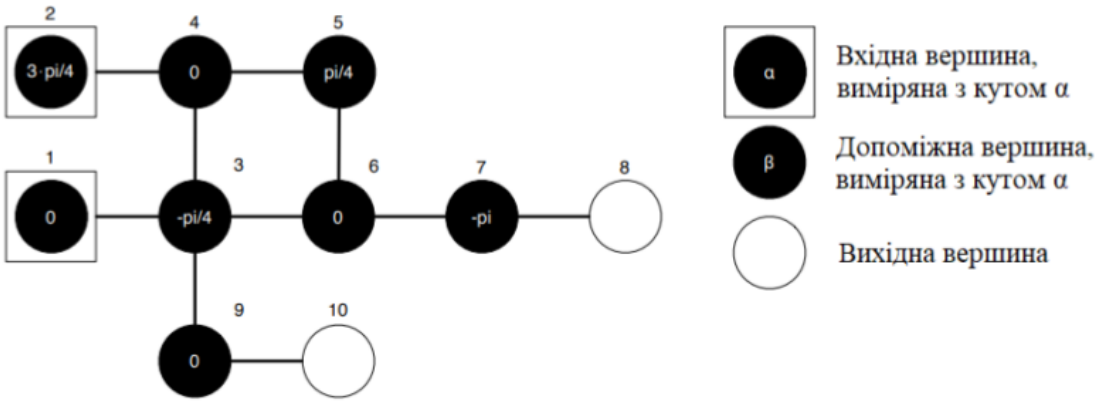
Проблема QCC для пристроїв NISQ

1. Обмеження набору воріт (кількість логічних елементів які процесор здатний використовувати)
2. Геометричні обмеження (кількість кубіті з якими можна взаємодіяти за допомогою логічних елементів)
3. Час виконання квантових схем

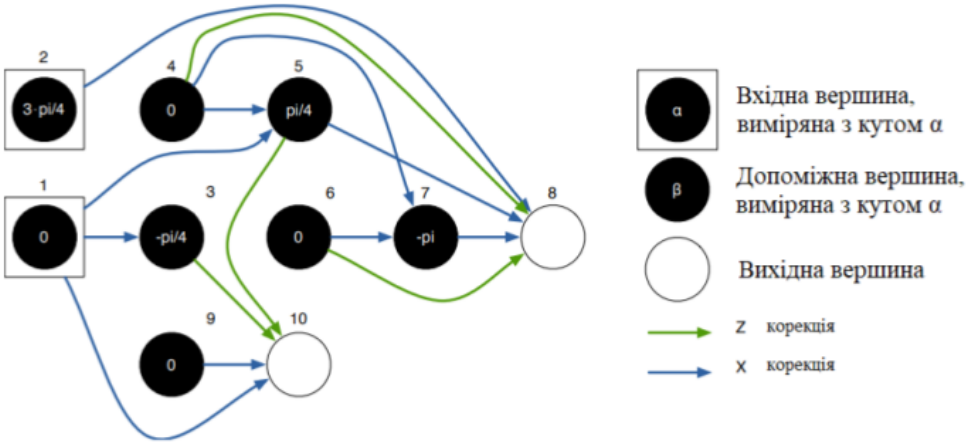
Проблема QCC для пристроїв NISQ

1. Обмеження набору воріт (кількість логічних елементів які процесор здатний використовувати)
2. Геометричні обмеження (кількість кубіті з якими можна взаємодіяти за допомогою логічних елементів)
3. Час виконання квантових схем

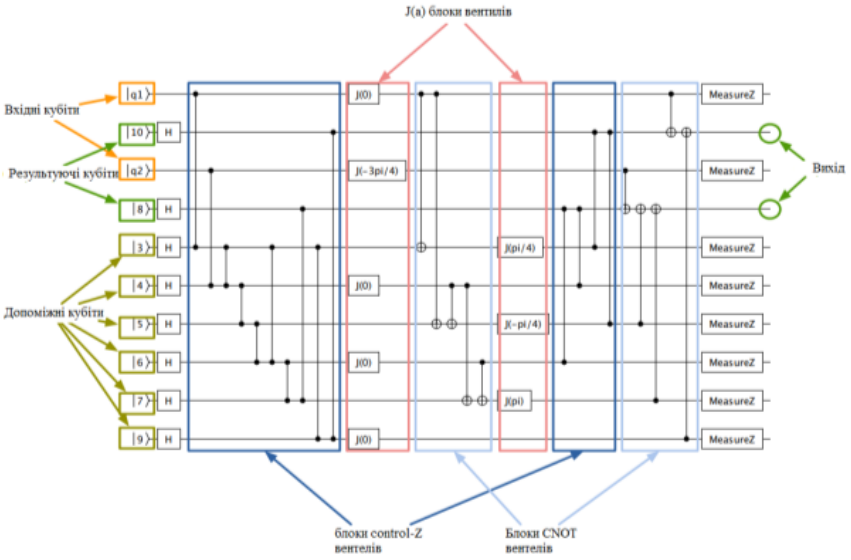
Метод оптимізації та паралельності компіляції квантової схеми для архітектур NISQ



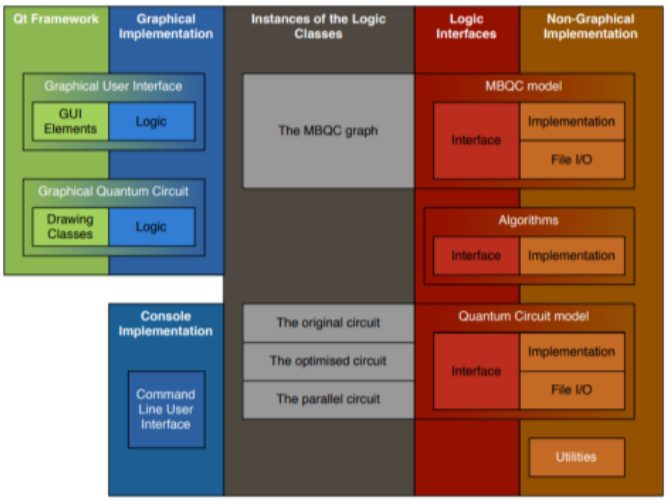
Метод оптимізації та паралельності компіляції квантової схеми для архітектур NISQ



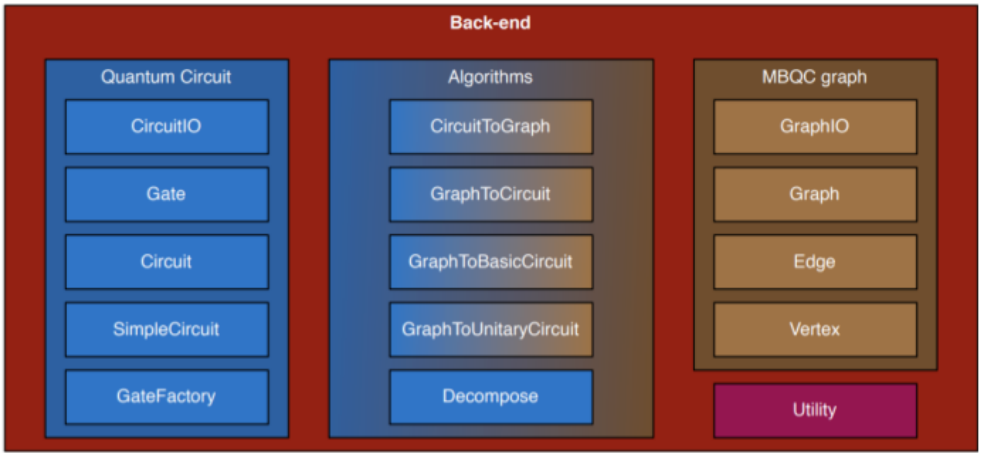
Приклад оптимізація



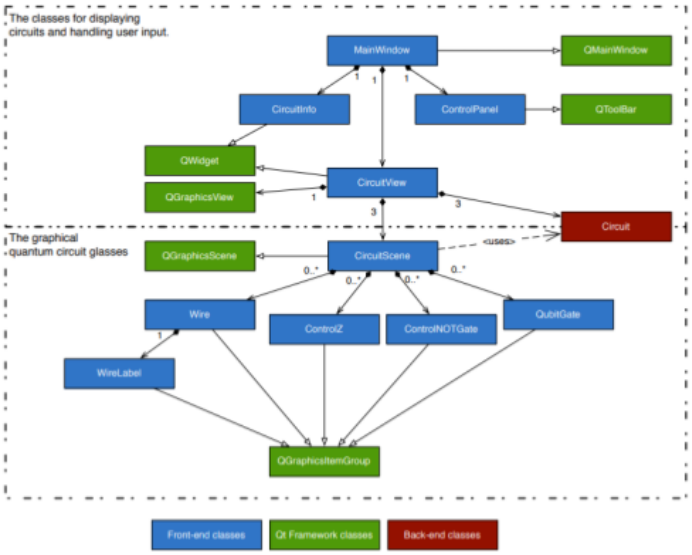
Архітектура квантового компілятора



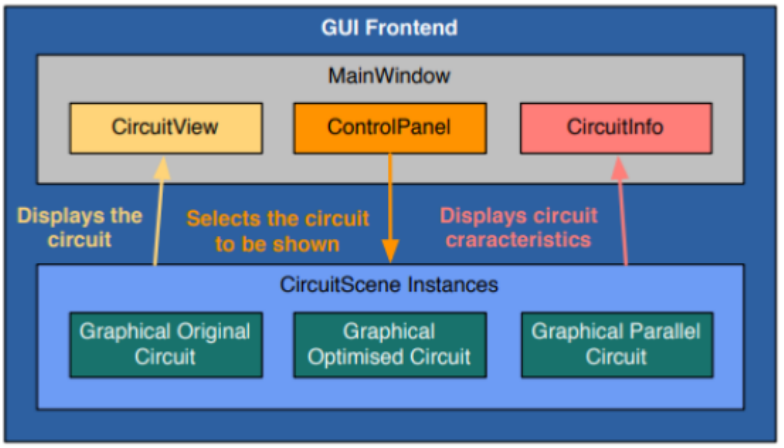
Архітектура квантового компілятора



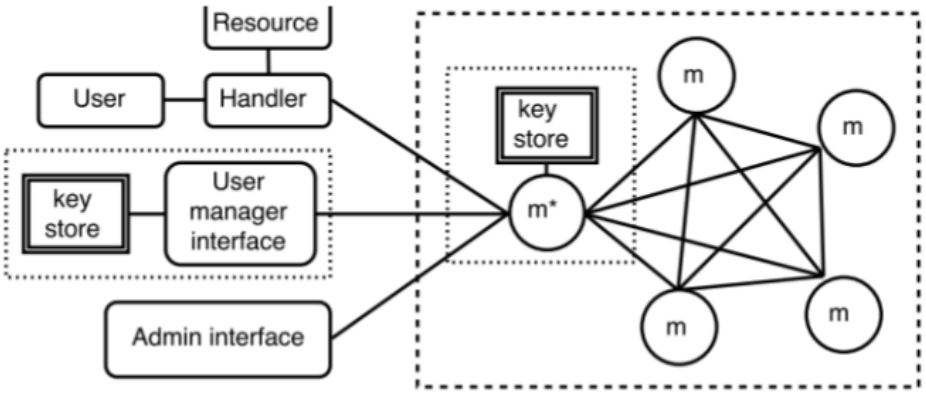
Архітектура квантового компілятора



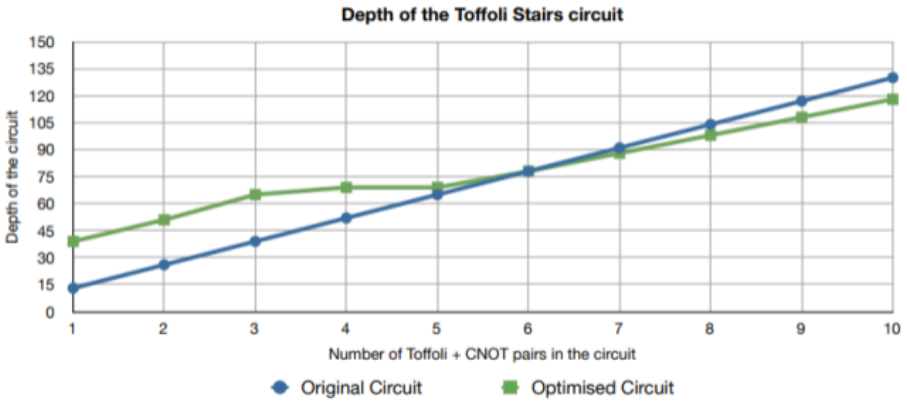
Архітектура квантового компілятора



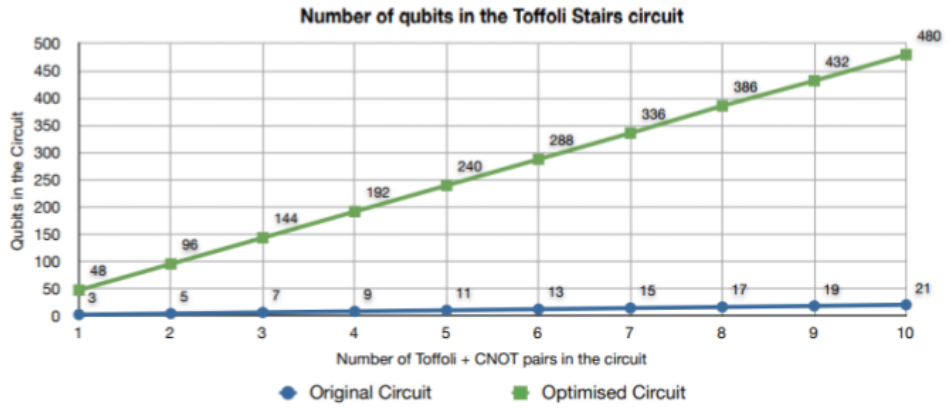
Архітектура авторизації



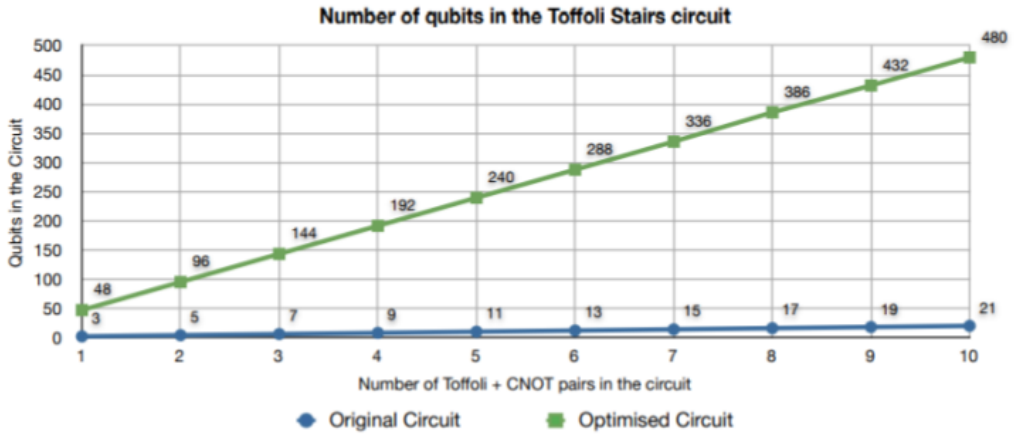
Результати



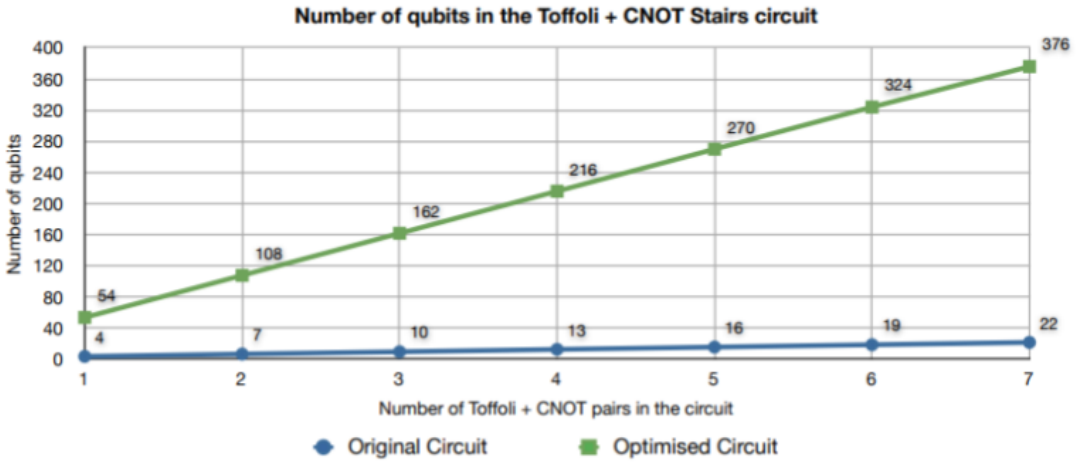
Результати



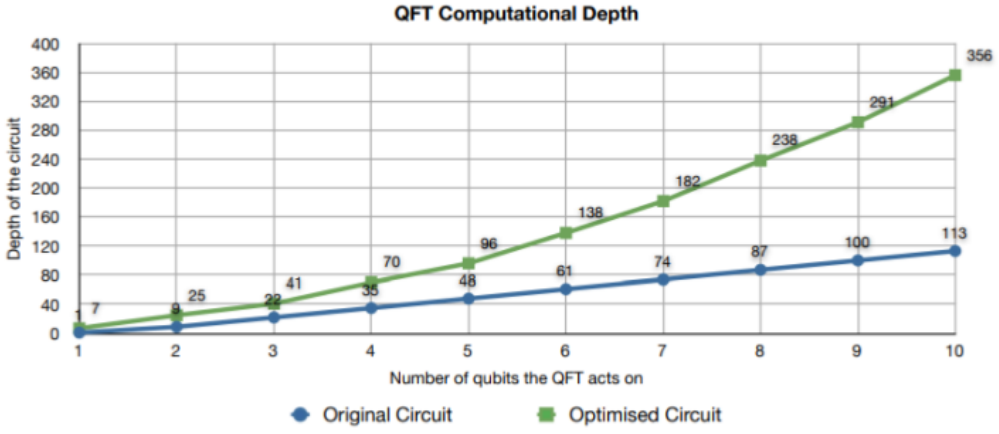
Результати



Результати



Результати



Переваги методу

1. Можливість паралелізувати схему
2. Оптимізація

Дякую за увагу!