浙江大学

量子计算理论基础与软件系统实验报告

课程名称：	量子计算理论基础与软件系统
作业名称：	QFT and QPE Algorithm
姓　　名：	周楠
学　　号：	3220102535
电子邮箱：	3220102535@zju.edu.cn
联系电话：	19858621101
指导教师：	卢丽强

2024 年 10 月 23 日

一. 实验目的和要求

本次实验中，我们使用 ‘qiskit‘ 模拟运行量子电路以进一步理解 QFT 和 QPE 算法的原理。

二. 实验环境

```
conda create -n quantum python=3.10
conda activate quantum
conda deactivate
conda env remove -n quantum
```

三. 实验流程

3.1 QFT 算法代码分析

```
def qft(qc:QuantumCircuit) -> QuantumCircuit:
    for i in range(qc.num_qubits - 1, -1, -1):
        qc.h(i)
        for j in range(i - 1, -1, -1):
            qc.cp(pi / 2 ** (i - j), j, i)
    for i in range(qc.num_qubits // 2):
        qc.swap(i, qc.num_qubits - i - 1)
    return qc
```

该函数接收一个量子电路 qc 作为输入，返回一个经过 QFT 变换后的量子电路。

1. 外层循环：对每个量子比特进行 Hadamard 变换

```
for i in range(qc.num_qubits - 1, -1, -1):
    qc.h(i)
```

这个循环是从最高位（qc.num_qubits - 1）到最低位（0）的量子比特，逐一对每个量子比特 i 应用一个 Hadamard 门（H）。qc.h(i) 表示在量子比特 i 上施加 Hadamard 门，它将量子比特从计算基态变换到叠加态。

2. 内层循环：施加受控相位门

```
for j in range(i - 1, -1, -1):
    qc.cp(pi / 2 ** (i - j), j, i)
```

- 对于每个量子比特 i，对所有比 i 小的量子比特 j，施加一个受控相位门（CP）。
- qc.cp(angle, control_qubit, target_qubit) 的作用是对 control_qubit 和 target_qubit 施加受控相位操作。相位角度 angle 是 $\pi/2^{(i-j)}$。

3. 交换量子比特

```
for i in range(qc.num_qubits // 2):
    qc.swap(i, qc.num_qubits - i - 1)
```

交换操作的目的是将 QFT 的最终结果调整到正确的比特顺序。

3.2 QFT 算法结果

1. 运行代码查看结果态采样的频率分布.

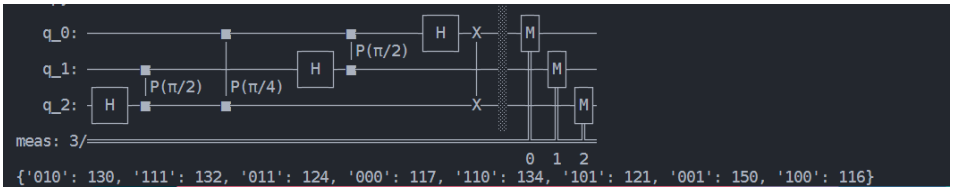

图 1: QFT 算法

量子傅里叶变换（QFT）的定义如下：

$$QFT(|j\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i \cdot jk}{2^n}} |k\rangle$$

对 $|j\rangle = |j_2 j_1 j_0\rangle = |000\rangle$ 应用 QFT。由于 $e^{2\pi i \cdot 0 \cdot k/2^n} = 1$，因此所有的系数都是相等的，变换的结果为：

$$QFT(|000\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i \cdot 0 \cdot k/2^n} |k\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle$$

$$QFT(|000\rangle) = \frac{1}{\sqrt{8}} (|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle)$$

这意味着，应用 QFT 之后，初始态 $|000\rangle$ 变成了所有可能量子态的均匀叠加态。

2. 将 n_qubits 改为 2，初态设置为 $|2\rangle$ 获得双量子比特的 QFT 电路，验证输出与理论结果一致。

```
# TODO: change the number of qubits
n_qubits = 2
qc = QuantumCircuit(n_qubits)
# TODO: add quantum gate to set the initial state
# 对qc进行操作，初态设置为10
qc.x(1)
```

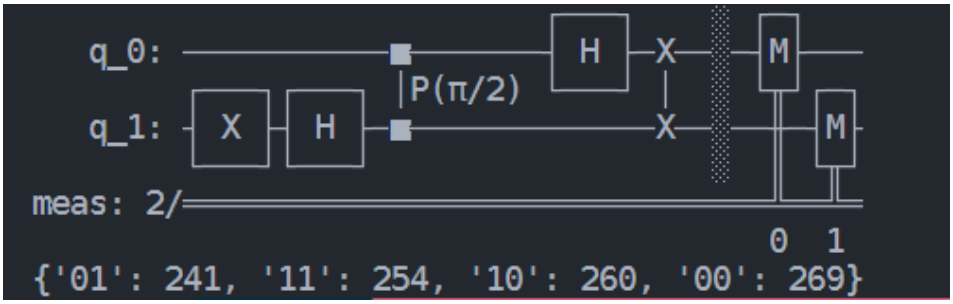

图 2: 双量子比特

得到最终的结果

$$QFT(|10\rangle) = \frac{1}{2}\sum_{k=0}^{3} e^{2\pi i \cdot 2k/4}|k\rangle$$

$$QFT(|10\rangle) = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

3. 通过观察 ‘qft‘ 函数或者理论推导，计算 QFT 算法量子门层数（在所有量子比特上并行操作的一组量子门视为一层）关于量子比特数的复杂度。

- 从 n-1 到 0 依次在每个比特上应用 H 门, 这些 H 门可以并行执行
- 对于第 i 个量子比特：需要与 i-1, i-2, ..., 0 号比特进行 CP 操作。这些 CP 门因为都作用在同一个目标比特 i 上，不能并行
- SWAP 门可以并行执行

总层数 = H 门层数 + CP 门层数 + SWAP 门层数 = 1 + n(n-1)/2 + 1 ∈ $O(n^2)$

3.3 QPE 算法分析

```
from numpy import pi
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
    , transpile
from qiskit.providers.basic_provider import BasicSimulator
from qiskit.circuit.library import QFT

def qft(qc: QuantumCircuit) -> QuantumCircuit:
    for i in range(qc.num_qubits - 1, -1, -1):
        qc.h(i)
        for j in range(i - 1, -1, -1):
            qc.cp(pi / 2 ** (i - j), j, i)
    for i in range(qc.num_qubits // 2):
        qc.swap(i, qc.num_qubits - i - 1)
    return qc

def create_qpe_circuit(n_count: int, theta: float) -> QuantumCircuit:
    # 创建量子和经典寄存器
    qr = QuantumRegister(n_count + 1) # n_count个计数比特 + 1个目标比
        特
    cr = ClassicalRegister(n_count)
    qc = QuantumCircuit(qr, cr)

    # 初始化目标比特为|1
    qc.x(n_count)

    # 对计数寄存器应用H门
    for i in range(n_count):
        qc.h(i)

    # 应用受控P门
    for i in range(n_count):
        for j in range(2**i): # 每个比特位需要重复2^i次
            qc.cp(theta, i, n_count)
```

```
32
33      # 应用逆QFT到计数寄存器
34      qc = qc.compose(QFT(n_count, inverse=True), list(range(n_count)))
35
36      # 测量计数寄存器
37      qc.measure(list(range(n_count)), list(range(n_count)))
38
39      return qc
40
41  # 测试 =2 /3的情况
42  print("\nTesting with  =2 /3:")
43  n_count = 3 # 使用3个计数比特
44  theta = 2*pi/3
45  qc = create_qpe_circuit(n_count, theta)
46  print("Circuit:")
47  print(qc)
48
49  backend = BasicSimulator()
50  tqc = transpile(qc, backend)
51  result = backend.run(tqc, shots=1000).result()
52  counts = result.get_counts()
53  print("\nMeasurement results ( =2 /3):")
54  print(counts)
```

1. 创建量子和经典寄存器

```
1  qr = QuantumRegister(n_count + 1) # n_count个计数比特 + 1个目标
       比特
2  cr = ClassicalRegister(n_count)
3  qc = QuantumCircuit(qr, cr)
```

(a) qr 是一个量子寄存器，包含 n_count 个计数比特和 1 个目标比特。目标比特是用于存储测量相位的特征态。

(b) cr 是一个经典寄存器，用于存储测量的结果。其大小为 n_count，与计数比特的数量一致。

2. 对计数寄存器应用 Hadamard 门

```
for i in range(n_count):
    qc.h(i)
```

对前 n_count 个比特（计数比特）应用 Hadamard 门，将这些比特置于均匀的叠加态。

3. 应用受控 P 门

```
for i in range(n_count):
    for j in range(2**i): # 每个比特位需要重复2^i次
        qc.cp(theta, i, n_count)
```

对于每个计数比特 i，重复应用受控相位门 2^i 次。受控相位门（qc.cp(theta, i, n_count)）表示对目标比特（n_count）施加一个相位旋转，受控于计数比特 i 的状态。

4. 应用逆 QFT 到计数寄存器, 测量计数寄存器

```
    qc = qc.compose(QFT(n_count, inverse=True), list(range(
        n_count)))
    qc.measure(list(range(n_count)), list(range(n_count)))
```

使用 Qiskit 中的 QFT 类，对前 n_count 个比特应用逆 QFT。这一步的目的是将之前编码在计数比特中的相位信息转换为二进制表示形式，以便可以通过测量提取。对前 n_count 个比特进行测量，将测量结果映射到经典寄存器中。每个量子比特的测量结果对应到一个经典比特。

3.4 QPE 实验结果

1. 更改 $P(\theta)$ 门中的角度参数为 $2\pi/3$

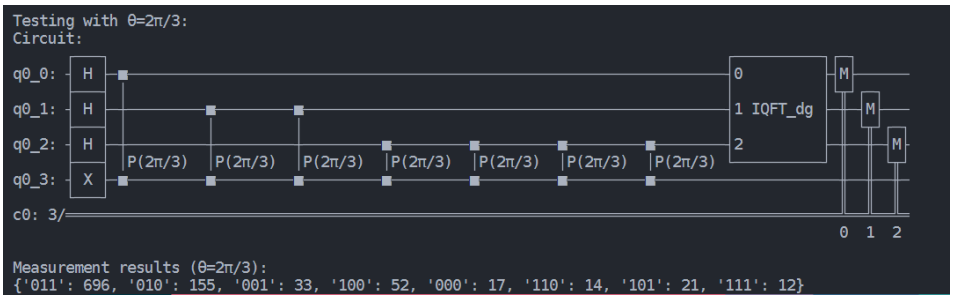

图 3: 更改 $P(\theta)$ 门中的角度参数为 $2\pi/3$

QPE 的输出结果对应于二进制小数形式的相位值。在我们的例子中，$\theta = 2\pi/3$ 对应的相位值为 $\frac{2}{3}$ 使用 3 个计数寄存器比特，我们期望的测量结果应该接近 $\frac{2}{3}$ 的二进制表示（0.101）。

2. 基于 $P(\theta)$ 门计算 QPE 算法量子门层数关于相位估计精度（即计数寄存器中的量子比特数）的复杂度。

   (a) 初始化部分：

   H 门：并行应用在 n 个量子比特上，需要 1 层

   (b) 受控 P 门部分:

   对第 i 个量子比特需要应用 2^i 个受控 P 门

   这些门需要串行执行，因为它们都作用在同一个目标量子比特上

   总层数 $= 2^0 + 2^1 + ... + 2^(n-1) = 2^n - 1$

   (c) 逆 QFT 部分：

   QFT 逆变换操作复杂度为 $O(n^2)$

   总的门层数复杂度是：$1 + (2^n - 1) + O(n)$ 主导项是 2^n 复杂度为 $O(2^n)$

四. 遇到的困难及解决方法

本次实验没有遇到任何困难，一帆风顺。

五. 总结与心得

在本次实验中，我们深入探索了量子计算的核心概念，特别是通过使用 qiskit 这一强大的量子编程框架来模拟运行量子电路，从而进一步加深了对量子傅里叶变换（QFT）和量子相位估计（QPE）算法的理解。

这种从理论到实践的过渡，不仅巩固了我们对 QFT 和 QPE 算法原理的理解，还让我们亲身体验了量子电路的设计与运行过程。例如，我们学会了如何使用 qiskit 中的量子门操作来构建 QFT 电路，并通过模拟运行观察到量子态的变换过程。这种亲自动手的体验，使得抽象的量子理论变得更加具体和生动。