Simulation of Shor's Algorithm

Brennan Saling, C'21
Faculty Mentor: Randolph Peterson
Department of Physics and Astronomy, The University of the South

Abstract

A Quantum Computing simulation of the commonly cited Shor's Algorithm. In order to have a better understanding of the inner workings of a Quantum Computer, it's future applications, and the current and future limitations of classical computing. The simulation uses Shor's Algorithm to do a series of factorizations, which are most commonly associated with cryptography. The quantum component include Qubits, Quantum Fourier Transforms, and Superposition Period Finders.

SEWANEE
THE UNIVERSITY OF THE SOUTH

Classical Components

1. Input a number to be factored referred to as N.
2. Check if N is a prime number, there are many tests that can determine if a given number is prime. The one used for this project is the Miller-Robin primality test. If N is prime number, the algorithm serves no purpose, so the code is finished.
3. Then check if N is a power of a prime number, that is, if there exists a prime number. If so, then then prime number is a factor of N, then we have the factorization of N. And the checking process can continue as planned.
4. Now the number N must be put through a series of calculations of
$$P1^{k1}$$
Until we find a factorization of N.
5. Generate a random number that is smaller than N, this will serve as a guess of sorts..
6. Check if the guess is coprime to N, using the gcd found below, if not, then the greatest common divisor of our guess and N is a factor of N. Then we have a factorization of N.

```
def gcd(g, p):
    if p > g:
        g, p = p, g
    while g > 0:
        g = g % p
        g, p = p, g
    return g


def equ(g, p, n):
    ans = 1
    while (p > 0):
        if p % 2:
            ans = ans * g % n
        p = p // 2
        g = g * g
    return ans
```

Figure 1: Two pieces of code from the Classical Component.

Flow of Information

1. Provide a Number
2. Check if that number is a prime number using Miller-Robin Primality Test
3. Generate a series of exponential numbers that represent the number
4. Generate a random guess
5. Check is guess is coprime to number
6. Use Quantum Phase Estimation
7. Calculate Continued Fraction Decomposition
8. Repeat Quantum Phase Estimation
9. If to much repetition, get a new random guess
10. Receive Factorization

Figure 2: Raman scattering.

Quantum Components

Now we know that x (our guess) and N are coprime to each other. There exists an order of x such that we can determine the full factorization this is where the quantum phase estimation comes in, which can be seen below.
The quantum phase estimation returns a number theta which is equal to a fraction of our guess. in decimal with certain precision. Then a continued fraction decomposition is needed to estimate the exact value of a component of our guess.
Unfortunately, because of the limitation of the continued fraction decomposition for reducible fractions, we may not get a proper component.
If the component is odd, we must try the phase estimation again or try another component.

```
operation InverseFT(qs : Qubit[]) : ()
{
    body {
        let qL = L(qs);
        for (i in 0..(qL - 1)) {
            for (j in 0..(i-1)) {
                (Controlled frac) ([qs[j]], (1, i - j, qs[i]));
            }
            H(qs[i]);
        }

        for (i in 0..qL) {
            Swap(qs[i],qs[qL-1-i]);
        }

    }
```

```
operation PhaseEstimation(
    x : Int,
    N : Int,
    p: Int,
{
    body {
      bit = 0;
      let eL = BS(N);
      using (ev = Qubit[eL]){
        X(ev[0]);
        using (t = Qubit[p]){
          PhaseEstimationImpl(x, N, t, ev);
            for (i in 0..(p - 1)) {
              set bit = bit * 2;
              if (M(t[i]) == One) {
                set bit = bit + 1;
            }
          }
        }
    }
```

```
operation PhaseEstimationImpl (
    x : Int,
    N : Int,
    t : Qubit[],
    ev : Qubit[]) : ()
{
    body {
      for (idx in 0..(tL - 1)) {
          H(t[idx]);
      }
      mutable p = 1;
      for (idx in 0..(tL - 1)) {
        (Controlled ConstructU) ([t[t - 1 -idx]], (x, N, p, ev));
          set p = p * 2;
      }
      (InverseFT)(t);
      (Adjoint QFT)(B(t));
    }
}
```

Figure 3: Three pieces of code from the Quantum Component.

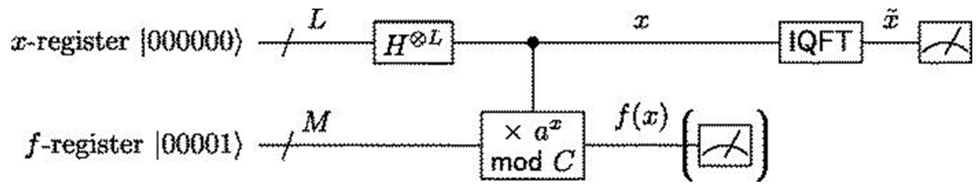

Figure 4: What an actual Phase Estimation circuit would look like in a physical quantum computer .

Background

- Shor's algorithm is a quantum computer algorithm for integer factorization. If given an integer it can find its prime factors. Invented by Peter Shor in 1994. It is a key piece in the future of cryptography.
- What makes this algorithm so much better than all the classical computer algorithm that can factor a given number is the speed at which a quantum computer can find the prime factors of numbers with dozens of digits compared to classical computers. What can take a quantum computer minutes can take a Classical computer a lifetime. It does this through quantum superposition, which is why the phase estimation in my code is often referred to as Superposition Period Finder. It can go through numerous possible answers at the same time, when a classical computer can only go through one at a time. This can be worrisome when you realize many of the security systems on your computer use this large numbers as keys to many important bits of information. Therefore Shor's Algorithm is considered one of the more applicable quantum computer algorithm's to date.

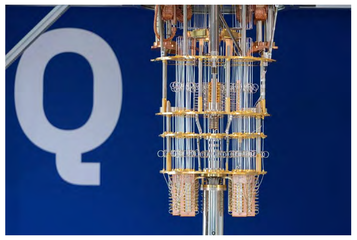

Figure 5: Pictures of two physical IBM Quantum Computers.

A Note of Stabilizing Circuits

- Stabilizing Circuits are a part of the quantum error correction component of Quantum Computers. They plays an important role in the practical use and engineering of quantum computers and quantum communication devices, and while they are important to physical quantum computers that had no use in my code for the quantum computer. This is because these circuits are built to reduce quantum noise which we have none of in a classical computer but they are an important part of Quantum Computing and therefore important to note when speaking of simulations.

Acknowledgements

The author would like to thank the following for their continued support and guidance throughout the semester:

- Faculty of the Physics Department
- Dr. Peterson for his helpful advice during the construction of this project.
- Junior and Senior Physics Majors
- Friends and Family