Project Report

Introduction to Parallel Scientific Computing

Implement/Simulate a Quantum Algorithm using Microsoft Quantum Development Kit

Ankit Pant – 2018201035
Tarun Mohandas – 2018201008

Abstract

Computers have come along way since the first theoretical constructs of Alan Turing – the Turing Machines. The quest to get faster and efficient have led to rapid developments in computers. A new paradigm of computing was proposed with the developments in Quantum Mechanics. Hence was born "Quantum Computers" which today are no longer mathematical constructs on paper. Though quantum computing still has a long way to go before being ubiquitous, using the postulates of Quantum Mechanics, several algorithm have already been proposed that make use of quantum parallelisation to dramatically reduce (often exponentially) the runtime required when compared to running similar algorithms on a classical computers. This project explores one such quantum algorithm – Shor's Algorithm. This algorithm is involved with the factorisation of integers which conventionally requires a very long runtime on classical computers. This runtime to compute the factors of an integer can be reduced significantly using Shor's Algorithm on a quantum computer. This project implements and simulates Shor's Algorithm on a classical computer using Microsoft's Q# language and Microsoft Q# Development Kit.
keywords: *quantum computers, quantum algorithms, Shor's Algorithm, Q#*

Acknowledgement

We express our gratitude to the course instructor Dr. Pawan Kumar for allowing us to proceed with this project. It was a good learning experience and has added to our knowledge and would enable us to explore further in the domain of Quantum Computing.

Contents

List of Figures

1 Introduction

With some strong theoretical foundations of Quantum Mechanics established around the early 1980s, the concept of Quantum Computing was proposed by Richard Feynman and Yuri Manin [1]. Loosely speaking Quantum Computers are promising as they seek to use nature as a processing device. Using the, often counter-intuitive, postulates and principles of Quantum Mechanics, a system was thought of that could be exponentially faster than any classical computer. However realising a physical Quantum Computer has been very challenging from an engineering standpoint and to this day a commercially viable Quantum Computer still seems elusive. Much of it attributed to the extreme physical conditions (low temperatures, vacuum conditions, etc.) that are needed for a quantum computer to work as intended. Owing to the engineering challenges, as well as the challenges, the development of quantum computers remained slow during the initial years. However in 1994, Peter Shor proposed an algorithm that could factor an integer in a time that was much faster than the classical algorithms. This was monumental since Internet was starting to boom during that time and much of the security of the Internet and the World Wide Web depended on the computational hardness of the Integer Factorisation problem. This invigorated the interest in quantum mechanics and revived the research in this area. Currently basic quantum computers are available today and new developments are being done at a rapid pace with even governments acting as sake-holders to establish 'quantum supramacy'.
This project involves implementing and simulating Shor's Algorithm. The algorithm is implemented in Microsoft's Q# programming language and used Microsoft Quantum Development kit as the development environment. The algorithm was then, using a classical computer (laptop), simulated to factor integers. The following sections elaborate on quantum computing and Shor's algorithm followed by the implementation and the corresponding results. Following which is a section containing the conclusions and a section containing the future scope.

2 Literature Review

The following sub-sections elaborate briefly the theoretical aspects of quantum computing as well as Shor's algorithm in general.

2.1 Quantum Computing - Introduction

Quantum computing is the area of study focused on developing computer technology based on the principles of quantum theory, which explains the nature and behaviour of energy and matter on the quantum (atomic and subatomic) level. Development of a quantum computer, if practical, would mark a leap forward in computing capability far greater than that from the abacus to a modern day supercomputer, with performance gains in the billion-fold realm and beyond. The quantum computer, following the laws of quantum physics, would gain enormous processing power through the ability to be in multiple states, and to perform tasks using all possible permutations simultaneously.

2.2 Qubit

The bit is the fundamental concept of classical computation and classical information. Quantum computation and quantum information are built upon an analogous concept, the quantum bit, or qubit for short. Just as a classical bit has a state – either 0 or 1 – a qubit also has a state. Two possible states for a qubit are the states $|0\rangle$ and $|1\rangle$

The difference between bits and qubits is that a qubit can be in a state other than $|0\rangle$ or $|1\rangle$. It is also possible to form linear combinations of states, often called super-positions:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

where α and β are complex constants.

2.3 Quantum Gates

A quantum logic gate is a basic quantum circuit operating on a small number of qubits. The following are some of the most common quantum gates.

2.3.1 CNOT Gate

The CNOT gate flips the second qubit (the target qubit) if and only if the first qubit (the control qubit) is $|1\rangle$. The following is the matrix transformation for the CNOT gate.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

2.3.2 Hadamard Gate

The Hadamard gate acts on a single qubit. It maps the basis state to $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ which means that a measurement will have equal probabilities to become 1 or 0 (i.e. creates a superposition). The following is the matrix transformation:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

2.3.3 Pauli-X gate

The Pauli-X gate acts on a single qubit. It is the quantum equivalent of the NOT gate. The following is the matrix transformation of Pauli-X gate.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

2.3.4 Pauli-Y gate

The Pauli-Y gate acts on a single qubit. It equates to a rotation around the Y-axis of the Bloch sphere by π radians.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

2.3.5 Pauli-Z gate

The Pauli-Z gate acts on a single qubit. It equates to a rotation around the Z-axis of the Bloch sphere by π radians.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

2.4 Quantum Fourier Transform

The quantum Fourier transform is the classical discrete Fourier transform applied to the vector of amplitudes of a quantum state, where we usually consider vectors of length $N = 2^n$. The quantum Fourier transform acts on a quantum state $|x\rangle = \sum_{i=0}^{N-1} x_i |i\rangle$ and maps it to a quantum state $\sum_{i=0}^{N-1} y_i |i\rangle$ according to the formula:

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \omega_n^{jk},\ k = 0, 1, 2, ...N-1$$

If $|x\rangle$ is a basis state, the QFT can be expressed as the map,

$$QFT = |x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_n^{xk} |k\rangle$$

The quantum Fourier transform can be viewed as a unitary matrix acting on quantum state vectors, where the unitary matrix F_N is given by

$$F_N = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & 1 & ... & 1 \\ 1 & \omega & \omega^2 & \omega^3 & ... & \omega^{N-1} \\ 1 & \omega^2 & \omega^3 & \omega^4 & ... & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^4 & \omega^5 & ... & \omega^{3(N-1)} \\ 1 & \omega^4 & \omega^5 & \omega^6 & ... & \omega^{4(N-1)} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & ... & \omega^{(N-1)(N-1)} \end{bmatrix}$$

2.5 Shor's Algorithm

2.5.1 The algorithm

Shor's algorithm is a quantum algorithm for integer factorization, that solves the following problem: Given an integer N, find its prime factors.

1. Use a polynomial algorithm to determine if N is prime or a power of prime. If it is a prime, declare that it is and exit. If it is a power of a prime number, declare that it is and exit.

2. Randomly choose an integer a such that $1 < a < N$. Perform Euclid's algorithm to determine $GCD(a, N)$. If the GCD is not 1, then return it and exit.

3. Use quantum circuit to find a period r.

4. If r is odd or if $a^r \equiv -1\ Mod\ N$, then return to Step 2 and choose another a.

5. Use Euclid's algorithm to calculate $GCD(a^{\frac{r}{2}}+1, N)$ and $GCD(a^{\frac{r}{2}}-1, N)$. Return atleast one of non-trivial solutions.

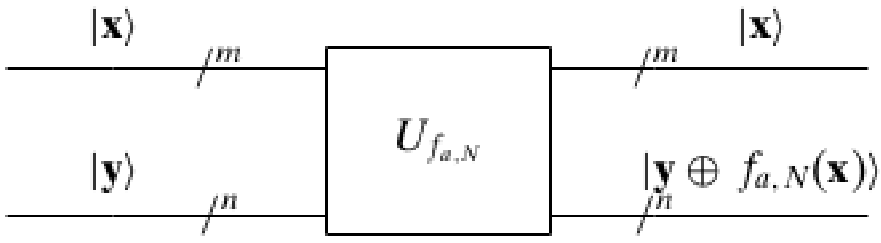
2.5.2 Quantum Part

1. We have to show that there is a quantum circuit that can implement the function $f_{a,N}$. The output of this function will always be less than N, and so we will need $n = log_2 N$ output bits. We will need to evaluate $f_{a,N}$ for at least the first N^2 values of x and so will need at least

$$m = log\, N^2 = 2\, log\, N = 2n$$

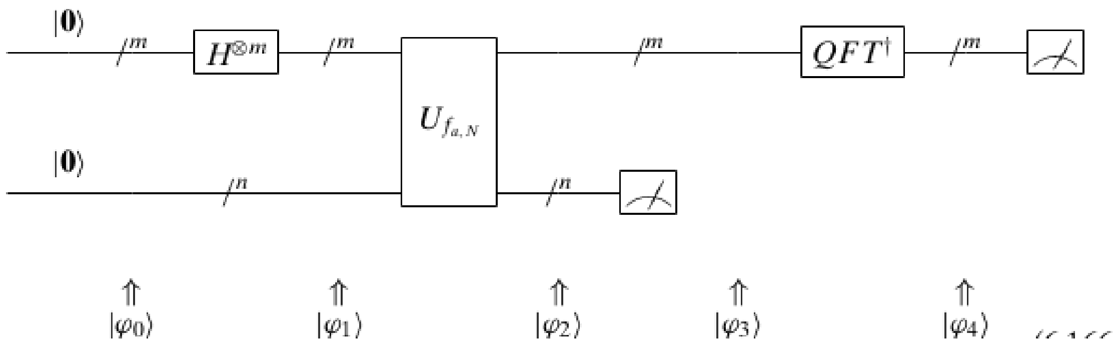
input qubits.

2. The quantum circuit that we would get will be the operator $U_{f_{a,N}}$, which we may visualize as



where $|x, y\rangle$ goes to $|x, y \ominus f_{a,N}\rangle = |x, y \bigoplus a^x mod\, N\rangle$

3. The following is the quantum circuit for the Shor's algorithm



4. The states of the system is given as follows:

$$\phi_0 = |0_m, 0_n\rangle$$

$$\phi_1 = \frac{\sum_{x\in\{0,1\}} |x, 0_n\rangle}{\sqrt{2^m}}$$

$$\phi_2 = \frac{\sum_{x\in\{0,1\}} |x, f_{a,N}(x)\rangle}{\sqrt{2^m}} = \frac{\sum_{x\in\{0,1\}} |x, a^x modN\rangle}{\sqrt{2^m}}$$

5. We measure the bottom qubits of $|\phi_2\rangle$which is in a superposition of many states. Let us say that after measuring the bottom qubits we find $a^{\overline{x}} modN$ for some $\overline{x}$. However by periodicity of $f_{a,N}$ we also have, $a^{\overline{x}} \equiv a^{\overline{x}+r} mod\, N$ and $a^{\overline{x}} \equiv a^{\overline{x}+2r} mod\, N$. For any $s \in Z$, we have, $a^{\overline{x}} \equiv a^{\overline{x}+sr} mod\, N$.

3 Methodology

The implementation of Shor's Algorithm using Microsoft Quantum Development Kit involved using two files.

- The quantum part of the algorithm (period finding, etc.) was written in Q# programming language
- The driver program along with the classical part of the algorithm was written in C# programming language

The project was then run using the .NET framework. The project was initialised using the following code:

```
$ dotnet new console -lang Q# --output Quantum_Factorization
```

The project was executed using the command (in the project directory):

```
$ dotnet run
```

The following sections elaborate on the quantum and classical part of the implemented algorithm. Since the quantum part of the algorithm acts as a subroutine to the classical part of the algorithm, when the classical part of the algorithm is elaborated, appropriate subroutine call is made.

3.1 Quantum Subroutine

The pseudo-code for the quantum part of the algorithm is given as:

Algorithm 1 Calculate period of the generator(a)

```
Require: $generator(a), number(N)$
Ensure: $a$ and $N$ are co-primes i.e. $GCD(a, N) = 1$
Output: Period $r$ of the generator
  Set number of qubits for register (qsize) as $\log_2(N)$
  Set number of bits for precision as $2 * qsize + 1$
  Set result = 1
  repeat
    Initialise quantum register (Q) (array) of size $qsize$ and set to appropriate machine
    architecture (Little-Endian)
    $Q \leftarrow ((a^r modN) * Q) modN$
    Quantum Phase Estimation (Q)
    Approximate $\frac{s}{r}$ using continued fraction convergence
    $r \leftarrow abs(denominator)$ (denominator= approximated r)
    $result \leftarrow (r * result)/GCD(result, r)$
  until $(a^r modN == 1)$
  return result
```

3.2 Classical Algorithm

The pseudo-code for the classical part of the algorithm is given as:

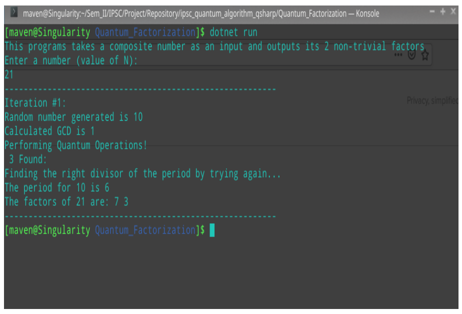
Algorithm 2 Calculate non-trivial factors of number: N

```
Require: Input composite integer N
Output: Two non-trivial factors of N
  Input number $N$
  if $n \,\%\, 2 == 0$ then
    $factor1 \leftarrow 2$
    $factor2 \leftarrow (N/2)$
    return (factor1, factor2)
  else
    $iter \leftarrow 0$
    for $iter <= (N/2)$ do
      $a \leftarrow RandomINT(2, N)$
      $gcd = GCD(a, N)$
      if $gcd > 1$ then
        $factor1 \leftarrow gcd$
        $factor2 \leftarrow (N/gcd)$
        return (factor1, factor2)
      else
        if $(a^2 \,\%\, N \neq 1)$ $AND$ $(a \,\%\, N \neq 1)$ $AND$ $(a \,\%\, N \neq -1)$ then
          Call Quantum subroutine to find period
          if $period \,\%\, 2 \neq 0$ then
            Print: *This iteration of Shor's algorithm failed. Trying next iteration*
            Try next iteration
          else
            $a_r \leftarrow a^{\frac{period}{2}} \,\%\, N$
            if $a_r \neq (N-1)$ then
              $factorcandidate1 \leftarrow GCD(a_r - 1, N)$
              $factorcandidate2 \leftarrow GCD(a_r + 1, N)$
              $factor1 \leftarrow Max(factorcandidate1, factorcandidate2)$
              $factor2 \leftarrow (N/factor1)$
              return (factor1, factor2)
            else
              Print: *This iteration of Shor's algorithm failed. Trying next iteration*
              Try next iteration
            end if
          end if
        else
          Print: *This iteration of Shor's algorithm failed. Trying next iteration*
          Try next iteration
        end if
      end if
    end for
  end if
```

The input provided was a small composite number (due to hardware limitations) and the output received was the two non-trivial factors of the input number.

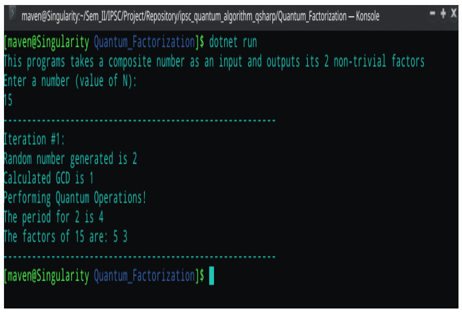
4 Results

The program was run on Microsoft Q# Development Kit's Simulator and the algorithm successfully factorized numbers (albeit small) into their respective two non-trivial factors. The output of the program can be seen in the following diagrams:



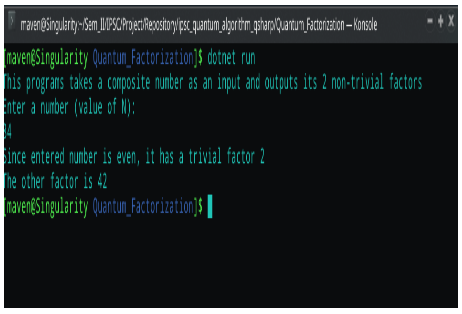
```
maven@Singularity:~/Sem_II/IPSC/Project/Repository/ipsc_quantum_algorithm_qsharp/Quantum_Factorization — Konsole
[maven@Singularity Quantum_Factorization]$ dotnet run
This programs takes a composite number as an input and outputs its 2 non-trivial factors
Enter a number (value of N):
21
--------------------------------------------------------
Iteration #1:
Random number generated is 10
Calculated GCD is 1
Performing Quantum Operations!
 3 Found:
Finding the right divisor of the period by trying again...
The period for 10 is 6
The factors of 21 are: 7 3
--------------------------------------------------------
[maven@Singularity Quantum_Factorization]$
```

Figure 1: Factorization of Number 21



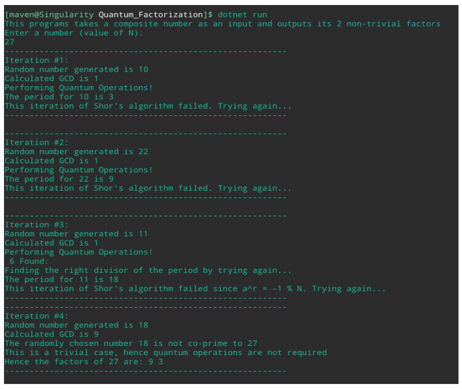
```
maven@Singularity:~/Sem_II/IPSC/Project/Repository/ipsc_quantum_algorithm_qsharp/Quantum_Factorization — Konsole
[maven@Singularity Quantum_Factorization]$ dotnet run
This programs takes a composite number as an input and outputs its 2 non-trivial factors
Enter a number (value of N):
15
--------------------------------------------------------
Iteration #1:
Random number generated is 2
Calculated GCD is 1
Performing Quantum Operations!
The period for 2 is 4
The factors of 15 are: 5 3
--------------------------------------------------------
[maven@Singularity Quantum_Factorization]$
```

Figure 2: Factorization of Number 15



```
maven@Singularity:~/Sem_II/IPSC/Project/Repository/ipsc_quantum_algorithm_qsharp/Quantum_Factorization — Konsole
[maven@Singularity Quantum_Factorization]$ dotnet run
This programs takes a composite number as an input and outputs its 2 non-trivial factors
Enter a number (value of N):
84
Since entered number is even, it has a trivial factor 2
The other factor is 42
[maven@Singularity Quantum_Factorization]$
```

Figure 3: Factorization of Number 84



```
[maven@Singularity Quantum_Factorization]$ dotnet run
This programs takes a composite number as an input and outputs its 2 non-trivial factors
Enter a number (value of N):
27
---------------------------------------------------------
Iteration #1:
Random number generated is 10
Calculated GCD is 1
Performing Quantum Operations!
The period for 10 is 3
This iteration of Shor's algorithm failed. Trying again...
---------------------------------------------------------

---------------------------------------------------------
Iteration #2:
Random number generated is 22
Calculated GCD is 1
Performing Quantum Operations!
The period for 22 is 9
This iteration of Shor's algorithm failed. Trying again...
---------------------------------------------------------

---------------------------------------------------------
Iteration #3:
Random number generated is 11
Calculated GCD is 1
Performing Quantum Operations!
 6 Found:
Finding the right divisor of the period by trying again...
The period for 11 is 18
This iteration of Shor's algorithm failed since a^r = -1 % N. Trying again...
---------------------------------------------------------
---------------------------------------------------------
Iteration #4:
Random number generated is 18
Calculated GCD is 9
The randomly chosen number 18 is not co-prime to 27
This is a trivial case, hence quantum operations are not required
Hence the factors of 27 are: 9 3
---------------------------------------------------------
```

Figure 4: Factorization of Number 27

It can be seen that the numbers which were factored and the algorithm gave two non-trivial factors as the output.

5 Conclusion

Quantum Computers if successfully commercialised would dramatically change the way we compute. Not only will it provide greater security over the Internet (using Quantum Cryptography) but also would be able to exploit quantum parallelisation to solve certain types of problems exponentially faster than a classical computer.
The project successfully implemented and simulated Shor's algorithm on a Quantum Simulation in a classical machine. Since the code was executed in the classical machine the same expected amount of speed-up was not observed as would be if the algorithm was executed on an actual quantum computer. Also since simulating quantum bits (qubits) on a classical computer requires a lot of memory (30 qubits simulation requires 16 GB RAM while a 40 qubit simulation requires around 16000 GB of RAM), small integers were chosen as inputs to factorize (due to limitation of classical hardware). However (despite being slow), the algorithm correctly factorizes the integer and given powerful enough hardware (or an actual quantum computer) can be scaled to factorizing larger numbers.

6 Future Scope

Given the time, effort, and capital being put into developing quantum computers, more powerful quantum computers are expected to be built in the near future. This project then can be extended to run the algorithm on an actual quantum computer hence achieving the desired speed-up.

References

[1] Quantum computing – Wikipedia the free encyclopedia, https://en.wikipedia.org/wiki/Quantum_computing

[2] Quantum Computing for Computer Scientists, Noson S. Yanofsky and Mirco A. Mannucci