Project Progress Report

Introduction to Parallel Scientific Computing

Implement/Simulate a Quantum Algorithm using

Microsoft Quantum Development Kit

Ankit Pant – 2018201035
Tarun Mohandas – 2018201008

Submitted on: 2 April 2019

Abstract

The project involves implementing/simulating a quantum algorithm using Microsoft Quantum Development Kit. Quantum Computing is an exciting area of research with the promise of changing the computing world dramatically. Quantum Algorithms use properties of quantum system (e.g. superposition) and can in principle run exponentially faster than their classical counterparts.
The project involves getting familiar with Microsoft's Quantum Development Kit and use it to implement a Quantum Algorithm. The working environment will be Linux.
keywords: *quantum algorithm, Quantum Development Kit, computing*

Contents

1 Progress Summary

Implementing a quantum algorithm on Microsoft Quantum Development Kit involves the following pre-requisites:

- A quick revision of the basic concepts of Quantum Mechanics
- Understanding basics of quantum computing and quantum gates
- Familiarization with Microsoft Q# Development Kit
- Understanding basic quantum algorithms

The progress made so far is summarised in the subsections that follow.

1.1 Studying basic Quantum Mechanics concepts

A quick revision of the basic concepts of quantum mechanics like superposition, entanglement, etc. was studied. It also involved revision of Linear Algebra (Vectors, Matrices, Tensors, etc.) [1].

1.2 Studying basic quantum Gates

Quantum gates, like the transistor gates of the classical computers, are vital to quantum computing. It is imperative to have a basic understanding of quantum gates and quantum circuits to be able to effectively implement quantum algorithms. Some of the quantum gates explored were [1]:

- **Hadamard Gate:** It applies the Hadamard Transformation to a single qubit. The Hadamard Transformation is defined as:

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

- **CNOT:** It applies a controlled not to a pair of qubits and creates a maximally entangled two-qubit state. It is defined as:

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

1.3 Setting up Q# Environment

The Q# environment was set up in Linux. The following steps were involved:

1. Installing the .NET sdk and runtime [2]
2. Installing IQ# : It was done using the command:

```
dotnet tool install -g Microsoft.Quantum.IQSharp
```

3. Installing Q# templates: It was done using the command:

   ```
   dotnet new -i Microsoft.Quantum.ProjectTemplates
   ```

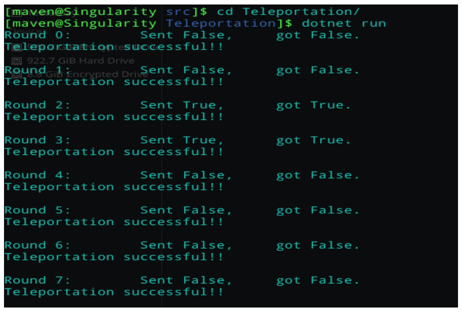
4. Installing Q# kernel for python and jupyter notebook interfacing (Optional). It was done using the command:

   ```
   dotnet iqsharp install
   ```

   Note: At the present time there is some issue with the installation of the Q# kernel for python in Manjaro Linux and hence was not possible. Hopefully it will be resolved soon.

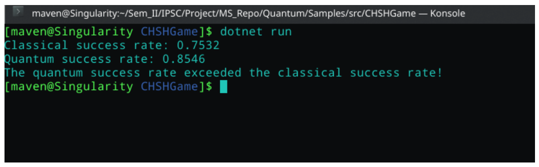
1.4 Running Sample Programs

Some sample programs (provided by Microsoft) were run to ensure that the environment is set up correctly. The following are the screenshots of the execution of Sample Quantum programs.



```
[maven@Singularity src]$ cd Teleportation/
[maven@Singularity Teleportation]$ dotnet run
Round 0:         Sent False,      got False.
Teleportation successful!!

Round 1:         Sent False,      got False.
Teleportation successful!!

Round 2:         Sent True,       got True.
Teleportation successful!!

Round 3:         Sent True,       got True.
Teleportation successful!!

Round 4:         Sent False,      got False.
Teleportation successful!!

Round 5:         Sent False,      got False.
Teleportation successful!!

Round 6:         Sent False,      got False.
Teleportation successful!!

Round 7:         Sent False,      got False.
Teleportation successful!!
```

Figure 1: Teleportation Sample

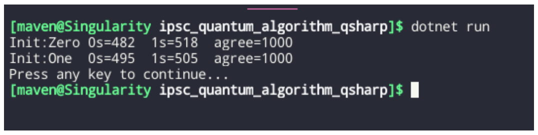


```
maven@Singularity:~/Sem_II/IPSC/Project/MS_Repo/Quantum/Samples/src/CHSHGame — Konsole
[maven@Singularity CHSHGame]$ dotnet run
Classical success rate: 0.7532
Quantum success rate: 0.8546
The quantum success rate exceeded the classical success rate!
[maven@Singularity CHSHGame]$
```

Figure 2: Classical vs Quantum Sample

1.5 Implementing Basic Quantum Programs

Two basic Quantum Programs were implemented. One was the basic ”Hello World” program and the other was entangling two qubits and performing measurements on them. The code repository can be found here (**Note:** You need to be logged into GitHub to be able to view the page. The code is in *test_Q#_environment* branch). The output of the entanglement program was:



```
[maven@Singularity ipsc_quantum_algorithm_qsharp]$ dotnet run
Init:Zero 0s=482  1s=518  agree=1000
Init:One  0s=495  1s=505  agree=1000
Press any key to continue...
[maven@Singularity ipsc_quantum_algorithm_qsharp]$ 
```

Figure 3: Entanglement program

2 Future Work

The following things are being worked on towards the completion of the project:

1. Exploring and studying various quantum algorithms
2. Selecting a quantum algorithm to implement
3. Fix issue with qsharp kernel for python to be able to use jupyter notebooks to present the code interactively

As the team makes progress with the project, additional things would be added/implemented.

References

[1] Quantum Computing Concepts – Microsoft Quantum Development Kit Preview, https://docs.microsoft.com/en-us/quantum/concepts/?view=qsharp-preview

[2] Microsoft .NET, https://dotnet.microsoft.com/