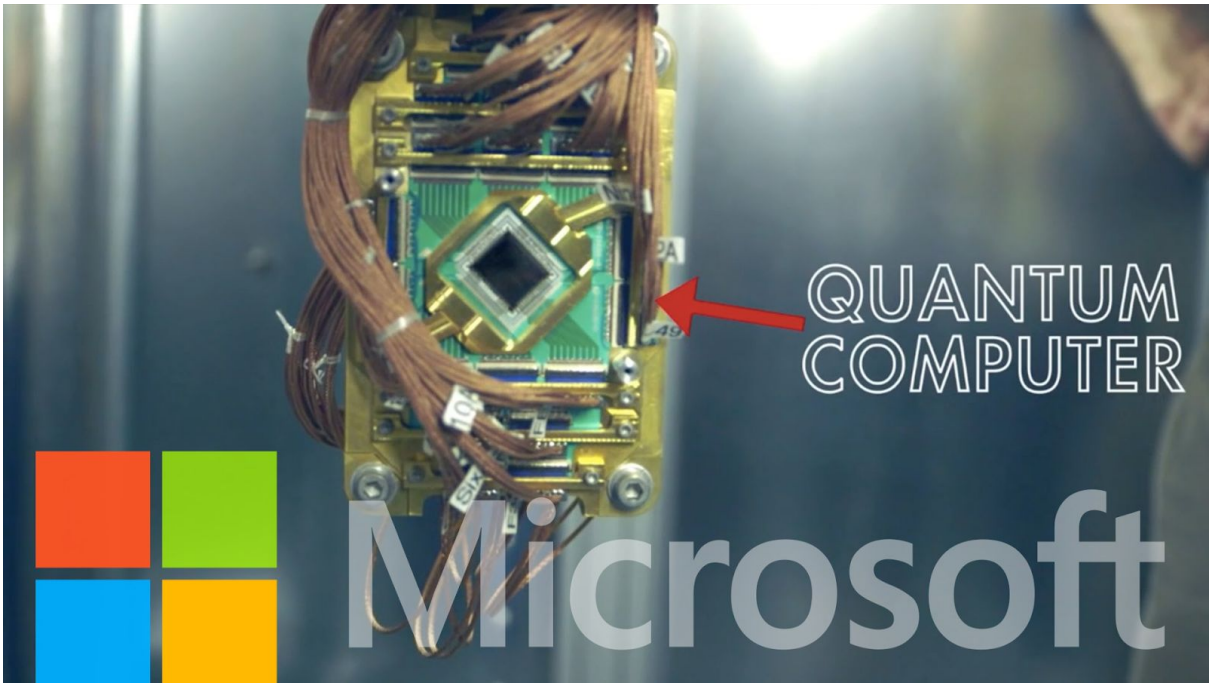
Group 37
Microsoft - Quantum Computing



Title Requirement:

Learn concepts of quantum computing. Bring your idea, create a simulation using Microsoft Quantum Development Kit with support from our mentors

Team Members:

Eoin Dowling - dowlineo@tcd.ie
Patrick Jennings - jenninpa@tcd.ie
Prapti Setty - settyp@tcd.ie
Isaac Walker - walkeri@tcd.ie
Daniel Cosgrove - cosgroda@tcd.ie
David Phillips - phillida@tcd.ie
Jingchen Zhang - zhangj5@tcd.ie

Client:

Microsoft
Anna Koloskova <annak@microsoft.com>

Mentors:

Paddy Healy : pathea@microsoft.com
Muiris Woulfe : mwoulfe@microsoft.com

TABLE OF CONTENTS

I. Introduction

i. Overview and Purpose of System

“Quantum computing is computing using quantum-mechanical phenomena, such as superposition and entanglement. A quantum computer is a device that performs quantum computing.

The field of quantum computing was initiated by the work of Paul Benioff (de) and Yuri Manin in 1980, Richard Feynman in 1982, and David Deutsch in 1985. A quantum computer with spins as quantum bits was also formulated for use as a quantum spacetime in 1968.

As of 2018, the development of actual quantum computers is still in its infancy, but experiments have been carried out in which quantum computational operations were executed on a very small number of quantum bits.”
https://en.wikipedia.org/wiki/Quantum_computing

With this in mind Microsoft set us the challenge of learning about Quantum Computing, bringing our ideas to the table, and then testing their recently released Quantum Computing Development Kit. This development kit includes what a developer would need to get started in the field: a Q# language and compiler along with a Q# library, a local quantum computing simulator, and a Visual Studio extension. The aim is to test whether with the lack of hardware, the software provided accurately simulates how a quantum computer would work.

ii. Scope

The scope of our project was potentially very large. Our clients’ main concern was for us to become familiar with their development kit and find some way to test it. One of our members looked into the promising idea of building a program to allow virtual quantum circuit design, with Q# as a backend generating the circuit output. While we were all behind this idea, after drawing up diagrams of how we would implement this and pitching it to the Microsoft team, our clients felt that this task strayed too far from what they were expecting (ie: we would spend too much time on the GUI as opposed to using Q#), and so rejected this idea.
After meeting up and discussing the various possibilities, we settled on implementing a chosen quantum algorithm in Q#, as this is itself a difficult problem which will take most of the term to complete.

iii. Objectives and Success criteria

The main objective, along with testing the software provided would be to decide whether the tool released adequately simulates how a quantum computer would work. The first task to achieve this would be to learn the concepts of quantum computing. To do this, after downloading the Visual Studio Extensions required, each member of the team was expected to complete the tutorial on how to use the Development Kit and Run a Program in their own time. The team them met up for a team meeting to help those who could be struggling with getting set up with the Q#.

There are several ways we might evaluate the effectiveness of the simulation. Quantum computer machines deliver the possibility of true randomness. The team aims to start by looking into this and possibly test whether we believe the software adequately simulates true randomness.

After discussing with the client, it was decided that our first method of evaluation should be to implement a theoretical “quantum algorithm” (ie: an algorithm which uses the properties of qubits and quantum mechanics) using Q#.
We have agreed to implement the Triangle Finding algorithm. To test the software by implementing this algorithm, we plan to implement a traditional computational algorithm(implementation of traditional algorithm found at https://www.geeksforgeeks.org/number-of-triangles-in-a-undirected-graph/) to solve the triangle finding problem and then compare this to our quantum algorithm.

iv. Definitions and Abbreviations

Triangle-finding problem

*“*The triangle-finding problem is the problem of determining whether a given graph contains a triangle (a clique of size 3). The best-known lower bound for quantum algorithms is $\Omega(N)$, but the best algorithm known requires $O(N^{1.297})$ queries, an improvement over the previous best $O(N^{1.3})$ queries.” - https://en.wikipedia.org/wiki/Quantum_algorithm#Triangle-finding_problem

“When the graph does contain a triangle, algorithms are often required to output three vertices which form a triangle in the graph.

It is possible to test whether a graph with m edges is triangle-free in time $O(m^{1.41})$. Another approach is to find the trace of A^3, where A is the adjacency matrix of the graph. The trace is zero if and only if the graph is triangle-free. For dense graphs, it is more efficient to use this simple algorithm which relies on matrix multiplication, since it gets the time complexity down to $O(n^{2.373})$, where n is the number of vertices.” - https://en.wikipedia.org/wiki/Triangle-free_graph#Triangle_finding_problem

v. Bibliographies

Wikipedia Articles like the following:
https://en.wikipedia.org/wiki/Quantum_computing
https://en.wikipedia.org/wiki/Quantum_algorithm

Microsoft Articles and help Demos:
https://docs.microsoft.com/en-us/quantum/quantum-installconfig?view=qsharp-preview

Quantum Algorithm for Triangle Finding in Sparse Graphs pdf

https://arxiv.org/pdf/1507.06878.pdf

Quantum Computing (Jozef Gruska, 1999) book : https://drive.google.com/file/d/1eO5CwDjXOnFrJQM6elg7Mhm2V9ra2hYL/view?usp=sharing

II. Current System

The current system is the tool provided by Microsoft. A github repository exists that can be cloned, with programs on a few Simple Algorithms. This repository gives us access to a few useful Quantum subroutines we could use in our system.
Repository: https://github.com/Microsoft/Quantum.git

III. Proposed System

i. Overview

The first step of the proposed system was to learn the concepts of quantum computing. As outlined above, we started by downloading the extension to Visual Studio and then going through the tutorial. The team then met with Microsoft on the 6th of February 2018 in the Microsoft building in Leopardstown Business Park. At this point, it was decided by the mentors and the team that the best focus would be to try to implement a few algorithms in the Q sharp language.

With this in mind, the team took a day to decide on the first algorithm they would implement. What was initially challenging for the members of the team was that there is not much documentation about the language, and a lot of the algorithms are “hypothetical.” With this in mind, we decided to explore the Simple Algorithms program given as a sample to try to understand what we would be capable of testing. Simple algorithms the team explored:

- Deutsch–Jozsa Quantum Algorithm
- Bernstein–Vazirani Fourier Sampling Quantum Algorithm
- Quantum Algorithm for Hidden Shifts of Bent Functions

After this point, the team decided that the algorithm to implement would be the **Quantum Algorithm for the Triangle Finding Problem**.

ii. Functional Requirements

“In Software engineering, a **functional requirement** defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define *what* a system is supposed to accomplish.” - https://en.wikipedia.org/wiki/Functional_requirement

The main functional requirement set to us by Microsoft is to test the software and the best method for this is to learn Q#, research an algorithm and then implement it. The chosen algorithm is the Quantum algorithm for finding triangles in a sparse graph.

The input for this algorithm is the adjacency matrix for the graph, the calculations required will be calculated as the algorithm is implemented. The output will be the three vertices of the triangle found. Our project will allow for the user to enter an undirected unweighted graph and assign edges between vertices. An adjacency matrix of the graph will be generated and inputted into Q#. Q# will then find a triangle if present and return it's three vertices. The result of the call to Q# will then be displayed visually in C#.

iii. Non- Functional Requirements

"In systems engineering, a **non-functional requirement** (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions."- https://en.wikipedia.org/wiki/Non-functional_requirement

The non-functional requirements of this project are sparse. The method used to judge the operation of the system will be whether we are able to implement the algorithm or not. The team here in Trinity will provide a short simple report of their explorations of the algorithm and outlining whether the quantum algorithm implementation is better than the traditional implementation.

iv. System Prototype

"A **prototype** is an early sample, model, or release of a product built to test a concept or process or to act as a thing to be replicated or learned from. It is a term used in a variety of contexts, including semantics, design, electronics, and software programming" - https://en.wikipedia.org/wiki/Prototype

The prototype is pre-existent in the form of the Q# language and development tool. Therefore, due to the nature of our project, i.e testing the software tool provided, there will be no system prototype produced by the team, except the code produced by the team while implementing the algorithm.

Signed:

_____

Anna Koloskova : annak@microsoft.com

_____

Paddy Healy : pathea@microsoft.com

_____

Muiris Woulfe : mwoulfe@microsoft.com