Quantum Computing Project

Introduction

The task set out to us by Microsoft was to learn about Quantum Computing, explore the newly released software development kit and test the kit to determine whether the kit was an accurate simulation of how a true quantum computer would work. The development kit includes what a developer would need to get started in the field of quantum computing: a Q# language and compiler along with a Q# library, a local quantum computing simulator, and a Visual Studio extension.

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.

Possibly one of the most exciting things about the advent of Quantum computing is the risk it poses to cryptocurrencies. All members in the team were interested in exploring this, but we lacked the in-depth knowledge of the field that would be required to pursue a project in this field.

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.

Finally, we, and our mentors at Microsoft, agreed that the best method to test the kit based on the knowledge and skill set of the team was to implement an algorithm in Q#. We decided to implement the Triangle Finding algorithm. To test the software by implementing this algorithm, we implemented two traditional computational algorithms to solve the triangle finding problem and then compare this to our quantum algorithm.

Preparation

To start, the team would like to talk about their experience of using the development kit. The initial instructions when setting up the extension proved slightly tricky for some members, and we had to organise a meeting to help each other set up. After getting set up in Visual Studio, with the extension added on, cloning the repositories proved to be a much simpler task in comparison. The next major challenge was to learn Q#. To do this, the team looked at sample Q# programs provided on the available github repository, and tried to understand not only the math behind the problems/algorithms explained, but also the implementation. After this, we were ready to move onto the Triangle Finding Problem.

Before discussing the solutions, we must discuss what the Triangle Finding Problem is and why the team chose to implement this algorithm. The triangle-finding problem is the problem of determining whether a given graph contains a triangle (a clique of size 3). Classical algorithms depend on the speed of matrix multiplication which has a time complexity of O(v^log2(3)) or O(v^3). There exist theoretical matrix multiplication algorithms such as Strassen's Algorithm with has one fewer assignment in it's recursive calls bringin the time complexity down to O(v^log2(7)) or ~O(v^2.81..). The best known matrix multiplication algorithms such as the Coppersmith-Winograd algorithm which as an approximate time complexity of O(v^2.4), where v is the number of vertices, however such algorithms can not be implemented practically. There exists a quantum algorithm theorised to run with Õ(v^1.25).

All the students on the team had done some amount of Graph Theory previously, giving us a base to begin on. Having seen the Graph Theory before, meant that we could understand the problem better and much faster. The team started off by researching an implementation in a language they recognised. Each team member was allowed to choose what language they preferred to research. The research was done only to aid the team understand the problem with more efficiency. After this, we could begin the research and implementation of the quantum algorithm. Our research and implementation is highlighted in the next section.

Research

We began by borrowing the book 'Quantum Computing' (1999 Jozef Gruska) from the library here in Trinity College Dublin. One of the team members took it upon themselves to read parts of the book that may be relevant to the problem. Having little success with the book, due to the amount of information, and a lack of time, we moved towards focusing on research papers online to help us with the implementation. The two research papers that we focused on are:

I. Grover's Search Paper: https://www.cs.cmu.edu/~odonnell/quantum15/lecture04.pdf

Grover's algorithm is a quantum algorithm for searching an unstructured database. It makes use of quantum phenomena such as superposition and amplitude amplification. We used this search twice in our implementation of the algorithm.

II. Triangle finding problem Paper: https://homepages.cwi.nl/~rdewolf/publ/qc/ed.pdf

Implementation

Our implementation of the Triangle Finding problem in Quantum is highlighted in the steps below:

1. Create array of all possible edges from adjacency matrix
2. Create an array of qubits, being log to the base 2 of the size of the array of edges, ceilinged, and with one extra which is the marked qubit that will contain the result
3. Create an oracle from the marked elements of the array. The oracle is a so called 'black box' which is essentially an unknown boolean function which returns one or zero
4. Run the oracle, followed by amplitude amplification for a predetermined number of steps
5. Check the marked qubit for a success or failure
6. Find the edge that the search found
7. If the marked qubit shows success, this is an edge. If successful, move to step 8, otherwise, repeat from step 2, or exit with failure if the max number of repeats has been reached
8. Similarly, find the array within the adjacency matrix that corresponds to one of the vertices, and run steps 2 to 6
9. If a success is found, this means that the vertex found makes an edge with the vertex we are checking
10. Check the second vertex of the found edge to check if an edge exists with the third vertex
11. If one does, a triangle is present, otherwise return to do steps 2 to 6 to find another vertex, or go back and find another edge if the max number of repeats has been reached

Results

To display our results, we chose to implement a graph that displayed the time comparison of the classical algorithms against the quantum algorithms.

When the "Make graph" button is clicked, many adjacency matrices are randomly generated for different numbers of vertices. We found that it was necessary to generate a lot of matrices for each size, as the variance in time for certain algorithms was huge. These adjacency matrices are fed into each triangle finding algorithm, and accurately timed. Then the average time for each number of vertices, for each algorithm, is displayed on a graph. As the time taken for certain algorithms is much longer than others, and we are more interested in the time complexity differences, each line is scaled to end at the same point.

Final Implementation

Our implementation of this quantum algorithm can be found at :

https://github.com/PraptiSetty/QuantumComputing.git