PHYM004: Quantum Computing

James Bull

26/02/1017

Abstract

This report will analyse a C program designed to mimic the operations of a quantum computer. The operations in question include applying basic operators such as the Hadamard gate and the CNOT gate to manipulate the values stored in the qubit addresses, while combining these to perform more complex tasks such as Grover's quantum search algorithm and Shor's algorithm. The program successfully performed these operations for an arbitrary number of qubits.

1 Introduction

QuantumComputing.c is a program designed to imitate the operations of a quantum computer. It does this by storing an array of size 2^N to represent a system of N qubits, qubits being the equivalent of bytes for a quantum computer. The individual qubits can either be in the ground state or an excited state, thus 2^N elements are needed to represent every different state of the system. The state of a qubit can be changed by multiplying the matrix by an array, a process which has the same effect as applying a quantum operator gate to a real set of qubits. The gates which the program simulates are a Hadamard gate which changes a qubit from an excited state to a ground state or vice versa, a phase shift gate which has the effect of changing the complex phase of a state, a CNOT gate which operates like a Hadamard gate but allows a qubit to dictate whether or not a qubits state is changed and finally a CPhase gate which acts like a CNOT gate but using a phase shift gate instead of a Hadamard gate. A combination of these can be used to implement Grover's quantum search algorithm which is able to quickly and efficiently search a set of qubits for an excited state and Shor's quantum factoring algorithm which is an algorithm able to efficiently find the lowest common denominators of a given number. A detailed theory and implementation of all of these operational can be found in the paper by D.Candela [1]. This report will analyse the capability of the program to perform these tasks and discuss the design choices which make up the program.

2 Design Choices

Dynamic memory allocation was used throughout the program as one of the goals was to simulate as many qubits as possible. As all arrays and operators scale with 2^N for N qubits, the memory required for the arrays can very quickly grow too large to allocate to the stack, thus dynamic memory allocation can be used to allocate memory, only being limited by hardware.
The memory allocation is even more extreme for the operators, all of which would have 2^{2N} elements. To limit the memory used by the operators, sparse matrices were used, D.Candela gives a method of creating sparse matrices for these operators by using the fact that each operator has a maximum of two non-zero elements per row; leading to a sparse matrix with 2^N rows, each with a value stored in the element and an integer to indicate which column the value is stored in. Initially this method was utilised however it was found that a more general method of storing the location of the element in the form of two integers followed by the value stored in the element. To do this one has to store both types of int and double complex in the same array, this can be done with the use of a union so long as one is careful when allocating space for the union to allocate enough space for the largest type in the union (in this case a double complex). Normally the standard way to do this would be a tagged union, that is a union where there is a flag allowing the program find out what type an element is. A tagged union is not necessary in this case as the first two elements of each part of the array will always be integers, and the third element will always be a double complex; this means that so long as the programmer is aware of the format of the sparse matrix, the tags would not find use.
When it came to allocating memory for the operators, a choice was made to have as few operators as possible allocated at a single instance. The reasoning for this is that for small N this will not increase the run time too significantly, while for large N this approach will not only allow for larger N as it is more memory efficient (the goal for this program was interpreted as being to increase the number of qubits as much as possible) but it may actually not be too much slower due to the fact that memory allocation slows as the program uses more memory. An exception to this design choice was in the Grover's quantum search algorithm where all the Hadamard operators were allocated and kept throughout the function. This was because the function needs N Hadamard operators which can have a large impact on the performance of the program, even for small N ($N < 8$).

3 Results

All sections of the results will refer to the projects mentioned in the paper by D.Candela.

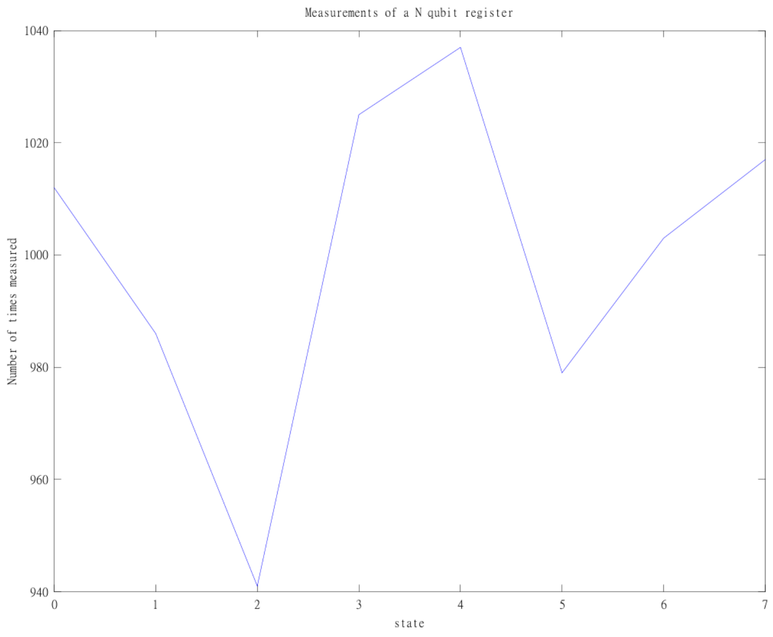

Figure 1: A plot showing the measurements obtained from a register of N qubits with even probability amplitudes.

Part 1 of the project required the program to simulate a series of measurements of the N qubit register. Figure 1 shows a plot for 1000 measurements of 3 qubits, all set to have a probability amplitude of 1.
Part 2 of the project was simply to apply Hadamard gates and phase shift gates to the register to make certain states, this was done with no issues.

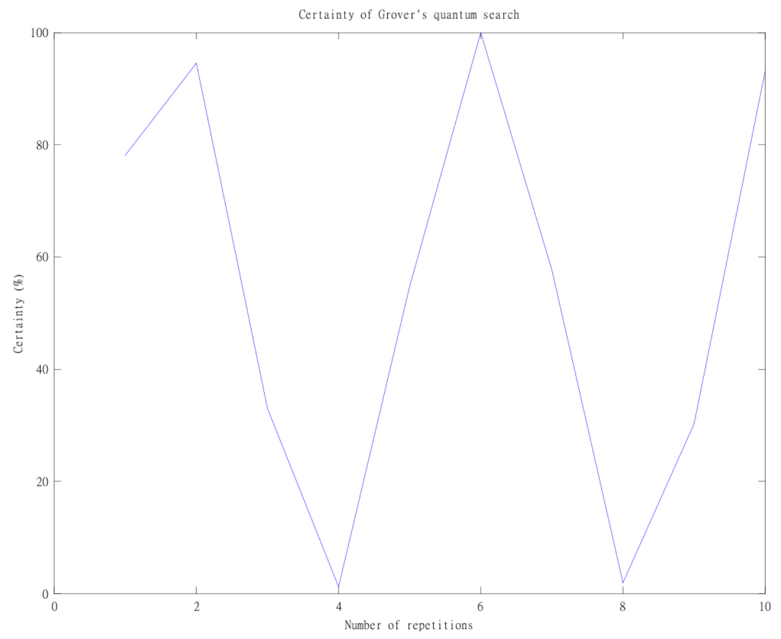

Figure 2: A plot showing the change in certainty of Grover's quantum search with number of repetitions of the algorithm ($N = 3$).

Part 3 of the project was to implement Grover's quantum search, this was done and the function works as expected. In the algorithm the main body is repeated many times to increase the certainty of the result, it was suggested that one should investigate how to certainty (given by the probability of the answer state) of the answer changes as the number of repetitions of the algorithm changes, the results of this can be seen in figure 2 where it is visible that the certainty is periodic. There exists an optimum number of repetitions to perform in the algorithm, this number is the closest integer to $\pi/4\sqrt{2^N}$. For $N = 3$ the optimum number of repetitions is 2, meaning that the number describing the optimum number of repetitions is likely to have been designed to hit the first peak in figure 2, thus maximizing certainty and minimizing execution time.
Part 4 of the project was to perform a series of operations using CNOT and Hadamard gates to make certain states, this was also done with no issues.
Parts 5 and 6 of the project were to extend the existing functions to work with an arbitrary number of qubits, this has been done successfully and Grover's quantum search has been run with greater than 9 qubits. More qubits are possible however the runtime increases exponentially due to the matrix multiplications being very costly as they are called many times. This makes it very time consuming to run very large numbers of qubits on some systems, for example Grover's quantum search using 8 qubits takes approximately 1 minute to

compute, using 9 qubits it takes over 20 minutes.

4 Conclusion

In conclusion the program has successfully simulated operations of a quantum computer using an arbitrary number of qubits. Design decisions have been made to provide clarity when programming and to optimise memory efficiency, allowing for a large number of qubits which was the goal of the program. Some elements of the program run slower than desired, a potential future fix to this would be to implement parallel processing to perform matrix multiplications.

5 References

[1] Undergraduate computational physics projects on quantum computings, D.Candela
`AmericanJournalofPhysics83,688(2015);http://doi.org/10.1119/1.4922296`, (last accessed 27/02/2016)