Implementation of Shor's algorithm in pyQuil

William Burton
Stanford University
CS269Q: Quantum Computer Programming

June 6 2019

1 Introduction

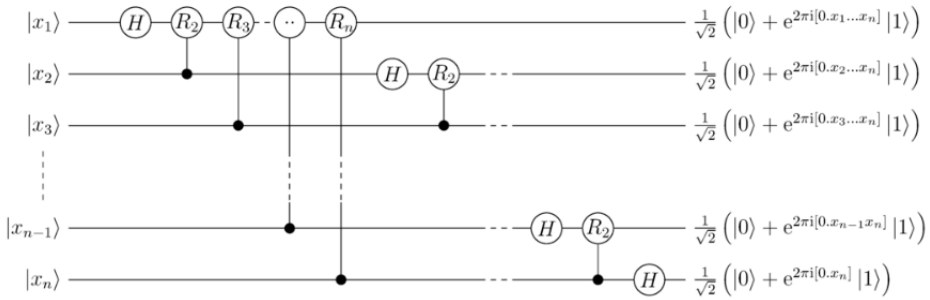
This project demonstrates an implementation of Shor's algorithm for factoring. The interesting component of this, and the topic of the remainder of the report, is an implementation for period finding written in pyQuil, that could be run on a QPU. In the discussed first attempt, older and inefficient methods for period finding were used that were so limiting it was impossible to factor much more than 3 bit numbers. In the second attempt, the code was improved to use the $2n + 3$ qubit circuit construction, that has approximately $O(n^3)$ gate depth, where n is the number of bits in the number to be factored.

2 First Attempt

The first implementation was somewhat naive. The challenging part of Shor's is period finding, which consists of quantum exponentiation, followed by an inverse Quantum Fourier Transform to find the period.

To create the exponentiation circuit, the circuits described in [1] were composed. In this case, the adder used $3n + 1$ qubits. This is due to one n-bit register for the value being added, one n-bit register for the carries, and one $n + 1$-bit register for the sum. As we see in the second attempt, this is drastically overkill if the number being added is known classically at compile time. When doing modular addition, another n-bit register is needed for the modulo. For controlled multiplication, an additional register is used for x, the value being multiplied in which is another n-bit register. Another n-bit register is used in controlled modular exponentiation. This brings the total number of bits for exponentiation up to $3n + 1 + n + n + n = 6n + 1$.

The QFT is relatively standard, and consisted of a structure similar to the one below:



Importantly, this QFT is over $2n$ bits in the period finding algorithm, where n is the number of bits in the number to be factored. Thus the first attempt came in at a whopping $8n+1$ qubits needed to factor. Needless to say, this was insufficient as it was not possible to properly factor 4 bit primes on a personal computer in a reasonable time. Thus, the more performant second attempt followed.

3 Second Attempt

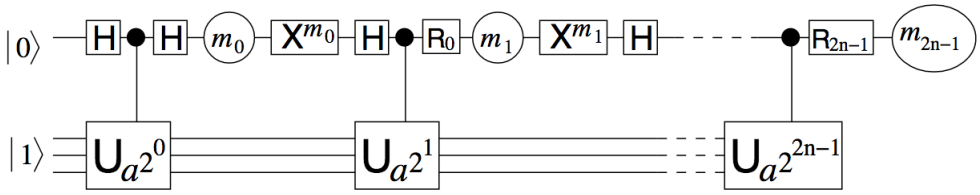
After further research it many improvements were found in [2], which suggests a method for implementing Shor's algorithm using only $2n+3$ qubits. For this method, a few interesting optimizations are used.

The first improvement, as hinted before, is that when adding a number that is known classically at compile time, the addition can be reduced to unitary single qubit gates in the Fourier space. This reduces the number of bits needed for addition down to $n+1$ qubits, the extra to allow for over/underflow detection.

To perform modular addition, only one additional bit is needed as an ancilla to detect underflow when subtracting by N, bringing the number of qubits up by only 1 to $n+2$.

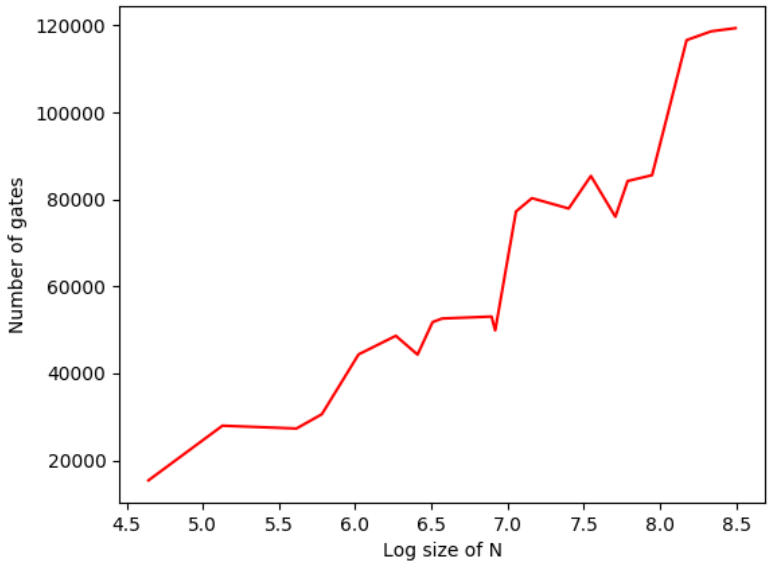
Multiplication necessarily has an additional register of size n, so the total goes up to $2n+2$, and is the same for modular multiplication.

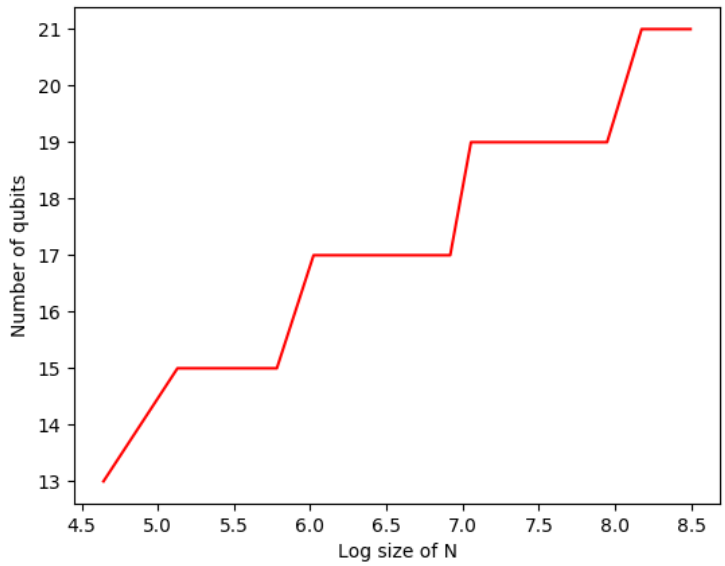
Finally, and probably most importantly, the paper discusses how to implement the QFT^{-1} interleaved with the exponentiation using just one more quibit. This brings the total number of qbits needed for period finding to a minuscule $2n+3$. This is a remarkably clever design and is the "secret sauce" of cutting down on the number of qubits used.



4 Runtime analysis

When implementing the final code, attempts were made to adhere to the specification in [2]. Therefore the final code uses $2n+3$ qubits, and a sub exponential number of gates. When using this implementation of period finding with the classical implementation of the rest of Shors algorithm from [3], it was actually possible to factor some products of primes on the QVM. The primes were not very large, however, with the demo product being 21 and taking a few seconds. Below are graphs of both the number of gates and the number of qubits used for the circuit to find the period of relative variously sized products.





These graphs are for all N such that N is a product of a pair of two primes up until 19, using $g = 3$ for the initial guess.

Of note, there remain some improvements to gate count that were not implemented. Foremost is condensing the addition gates into exactly n single qubit gates. This is possible because the gates are all known at compile time and the unitaries can be composed. The implementation detail that inhibited this was bloating the instruction set by defining many different gates. However, it may be the case that there actually is not performance loss here and would be good further work. The expected gain of reducing the gate size by a factor of n is very desirable.

Additionally, it is possible to reduce the number of swap gates in the modular multiplication gates, but as mentioned in [2], this is not particularly interesting as it leaves the total runtime relatively unaffected.

5 Conclusion

From cursory research, this is one of the first implementations of Shors algorithm that is publicly available on github to use the Quil instruction set. The other implementations, such as [3], frequently implement their own incarnation of a quantum simulator, and many do not directly translate well to actually implementation. Not only is this interesting for the Quil language, but it is also interesting as it can be run directly on an actual QPU. Check out the code repo at https://github.com/burtonwilliamt/pyquil_shors.

6 References

1 Vendral et al. Quantum Networks for Elementary Arithmetic Operations. 11 1995.

2 Beauregard. Circuit for Shor's algorithm using 2n+3 qubits. 5 2002

3 toddwildey. shors-python https://github.com/toddwildey/shors-python.

4 Peter Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. 8 1995.