International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6S3, September 2019

Implementation of Shor's Quantum Factoring Algorithm using Project Q Framework

Adjie Wahyu Wicaksono, Arya Wicaksana

Abstract: ***Prime number factorization is a problem in computer science where the solution to that problem takes super-polynomial time classically. Shor's quantum factoring algorithm is able to solve the problem in polynomial time by harnessing the power of quantum computing. The implementation of the quantum algorithm itself is not detailed by Shor in his paper. In this paper, an approach and experiment to implement Shor's quantum factoring algorithm are proposed. The implementation is done using Python and a quantum computer simulator from ProjectQ. The testing and evaluation are completed in two computers with different hardware specifications. User time of the implementation is measured in comparison with other quantum computer simulators: ProjectQ and Quantum Computing Playground. This comparison was done to show the performance of Shor's algorithm when simulated using different hardware. There is a 33% improvement in the execution time (user time) between the two computers with the accuracy of prime factorization in this implementation is inversely proportional to the number of qubits used. Further improvements upon the program that has been developed for this paper is its accuracy in terms of finding the factors of a number and the number of qubits used, as previously mentioned.***

Index Terms: ***prime factorization, projectQ, quantum algorithm, Shor.***

I. INTRODUCTION

Quantum mechanics is a fundamental theory in physics which describes the inner workings of the universe on the smallest energy scale [1]. This fundamental theory establishes new understandings and possibilities previously untapped by classical physics [2]. Today the quantum computer is one of the biggest scientific breakthroughs in the 20th century. A quantum computer is made by infusing the theory of quantum mechanics into computer science. It harnesses the power of quantum mechanics to perform computation [2], [3]. Thus, a quantum computer allows new kinds of computations which are previously impossible to be done in a classical computer [3].

IBM Q System One (IBM Q) is the first quantum computer that is publicly available through an online platform [4]. This quantum computer is able to simulate the bondings of elements, the learning process of artificial intelligence, and optimize the process of quantum computation itself [4]. Major challenges in developing a quantum computer drive engineers to create a platform that enables the simulations of quantum computers [5]. ETH Zurich has a publicly accessible Python library and framework that simulates quantum computation on a classical computer named ProjectQ [6]. This library uses its own simulator and compiler developed by ETH Zurich to compile and execute quantum programs on a classical computer [6].

ProjectQ contains FermiLib, plugins for FermiLib, as well as compatibility with OpenFermion, all of which are open-source projects for quantum simulation algorithms. All examples that work with these frameworks naturally work with ProjectQ [5]. ProjectQ also provides a tutorial of Shor's algorithm implementation using the framework. In comparison with another simulator such as QDK (Quantum Development Kit) which only provide an example in the documentation. Moreover, there is no implementation of Shor's algorithm available yet in neither pyQuil nor Qiskit [5]. The availability of the implementation of Shor's algorithm in ProjectQ encourages another implementation of Shor's quantum factoring algorithm to be done using the framework. This is due to the fact that the framework already has sufficient support for implementing Shor's quantum factoring algorithm.

Quantum Computing Playground is another quantum computer simulator that was developed by a group of Google engineers in 2014 [7]. It uses QScript scripting language and compiler to simulate a quantum computer and available for public use. The scripting language is specifically built for the Quantum Computing Playground website [7]. The simulator is able to simulate quantum computations up to 22-qubits and offers its own implementation of Shor's and Grover's quantum algorithm for public use [7].

One of the usages of quantum computing is integer factorization [8]. This is a big discovery in mathematics and computation because this process takes super-polynomial time using a classical computer [9]. In number theory, this problem is called integer factorization problem. Integer factorization is the process of breaking down a composite integer into smaller ones that when multiplied back produces the initial value [10]. In fact, this problem is used as the fundamental security factor in the RSA cryptography system [11]. In 1994, Peter W. Shor discovers the quantum factoring algorithm that is able to solve the integer factorization problem in polynomial time [12]. Shor's quantum factoring algorithm shows better performance compared to classical factorization algorithm such as the General Number Field Sieve, where Shor's quantum factoring algorithm is able to run in $O(\log N)^3$ time and $O(\log N)$ space [9].

Manuscript published on 30 September 2019.
***** Correspondence Author (s)
Adjie Wahyu Wicaksono, Department of Informatics, Universitas Multimedia Nusantara, Tangerang - 15810, Indonesia.
Arya Wicaksana, Department of Informatics, Universitas Multimedia Nusantara, Tangerang - 15810, Indonesia.

The algorithm aims to find the order of $x^a \bmod N$ using a quantum computer for the reason that factorization can be reduced by finding the order of an element [13]. However, Peter W. Shor never published the implementation of his quantum algorithm as there was no quantum computer available yet by the time the paper was published.

The development of quantum computer and quantum computer simulator today has established quantum computing as a new field in computer science. The way a quantum computer works is different from a classical computer. Peter W. Shor with his quantum factoring algorithm has demonstrated theoretically the power of quantum computer over a classical computer. There have been several experiments done to optimize Shor's factoring algorithm such as by optimizing the order finding circuit or optimizing the matrix product state where both shows a significant leap of performance [14], [15]. This research will use Shor's original algorithm despite there being further breakthrough regarding the algorithm since its conception in 1994. This is done to show the basic essence of Shor's factoring algorithm.

Thus, this paper proposes the implementation of Shor's quantum algorithm using the ProjectQ framework. The performance of the implementation is evaluated on two different hardware and measured by the execution time (user time) [16], [17]. The results are compared with implementation done by the Quantum Computing Playground and implementation done by the ProjectQ. This study aims to expose the power of quantum computing and the way to achieve it by harnessing quantum mechanics. This paper also exhibits the experimentation of implementing Shor's quantum factoring algorithm for prime factorization.

II. METHODS

The foundation of the implementation of Shor's quantum factoring algorithm in this research is provided by Matthew Hayward's in [18]:

1) Determine if the number N (the number to be factored) is a prime or an even number. The process will continue with Shor's algorithm if it's neither.
2) Pick an integer q that is the power of two which satisfies $n^2 \leq q < 2n^2$.
3) Pick a random integer x that is coprime to n.
4) Partition a quantum register into two parts, Register 1 and Register 2. Register 1 must have enough space to store integers up to $(q\text{-}1)$ while Register 2 needs enough space to store integers up to $(n\text{-}1)$.
5) Load Register 1 with an equally weighted superposition of all integers from 0 to $(q\text{-}1)$. Load Register 2 with all zeros.
6) Apply the transformation $x^a \bmod N$ to Register 1 and store the results in Register 2.
7) Measure Register 2 to get the value of k. This process has the side effect of collapsing Register 1 into the same value to that of Register 2 due to the quantum entanglement.
8) Perform the discrete Fourier transform on Register 1.
9) Measure the state of Register 1 to get the value of m. This variable has a high chance of being a multiple of q/r, where r is the desired period.
10) Perform floating point division of m/q, then calculate the best rational approximation to m/q whose denominator is less than or equal to q. This denominator is taken as the candidate for r. If the candidate r is odd, it could be doubled if doing so leads to a value less than q.
11) The factors of N can be determined by taking $GCD(x^{r/2} + 1, N)$ and $GCD(x^{r/2} - 1, N)$.

In addition to the steps stated above, here in Fig. 1 is the additional step added based on ProjectQ's implementation of Shor's quantum factoring algorithm to determine the factors of N when only one of the factors is found.

```
IF f1 * f2 != N && f1 * f2 > 1 && (N/(f1 * f2)) * f1 * f2
        f1 = f1 * f2
        f2 = N/ (f1 * f2)
END IF
```

Fig. 1. Pseudocode of the additional step

Fig. 2 shows the pseudocode of the modular exponentiation process used in this research. Found in step 6 of Matthew Haywards implementation of Shor's quantum factoring algorithm, this process applies the transformation $x^a \bmod N$ to Register 2. In this case, the measurements array could be prepared in an equally weighted superposition of the value q. This process will produce the value of q/r where it will later then be used to find the desired period.

The value of n is the number of bits required to represent the integer value of $q\text{-}1$ while a represents Register 1 that is loaded with an equally weighted superposition of integers up to $q\text{-}1$. Due to the lack of ways for doing modular exponentiation, here it is done by performing modular multiplication qubit-by-qubit. The value of a that has gone through modular multiplication is stored within the ctrl_qubit variable which is then measured and stored in the measurements array. All the values inside array measurements are added up to find the value of q/r as mentioned in Step 9 of Hayward's implementation of Shor's quantum factoring algorithm.

```
FOR k = 0 until n
        current_x = pow(x, 1 << (n - 1 - k)) mod N
        H(ctrl_qubit)
        WITH Control(ctrl_qubit)
                (a * current_x) mod N
        END WITH
        Measure(ctrl_qubit)
        measurements[k] = ctrl_qubit
END FOR
```

Fig. 2. Pseudocode of the quantum sub-routine

The testing method that is used for this research is the white-box testing methodology, which is a testing method where the program is examined at each stage of the process to determine whether the examined parts are working as intended or not. The program's performance is compared to ProjectQ's implementation and Quantum Computing Playground's implementation of Shor's quantum factoring algorithm. The test-cases used for testing can be seen in Table I. The implementation is evaluated by measuring the execution time, specifically the user time.

User time is the wall-clock time that is assigned to the simulator [14]. Besides comparing the user time upon different implementations of the algorithm, the performance of each implementation is also evaluated on two different computers. The two computers that are used in this research are explained in Table II.

Table- I: Test-cases

Case	N	Factors (F1, F2)
1	15	3, 5
2	21	3, 7
3	33	3, 11
4	35	5, 7
5	55	5, 11

Table- II: Computers specification

	Computer 1	Computer 2
RAM	32GB DDR4	8GB
Processor	Intel Core Processor i7-8700k	Intel Core Processor i5-8250u
Number of cores	6	4
Clock Speed	3.70-4.70GHz	1.60-1.80GHZ

III. RESULTS AND ANALYSIS

A. Testing

Table III presents the testing results of variables N, isPrime, isEven, and q according to the test-cases previously described. N refers to the input of the program. The isEven and isPrime variables are a boolean data type that keeps track if N is either an even number or prime number respectively. The value of variable q is determined in the second step.

Table- III: Testing results for N, isEven, isPrime, and q

Case	N (input)	isEven	isPrime	q
1	15	False	False	256
2	21	False	False	512
3	33	False	False	2048
4	35	False	False	2048
5	55	False	False	4096

Table IV shows the results of the value of a which is the equivalent of register 1 in Matthew Hayward's implementation of Shor's algorithm. It is a quantum register that will hold the equally weighted superposition of the value of variable q.

Table- IV: Testing results for value *a* (register 1)

Case	*a (register 1)*
1	$\frac{1}{\sqrt{256}}\|00000000\rangle , \dots , \frac{1}{\sqrt{256}}\|11111111\rangle$
2	$\frac{1}{\sqrt{512}}\|000000000\rangle , \dots , \frac{1}{\sqrt{512}}\|111111111\rangle$
3	$\frac{1}{\sqrt{2048}}\|000000000000\rangle , \dots , \frac{1}{\sqrt{2048}}\|111111111111\rangle$
4	$\frac{1}{\sqrt{2048}}\|000000000000\rangle , \dots , \frac{1}{\sqrt{2048512}}\|111111111111\rangle$
5	$\frac{1}{\sqrt{4096}}\|0000000000000\rangle , \dots , \frac{1}{\sqrt{4096}}\|1111111111111\rangle$

B. Performance Comparison Between Implementations

The performance of each implementation (ProjectQ, this research, and Quantum Computing Playground) is compared side by side in Fig. 3. It can be seen that this paper's implementation had a drastic increase in the execution time in the case of 20-qubits usage. This shows that up to 18-qubits, this paper's implementation had the second fastest execution time out of the three. It can also be seen that ProjectQ's implementation needed the least time to execute the algorithm using the least amount of qubits, which is approximately half of those used by the paper's implementation. The cases where its execution time is labeled "N/A" on the figure is caused by either two things: That implementation did not need to use or did not use that amount of qubits or the implementation was unable to calculate the factors using that many qubits.

In this paper, the implementation and simulation that is carried out do not use ancilla qubits. Hence, the implementation is intended for a one-time execution of the algorithm. The problem dimension for prime number factorization is targeted for positive integer with size no larger than 10-bits. The comparison of user time between three implementations on average taken from both computers is presented in Fig. 3.

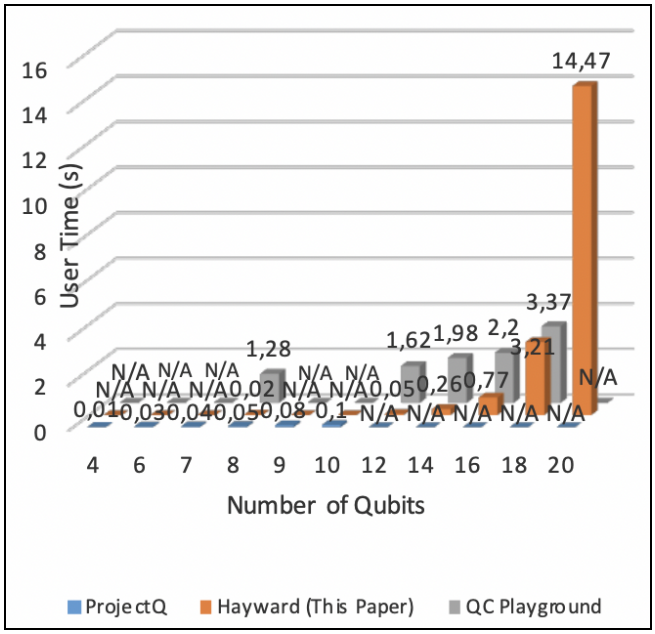

Fig. 3. Performance comparison

C. Performance Comparison Between Computers

Fig. 4 displays the comparison of user time need to factorize an integer between the two computers used in this research. The time shown on Fig. 4 is the average time taken by all three implementations on each computer. It is shown that the number of cores and the clock-speed do affect the execution time of Shor's quantum algorithm when executed on a simulator. It is shown that the performance received an increase of 102% on average between the two computers.

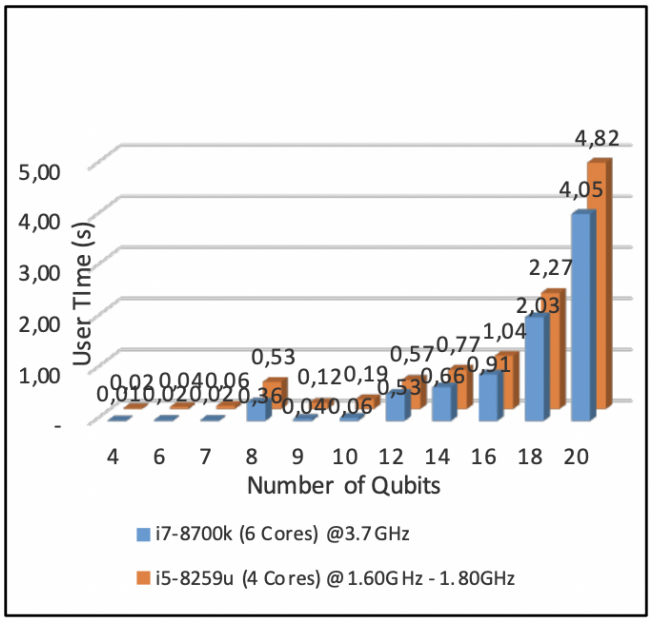

Fig. 4. Average performance comparison between computers

Hayward

Fig. 5 shows the performance of the implementation of Shor's quantum algorithm done in this paper using two different computers. This implementation has the longest execution time between cases with an increase of 322% on average. It also has an average increase in performance of 33% when executed on Computer 1. Most notably, This implementation has the highest leap of execution time between the case using 18-qubits and 20-qubits.

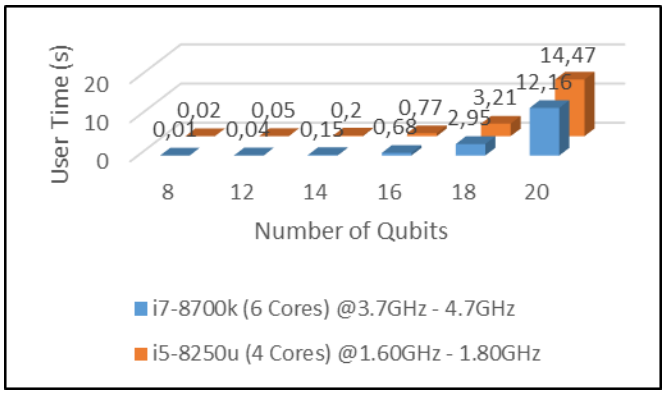

Fig. 5. Performance comparison of the implementation

ProjectQ

Fig. 6 presents the performance of ProjectQ's implementation of Shor's quantum algorithm on two different computers. This implementation produces the highest amount of increase when using Computer 1 with an average increase in performance of 164%. It also has an increase in execution time of 68% between each case of qubits used on average.

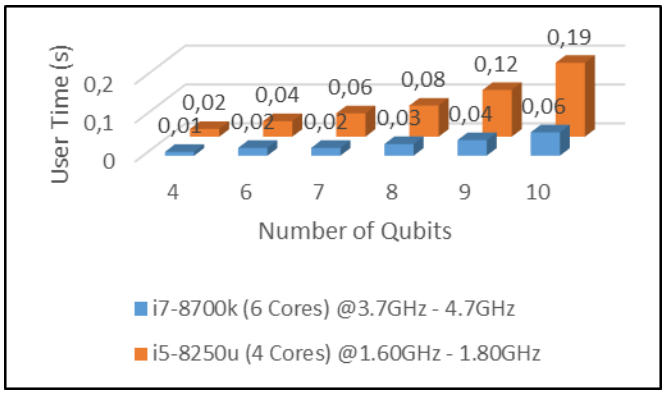

Fig. 6. Performance comparison of ProjectQ's implementation

Quantum Computing Playground

Compared to the two previous implementations, Quantum Computing Playground's (QCP's) implementation produces the least amount of increase in performance between the two computers used with 18% increase on average. It also has the least amount of increase between the number of qubits used with an average of 28% increase. The chart labeled "N/A" shown in Fig. 7 below is caused by this implementation ability to calculate the factors of a 10-bit integer.

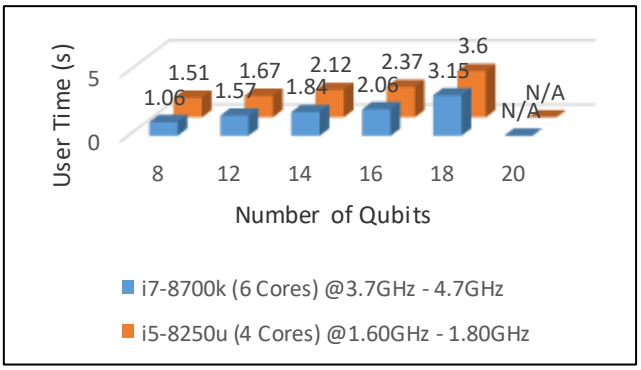

Fig. 7. Performance comparison of QCP's implementation

D. Success to Failure Ratio

In Table V, the ratio of success to failure of the program developed in this paper is presented. Each ratio is taken from 30 different trials for each case of qubits used (8, 12, 14, 16, 18, and 20). It can be inferred from the table below that the number of qubits used to factorize an integer is inversely proportional to the accuracy of finding the factor of the given integer.

This is due to the probabilistic nature of quantum which is also one of the fundamental differences between quantum computing and probabilistic classical computing. Quantum algorithms are often probabilistic including Shor's quantum factoring algorithm, in that they provide the correct solution only with a certain known probability.

Table- V: Success to failure ratio

Number of qubits	Ratio (success to failure)
8	21:9
12	19:11
14	18:12
16	13:17
18	9:21
20	8:22

IV. CONCLUSION

In this research, the proposed implementation of Shor's quantum algorithm has been successfully implemented using the ProjectQ framework. It could also be concluded that the clock-speed and number of cores that are used by a computer affect the execution time needed to run Shor's quantum factoring algorithm. The implementation developed for this research received an increase in performance of 33% on average. Furthermore, the program developed for this research shows to be the slowest one when factoring numbers of size 10-bit or higher compared to ProjectQ and Quantum Computing Playground's implementation. On the opposite, ProjectQ's implementation had the fastest execution time while also using the least amount of qubits. In addition, this research also shows that the number of qubits used to factorize an integer is inversely proportional to the accuracy of finding the factor of the given integer.

Further research can be done on two major aspects. First, ProjectQ's implementation shows that Shor's quantum algorithm can be executed with fewer qubits than originally calculated. Knowing this, the number of qubits required to run Shor's quantum factoring algorithm in this program can be reduced to increase its performance (simulation speed). Second, further research can also be focused on increasing the accuracy (success to failure ratio) of finding the factors of larger integers (larger number of qubits).

REFERENCES

1. R. Feynman, R. Leighton, M. Sands, and R. Lindsay, "The Feynman Lectures on Physics, Vol. 3: Quantum Mechanics," *Physics Today*, vol. 19, no. 11, 1966, pp. 80-83.
2. M. Nielsen, I. Chuang, and L. Grover, "Quantum Computation and Quantum Information," *American Journal of Physics*, vol. 70, no. 5, 2002, pp. 558-559.
3. A. Avila, R. Reiser, A. Yamin, and M. Pilla, "Parallel simulation of Shor's and Grover's algorithms in the distributed geometric machine," *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2017, pp. 412-419.
4. G. Dueck, A. Pathak, M. Rahman, A. Shukla, and A. Banerjee, "Optimization of Circuits for IBM's Five-Qubit Quantum Computers," *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018.
5. R. LaRose, "Overview and Comparison of Gate Level Quantum Software Platforms," *Quantum*, vol. 3, 2019, p. 130.
6. D. Steiger, and T. Häner (2019). ProjectQ. [Online]. Available: http://projectq.ch/.
7. L. Culp (2019). Quantum Computing Playground [Online]. Available: http://www.quantumplayground.net/
8. Y. Wang, H. Zhang, and H. Wang, "Quantum polynomial-time fixed-point attack for RSA," *China Communications*, vol. 15, no. 2, 2018, pp. 25-32.
9. Jr. S. J. Lomonaco, "A Lecture on Shor's Quantum Factoring Algorithm Version 1.1," 2000.
10. S. Nguyen, S. Ghebregiorgish, N. Alabbasi, and C. Rong, "Integer Factorization Using Hadoop," *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011.
11. T. Kleinjung, et al., "Factorization of a 768-Bit RSA Modulus," *Advances in Cryptology – CRYPTO 2010*, 2010, pp. 333-350.
12. P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994.
13. Z. Cao, and L. Liu, "On the Complexity of Shor's Algorithm for Factorization," *2009 Second International Symposium on Information Science and Engineering*, 2009.
14. S. Beauregard, "Circuit for Shor's Algorithm using 2n+3 qubits," *Quantum Information & Computation*, 2003, pp. 175-185.
15. A. Dang, C. D. Hill, and L. C. L. Hollenberg, "Optimising Matrix Product State Simulations of Shor's Algorithm," *Quantum*, vol. 3, 2019, pp. 116-124.
16. A. Wicaksana, and C. Tang, "Virtual Prototyping Platform for Multiprocessor System-on-Chip Hardware/Software Co-design and Co-verification," in *Computer and Information Science*, R. Lee, Ed. Cham: Springer, 2017, pp. 93-108.
17. A. Wicaksana, *Multiprocessor System-on-Chip (MPSoC) virtual platform for hardware and software co-design and co-verification*, Universiti Tunku Abdul Rahman, 2019.
18. M. Hayward, *Quantum Computing and Shor's Algorithm*, University of Illinois, 2015.

AUTHORS PROFILE

Adjie Wahyu Wicaksono, undergraduate student in Informatics at Universitas Multimedia Nusantara.

Arya Wicaksana, graduated from Universiti Tunku Abdul Rahman in VLSI Engineering (M.Eng.Sc.) and Universitas Multimedia Nusantara in Computer Science (S.Kom.). Research interests and works are: computational intelligence and quantum computing. E-mail: arya.wicaksana@umn.ac.id.