Implementing Quantum Finite Automata Algorithms on Noisy Devices

Utku Birkan[1,2], Özlem Salehi[3,7], Viktor Olejar[4], Cem Nurlu[5], and Abuzer Yakaryılmaz[6,7]

[1] Department of Computer Engineering, Middle East Technical University, Turkey
[2] Department of Physics, Middle East Technical University, Turkey
utku.birkan@gmail.com
[3] Department of Computer Science, Özyeğin University, Turkey
ozlemsalehi@gmail.com
[4] Institute of Mathematics, P.J. Šafárik University in Košice, Slovakia
viki.olejar@gmail.com
[5] Department of Physics, Boğaziçi University, Turkey cem.nurlu@gmail.com
[6] Center for Quantum Computer Science, University of Latvia, Latvia abuzer@lu.lv
[7] QWorld Association, https://qworld.net

Abstract. Quantum finite automata (QFAs) literature offers an alternative mathematical model for studying quantum systems with finite memory. As a superiority of quantum computing, QFAs have been shown exponentially more succinct on certain problems such as recognizing the language $\texttt{MOD}_\text{p} = \{a^j \mid j \equiv 0 \mod p\}$ with bounded error, where p is a prime number. In this paper we present improved circuit based implementations for QFA algorithms recognizing the $\texttt{MOD}_\text{p}$ problem using the Qiskit framework. We focus on the case $p = 11$ and provide a 3 qubit implementation for the $\texttt{MOD}_{11}$ problem reducing the total number of required gates using alternative approaches. We run the circuits on real IBM quantum devices but due to the limitation of the real quantum devices in the NISQ era, the results are heavily affected by the noise. This limitation reveals once again the need for algorithms using less amount of resources. Consequently, we consider an alternative 3 qubit implementation which works better in practice and obtain promising results even for the problem $\texttt{MOD}_{31}$.

Keywords: quantum finite automata · quantum circuit · rotation gate · quantum algorithms.

1 Introduction

Quantum finite automata literature offers an alternative mathematical model for studying quantum systems with finite memory. Many different models have been proposed with varying computational powers [5]. Moore-Crutchfield quantum finite automaton (MCQFA) [9] is one of the earliest proposed models which is obtained by replacing the transition matrices of the classical finite automata by unitary operators. Despite the fact that they are weaker than their classical

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77980-1_1

counterparts in terms of their language recognition power, for certain languages MCQFAs have been shown to be more succinct. One such example is the language $\texttt{MOD}_\mathrm{p} = \{a^j \mid j \equiv 0 \mod p\}$, where p is a prime number: MCQFAs were shown to be exponentially more space-efficient than their classical counterparts [3].

Experimental demonstration of quantum finite automata has recently gained popularity. In [12], the authors implement an optical quantum finite automaton for solving promise problems. A photonic implementation for $\texttt{MOD}_\mathrm{p}$ problem is presented in [8]. MCQFA for the $\texttt{MOD}_\mathrm{p}$ problem has been also implemented using a circuit based approach within Qiskit and Rigetti frameworks by Kālis in his Master's Thesis [7].

As a continuation of [7], in this paper we present improved circuit based implementations for MCQFA recognizing the $\texttt{MOD}_\mathrm{p}$ problem using the Qiskit framework. We start with the naive implementation proposed in [7] and provide a new implementation which reduces both the number of qubits and the number of required basis gates, due to an improved implementation of the multi-controlled rotation gate around y-axis and the order in which the gates are applied. We demonstrate the results of the experiments carried out by IBMQ backends for both the improved naive implementation and the optimized implementation of [7]. Regarding the optimized implementation, we experimentally look for the parameters which would minimize the maximum error rate.

We also propose a 3-qubit parallel implementation which works better in practice for the $\texttt{MOD}_{11}$ and $\texttt{MOD}_{31}$ problems. The choice of parameters for this implementation heavily influences the outcomes unlike the optimized implementation where this choice does not have a huge impact on the acceptance probabilities.

We conclude by suggesting a new implementation for the rotation gate around y-axis, taking into account the new *basis gates*, the gates that are implemented at the hardware level–that have been recently started to be used by IBM. This new proposal lays the foundations for future work on the subject.

The source code of our quantum circuits can be accessed from https://gitlab.com/qworld/qresearch/research-projects/qfa-implementation/-/tree/iccs-2021.

2 Background

We assume that the reader is familiar with the basic concepts and terminology in automata theory and quantum computation. We refer the reader to [10,11,5] for details.

Throughout the paper, Σ denotes the finite input alphabet, not containing the left and right-end markers (¢ and \$, respectively), and $\tilde{\Sigma}$ denotes $\Sigma \cup \{¢, \$\}$. For a string $w \in \Sigma^*$, its length is denoted by $|w|$ and, if $|w| > 0$, w_i denotes the i^{th} symbol of w. For any given input string w, an automaton processes string $\tilde{w} = ¢w\$$ by reading it symbol by symbol and from left to right.

There are several models of quantum finite automata (QFAs) in the literature with different computational powers [5]. In this paper, we focus on the known most restricted model called as *Moore-Crutchfield quantum finite automaton* (MCQFA) model [9].

Formally, a d-state MCQFA is a 5-tuple

$$M = (\Sigma, Q, \{U_\sigma \mid \sigma \in \tilde{\Sigma}\}, q_s, Q_A),$$

where $Q = \{q_1, \dots, q_d\}$ is the finite *set of states*, U_σ is the *unitary operator* for symbol $\sigma \in \tilde{\Sigma}$, $q_s \in Q$ is the *start state*, and $Q_A \subseteq Q$ is the *set of accepting states.*

The computation of M is traced by a d-dimensional vector, called the state vector, where j^{th} entry corresponds to state q_j. At the beginning of computation, M is in quantum state $|q_s\rangle$, a zero vector except its s^{th} entry, which is 1. For each scanned symbol, say σ, M applies the unitary operator U_σ to the state vector. After reading symbol \$, the state vector is measured in the computational basis. If an accepting state is observed, the input is accepted. Otherwise, the input is rejected.

For a given input $w \in \Sigma^*$, the final state vector $|v_f\rangle$ is calculated as

$$|v_f\rangle = U_{\$} U_{w_{|w|}} \cdots U_{w_1} U_{¢} |q_s\rangle .$$

Let $|v_f\rangle = (\alpha_1 \;\; \alpha_2 \;\; \cdots \;\; \alpha_d)^T$. Then, the probability of observing the state q_j is $|\alpha_j|^2$, and so, the accepting probability of M on w is $\sum_{q_j \in Q_A} |\alpha_j|^2$.

3 $\mathtt{MOD_p}$ Problem and QFA Algorithms

For any prime number p, we define language

$$\mathtt{MOD}_{\mathrm{p}} = \{a^j \mid j \equiv 0 \mod p\}.$$

Ambainis and Freivalds [3] showed that MCQFAs are exponentially more succinct than their classical counterparts, i.e., $\mathtt{MOD}_{\mathrm{p}}$ can be recognized by an MCQFA with $\mathcal{O}(\log p)$ states with bounded error, while any probabilistic finite automaton requires at least p states to recognize the same language with bounded error. The MCQFA constructions given in [3] were improved later by Ambainis and Nahimovs [4].

3.1 2-State QFA

We start with giving the description of a 2-state MCQFA that accepts each member of $\mathtt{MOD}_{\mathrm{p}}$ with with probability 1 and rejects each nonmember with a nonzero probability.

Let M_p be an MCQFA with the set of states $Q = \{q_1, q_2\}$, where q_1 is the starting state and the only accepting state. The identity operator is applied when reading ¢ or \$. Let $\Sigma = \{a\}$, which is often denoted as a unary alphabet. For each symbol a, the counter-clockwise rotation with angle $2\pi/p$ on the unit circle is applied:

$$U_a = \begin{pmatrix} \cos(2\pi/p) & -\sin(2\pi/p) \\ \sin(2\pi/p) & \cos(2\pi/p) \end{pmatrix} .$$

The minimal rejecting probability of a non-member string w by the automaton M_p is $\sin^2(|w| \cdot 2\pi/p)$, which gets closer to 0 when $|w|$ approaches an integer multiple of p. One may notice that instead of $2\pi/p$, it's also possible to use the rotation angle $k \cdot 2\pi/p$ for some $k \in \{1, \dots, p-1\}$. It is easy to see that the rejecting probability of each non-member differs for different values of k, but the minimal rejecting probability will not be changed when considering all non-members.

On the other hand, to obtain a fixed error bound, we can execute more than one 2-state MCQFA in parallel, each of which uses a different rotation angle.

3.2 $\mathcal{O}(\log p)$-State QFAs

Here we explain how to combine 2-state MCQFAs with different rotation angles to obtain a fixed error bound.

First, we define the 2-state MCQFA M_p^k as same as M_p except for the rotation angle for the symbol a, which is now $k \cdot 2\pi/p$ where $k \in \{1, \dots, p-1\}$.

Then we define the $2d$-state MCQFA M_p^K, where K is a set formed by d many k values: $K = \{k_1, \dots, k_d\}$ and each $k_j \in K$ is an integer between 1 and $p-1$. The MCQFA M_p^K executes d 2-state MCQFAs $\{M_p^{k_1}, \dots, M_p^{k_d}\}$ in parallel. The state set of M_p^K is formed by d pairs of $\{q_1, q_2\}$:

$$\{q_1^1, q_2^1, q_1^2, q_2^2, \dots, q_1^d, q_2^d\}\,.$$

The state q_1^1 is the starting state and the only accepting state. At the beginning of the computation, M_p^K applies a unitary operator $U_{¢}$ when reading the symbol ¢ and enters the following superposition:

$$|q_1^1\rangle \xrightarrow{U_{¢}} \frac{1}{\sqrt{d}}|q_1^1\rangle + \frac{1}{\sqrt{d}}|q_1^2\rangle + \dots + \frac{1}{\sqrt{d}}|q_1^d\rangle\,.$$

In other words, we can say that M_p^K enters an equal superposition of 2-state MCQFAs $M_p^{k_1}, M_p^{k_2}, \dots, M_p^{k_d}$.

Until reading the right end-marker, M_p^K executes each 2-state sub-automaton, $M_p^{k_j}$, in parallel, where $M_p^{k_j}$ rotates with angle $2\pi k_j/p$. Thus, the overall unitary matrix of M_p^K for symbol a is

$$U_a = \bigoplus_{j=1}^{d} R_j = \begin{pmatrix} R_1 & 0 & \cdots & 0 \\ 0 & R_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_d \end{pmatrix},$$

where

$$R_j = \begin{pmatrix} \cos(2\pi k_j/p) & -\sin(2\pi k_j/p) \\ \sin(2\pi k_j/p) & \cos(2\pi k_j/p) \end{pmatrix}.$$

After reading the symbol \$, we apply the unitary operator $U_{\$} = U_{¢}^{-1}$. This overall algorithm gives us an exponential advantage of quantum computation over classical computation for some suitable values for K, for each p. It was shown [3] that, for each p, there exists a set of K with $d = \mathcal{O}(\log p)$ elements such that M_p^K recognizes $\texttt{MOD}_\text{p}$ with a fixed error bound.

4 $\texttt{MOD}_\textbf{p}$ Implementations

In this section, we present our implementation schema in Qiskit and results on simulators and real machines.

4.1 Single Qubit Implementation

We start with a single qubit implementation. An example implementation of 2-state MCQFA M_7 for $\texttt{MOD}_7$ is given in Figure 1 where the input string is aaa:

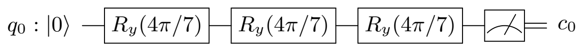


Fig. 1. Single qubit $\texttt{MOD}_7$ implementation

This circuit has one qubit (q_0) and one bit (c_0). There are different rotation operators (gates) in Qiskit. Here we use R_y gate, which is defined on the Bloch sphere and takes the twice of the rotation angle as its parameter to implement the rotation on the unit circle on the $|0\rangle - |1\rangle$ plane. The outcome of the measurement at the end is written to the classical bit c_0.

4.2 Three-Qubit Implementations of $\texttt{MOD}_\textbf{p}$

A Naive Implementation To implement the unitary operator given in Equation (3.2), we use controlled gates, the conditional statements of the circuits. The implementation cost of the controlled gates are expensive and unfortunately, the straightforward implementation of the above algorithm is costly.

Kālis [7] gave a four-qubit implementation of the above algorithm for the problem $\texttt{MOD}_{11}$, where three qubits are used to simulate four sub-automata and one ancilla qubit is used to implement the controlled operators.

Here we present our implementation schema by using only 3 qubits. All diagrams are obtained by using Qiskit [1]. We use three qubits called q_2, q_1, q_0. We implement $U_{\mathbb{C}}$ operator by applying Hadamard gates to q_2 and q_1. The initial state is $|000\rangle$. After applying Hadamard operators, we will have the following superposition, in which we represent the state of q_0 separately:

$$|v_{\mathbb{C}}\rangle = \frac{1}{2}|00\rangle \otimes |0\rangle + \frac{1}{2}|01\rangle \otimes |0\rangle + \frac{1}{2}|10\rangle \otimes |0\rangle + \frac{1}{2}|11\rangle \otimes |0\rangle$$

The unitary matrix for symbol a is represented as follows:

$$U_a = \begin{pmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2 & 0 & 0 \\ 0 & 0 & R_3 & 0 \\ 0 & 0 & 0 & R_4 \end{pmatrix}$$

In order to implement U_a, we apply R_1 when $q_2 \otimes q_1$ is in state $|00\rangle$, R_2 when $q_2 \otimes q_1$ is in state $|01\rangle$, R_3 when $q_2 \otimes q_1$ is in state $|10\rangle$, and R_4 when $q_2 \otimes q_1$ is in state $|11\rangle$.

We pick $K = \{1, 2, 4, 8\}$. Then, after applying U_a, the new superposition $(U_a U_{¢} |000\rangle)$ becomes

$$|v_1\rangle = \frac{1}{2}|00\rangle \otimes R_y\left({}^{2\pi}/_{11}\right)|0\rangle + \frac{1}{2}|01\rangle \otimes R_y\left({}^{4\pi}/_{11}\right)|0\rangle + \frac{1}{2}|10\rangle \otimes R_y\left({}^{8\pi}/_{11}\right)|0\rangle + \frac{1}{2}|11\rangle \otimes R_y\left({}^{16\pi}/_{11}\right)|0\rangle .$$

Once we have a block for U_a, then we can repeat it in the circuit as many times as the number of symbols in the input. If our input is a^m, then the block for U_a is repeated m times. After applying U_a^m, the new superposition becomes

$$|v_m\rangle = \frac{1}{2}|00\rangle \otimes R_y^m\left({}^{2\pi}/_{11}\right)|0\rangle + \frac{1}{2}|01\rangle \otimes R_y^m\left({}^{4\pi}/_{11}\right)|0\rangle + \frac{1}{2}|10\rangle \otimes R_y^m\left({}^{8\pi}/_{11}\right)|0\rangle + \frac{1}{2}|11\rangle \otimes R_y^m\left({}^{16\pi}/_{11}\right)|0\rangle .$$

After reading the whole input, before the measurement, we apply the Hadamard gates which correspond to the operator for symbol \$.

There is no single-gate solution we can use to implement all of these operators though. Besides, the controlled operators are activated only when all of the control qubits are in state $|1\rangle$. This is why X gates are used; to activate the control qubits when they are in state $|0\rangle$. Figure 2 depicts a circuit implementing U_a.

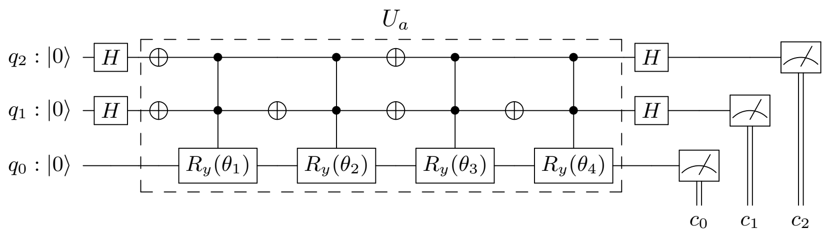


Fig. 2. A naive $\texttt{MOD}_\text{p}$ circuit implementation

Initially, two X gates (represented as $\oplus$ in the circuit diagrams) are applied and R_1 is implemented as the controlled rotation gate will be activated only when the control qubits are initially in state $|00\rangle$. Next, we apply X gate to q_2 and so the controlled qubits are activated when in state $|01\rangle$ and we implement R_2. Similarly, we implement R_3 by applying X gates to both qubits so that the controlled qubits are activated when in state $|10\rangle$, and finally we implement R_4 so that the control qubits are activated in state $|11\rangle$.

Next, we discuss how to reduce the number of X gates and an improved implementation is given in Figure 3. Initially, R_4 is implemented as the controlled operators will be activated only in state $|11\rangle$. Next, we apply X gate to q_2 and so the controlled qubits are activated when in state $|10\rangle$, and so, we implement R_3. We apply X gate to q_1 and similarly the controlled qubits are activated when in state $|11\rangle$ so that we implement R_1. Finally, we apply X gate to q_2 again to implement R_2. Note, that we apply one more X gate at the end so that the initial value of q_2 is restored.

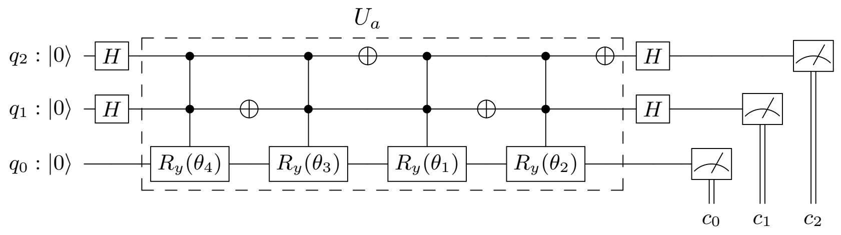
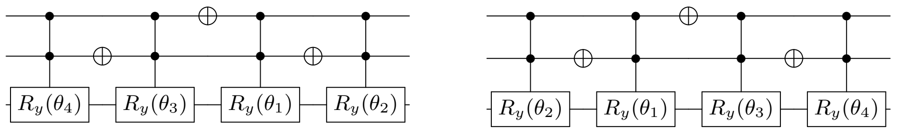


Fig. 3. An improved (but still naive) `MOD`$_\mathrm{p}$ circuit implementation

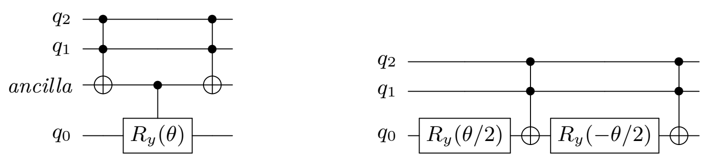
The number of X gates can be reduced further by omitting the last X gate in the above diagram and changing the order of R_y gates in the next round so that they follow the values of the controlled qubits. For each scanned a, either one of the blocks is applied, alternating between the two, starting with the first block. Overall, three X gates are used instead of four X gates for a single a. Note that when the input length is odd, we should always use an extra X gate before the final pair of Hadamard gates. The blocks are depicted below.



(a) First Block (b) Second Block

Fig. 4. Reducing the NOT gates using two blocks to implement U_a

When we implement the circuit in Qiskit, there are different approaches to implement the multi-controlled rotation gate. One option is Kālis' implementation [7] that takes advantage of Toffoli gates and controlled rotations as seen in Figure 5a. Another possibility is to use the built-in RYGate class in Qiskit. An alternative implementation of controlled R_y gate presented in [6] is given in Figure 5b. In this implementation, the rotation gates applied on the target qubit cancel each other unless the control qubits are in state $|11\rangle$ which is checked by the Toffoli gate, thus yielding the same effect as an R_y gate controlled by two qubits. As a further improvement, the Toffoli gate can be replaced with the simplified Toffoli gate, (also referred to as Margolus gate in Qiskit) which has a reduced cost compared to Toffoli gate. This replacement does not affect the overall algorithm as only the states corresponding to first and second sub-automata and that of third and fourth sub-automata are swapped.



(a) Kālis' implementation (b) Alternative implementation

Fig. 5. Controlled R_y gate implementations

Before running a circuit on an IBMQ device, each gate is decomposed into basis gates U_1, U_2, U_3 and CX[8] and this decomposition also depends on the backend on which the circuit is run and the physical qubits used on the machine. In the table given below, the number of basis gates required to implement the rotation gate with two controls using the two approaches is given for the IBMQ Santiago and IBMQ Yorktown machines with the default optimization level.

Table 1. Number of basis gates required by the controlled rotation gate implementations

	`RYGate`				Alternative			
	U_1	U_2	U_3	CX	U_1	U_2	U_3	CX
Santiago	0	0	6	11	1	2	1	9
Yorktown	0	0	6	8	4	1	2	6

Next, we present some experimental results about $\texttt{MOD}_{11}$ problem comparing the performance of the 4 qubit implementation which was originally proposed in [7] and our improved version with 3 qubits where the number of X gates are reduced and the controlled rotation gates are implemented using the alternative approach. The acceptance probability of each word is the number of times the states $|000\rangle$ and $|0000\rangle$ are observed divided by the number of shots (which was

[8] The set of basis gates was changed to CX, I, R_z, $\sqrt{X}$, X in January 2021.

taken as 1000 for the experiments) for 3 qubit and 4 qubit implementations, respectively. The results do not look promising as the acceptance probabilities are around 0.125 and 0.0625 for the 3 qubit and 4 qubit implementations, which are the probabilities of observing a random result. Ideally, the acceptance probabilities for the word lengths 11 and 22 would have been close to 1.

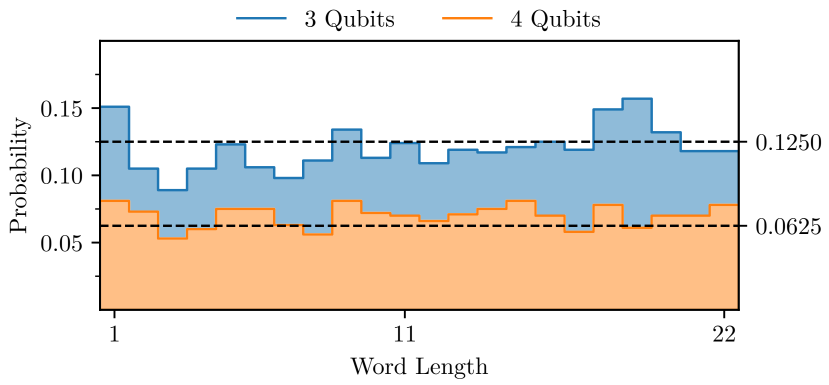


Fig. 6. Acceptance probabilities for $\texttt{MOD}_{11}$ naive implementation

There are three main sources of error in IBMQ machines: number of qubits, circuit depth and the number of CX gates. Even though our 3-qubits naive implementation reduced the number of qubits and CX gates, this improvement was not enough to have any meaningful result. Each Margolous gate still requires 3 CX gates which is better compared to the Toffoli gate which requires 6 but it is still not enough[1]. In addition, the connectivity of the underlying hardware requires some additional CX gates when the 4-qubits circuit is transpiled. Nevertheless, our implementation provides a significant improvement in the number of basis gates required for the implementation of the algorithm. In Table 2, we list the number of basis gates required by both implementations for word length 11 using the default optimization level by IBMQ Santiago backend.

An Optimized Implementation The circuit construction above can be improved by sacrificing some freedom in the selection of rotation angles as proposed in [7]. In the circuit diagram given in Figure 7, only the controlled rotation operators are used, where the unitary matrix for symbol a is as follows:

$$\tilde{U}_a = \begin{pmatrix} R_1 & 0 & 0 & 0 \\ 0 & R_2R_1 & 0 & 0 \\ 0 & 0 & R_3R_1 & 0 \\ 0 & 0 & 0 & R_3R_2R_1 \end{pmatrix}$$

Thus, each sub-automaton applies a combination of rotations among three rotations.

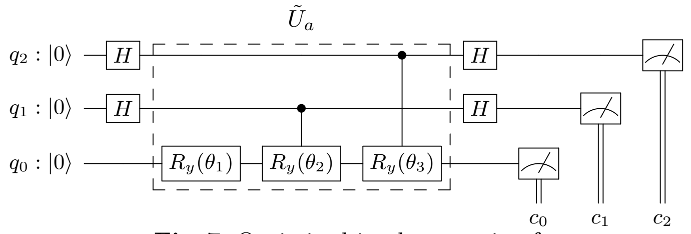


Fig. 7. Optimized implementation for $\texttt{MOD}_{11}$

Upon reading the left end-marker, we have the superposition state $|v_{\mathbb{C}}\rangle$. The block $\tilde{U}_a$ between the Hadamard operators represent the unitary operator corresponding to symbol a. After reading the first a, the new superposition becomes

$$|\tilde{v}_1\rangle = \frac{1}{2}|00\rangle \otimes R_y(\theta_1)|0\rangle + \frac{1}{2}|01\rangle \otimes R_y(\theta_2)R_y(\theta_1)|0\rangle + \frac{1}{2}|10\rangle \otimes R_y(\theta_3)R_y(\theta_1)|0\rangle + \frac{1}{2}|11\rangle \otimes R_y(\theta_3)R_y(\theta_2)R_y(\theta_1)|0\rangle .$$

By letting $\phi_1 = \theta_1$, $\phi_2 = \theta_1 + \theta_2$, $\phi_3 = \theta_1 + \theta_3$, and $\phi_4 = \theta_1 + \theta_2 + \theta_3$, the state $|\tilde{v}_1\rangle$ can be equivalently expressed as

$$|\tilde{v}_1\rangle = \frac{1}{2}|00\rangle \otimes R_y(\phi_1)|0\rangle + \frac{1}{2}|01\rangle \otimes R_y(\phi_2)|0\rangle + \frac{1}{2}|10\rangle \otimes R_y(\phi_3)|0\rangle + \frac{1}{2}|11\rangle \otimes R_y(\phi_4)|0\rangle .$$

Compared to the 3 qubit implementation presented in the previous subsection, this implementation uses less number of gates and especially the number of CX gates is reduced. Furthermore, as no multi-controlled gates are required, the number of CX gates used by the IBMQ Santiago machine even reduces when the default optimization is used. The number of required basis gates for various implementations is given in Table 2 for word length 11.

Table 2. Number of basis gates required by naive and optimized implementations ran on IBMQ Santiago

	Basis Gates			
Implementation	U_1	U_2	U_3	CX
3 Qubits Naive	270	15	121	270
4 Qubits Naive	561	154	114	1471
Optimized	0	4	44	55

We conducted experiments on IBMQ Santiago machine with two different set of values of k, $\{2, 4, 8\}$ and $\{4, 9, 10\}$. A discussion about the choice of the value of k is presented in the next subsection. The results are still far from the ideal as it can be seen in Figure 8. We also plotted the ideal results we got from the simulator. When compared with the naive implementation, we observe that the acceptance probabilities fluctuate until a certain word length but after some point they tend to converge to $1/8$, the probability of selecting a random state.

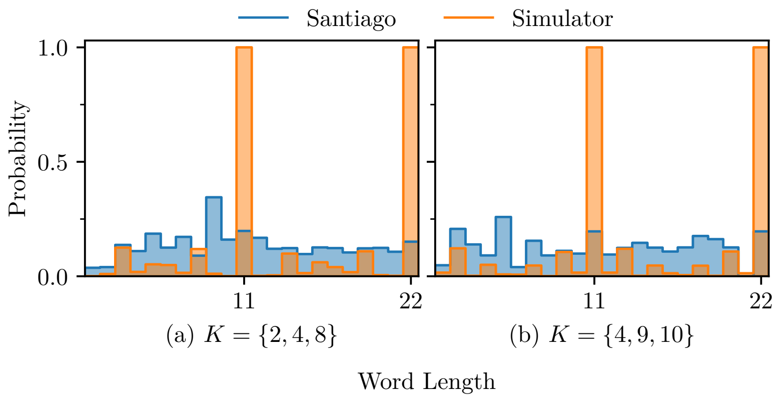


Fig. 8. Acceptance probabilities for $\texttt{MOD}_{11}$ optimized implementation

A Parallel Implementation Recall that in the single qubit implementation, we have a single automaton, but the problem is, that we get arbitrarily large error for nonmember strings and we try to reduce it by running multiple sub-automata in parallel. In this section we provide some experimental results about $\texttt{MOD}_\text{p}$ problem where we simply run each sub-automaton using a single qubit. Although theoretically this approach has no memory advantage, it works better in real devices as no controlled gates are used while still providing a space advantage in terms of number of bits in practice.

Consider the circuit diagram given in Figure 9. Using three qubits, we run three automata in parallel with three different rotation angles.

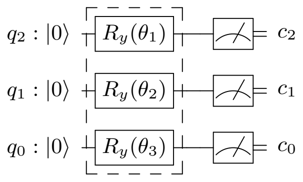


Fig. 9. Parallel $\texttt{MOD}_{11}$ circuit where each qubit implements an MCQFA

In this implementation, the unitary operators corresponding to ¢ and \$ are identity operators. Upon reading the first a, the new state becomes

$$|v_1'\rangle = R_y(\theta_1)\,|0\rangle \otimes R_y(\theta_2)\,|0\rangle \otimes R_y(\theta_3)\,|0\rangle\,.$$

We conducted experiments on IBMQ Santiago machine for $\mathtt{MOD}_{11}$ problem with $K = \{1, 2, 4\}$ and for $\mathtt{MOD}_{31}$ problem with $K = \{8, 12, 26\}$. The results are summarized in Figure 10.

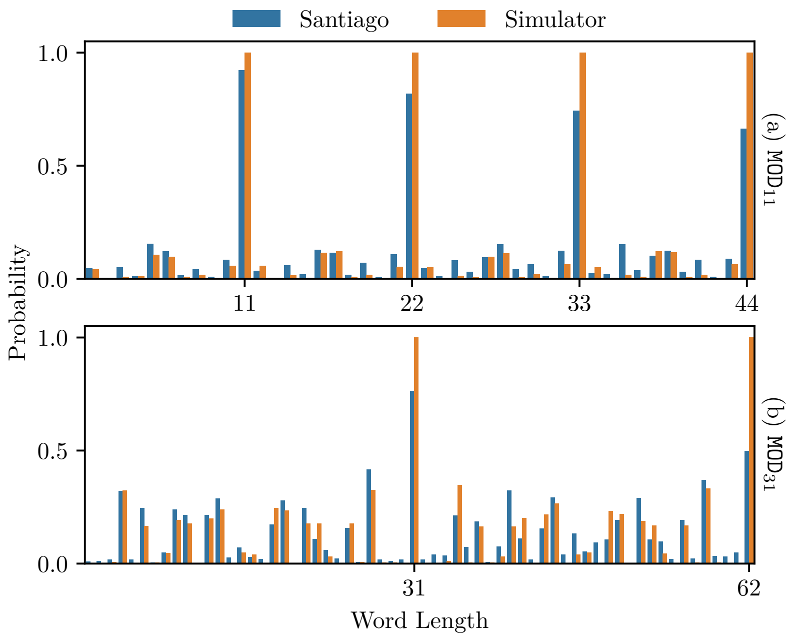


Fig. 10. Acceptance probabilities for $\mathtt{MOD}_{31}$ and $\mathtt{MOD}_{11}$ parallel implementations

From the graphs above we can see that the experimental results on real machines coincide with the simulator outcome especially for small word lengths. The number of required basis gates to implement the parallel implementation is simply three times the length of the word.

Choosing Values of k In [4], the authors consider various values of k for the optimized implementation. One proposal is the cyclic sequences which work well in numerical experiments and give an MCQFA with $\mathcal{O}(\log p)$ states. Another proposal is the AIKPS sequences [2] for which the authors provide a rigorous proof but it requires larger number of states.

We conducted several experiments on the local simulator to see which values of k produce better results for the optimized and parallel MCQFA implementations. We define the maximum error as the highest acceptance probability for a nonmember string and we investigated for which values of k, the maximum error is minimized. Experimental findings are listed below, in Table 3.

Table 3. Standard deviation and mean values for compared circuits

	Min	Max	Mean	Std. Dev.
$\mathtt{MOD}_{11}$ Optimized	0.010	0.080	0.034	0.018
$\mathtt{MOD}_{11}$ Parallel	0.109	0.611	0.270	0.148
$\mathtt{MOD}_{31}$ Parallel	0.319	0.974	0.615	0.167

In the optimized implementation, the maximum error ranges between 0.01 and 0.08 with a standard deviation of 0.018 so it can be concluded that the choice of different values of k does not have a significant impact on the success probabilities, for this specific case. When we move on to parallel implementation, we observe that the maximum error probabilities vary heavily depending on the choice of the value of k. Interquartile range is much larger this time. Furthermore, $\mathtt{MOD}_{31}$ results include error values as high as 0.97 in the third quartile. $\mathtt{MOD}_{11}$ also has some outliers that show far greater error than we would like to work with. With this insight, we picked the values of k accordingly in the parallel implementation, which had an impact on the quality of the results we obtained.

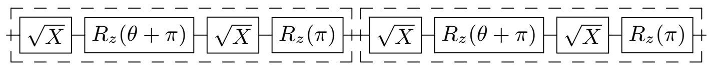
5 Conclusion and Future Work

The goal of this study was to investigate circuit implementations for quantum automata algorithms solving $\mathtt{MOD}_{\mathrm{p}}$ problem. As a way of dealing with the limitations of NISQ devices, we considered different implementation ideas that reduce the number of gates and qubits used. Our findings contribute to the growing field of research on efficient implementations of quantum algorithms using limited memory.

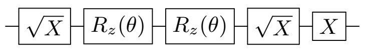
Recently, the basis gates of IBMQ backends were reconfigured as CX, I, R_z, $\sqrt{X}$, and X. As a result, a new methodology should be developed in order to reduce the number of required gates. For instance, the circuit with two consecutive R_y gates that is transpiled by Qiskit is given in Figure 11a. Instead of this design, we propose the implementation given in Figure 11b, which would reduce the number of required basis gates. We use the fact that $R_y(\theta) = \sqrt{X} \cdot R_z(\theta) \cdot \sqrt{X} \cdot X$, hence multiple rotations can be expressed as $R_y^n(\theta) = \sqrt{X} \cdot R_z^n(\theta) \cdot \sqrt{X} \cdot X$.

Acknowledgements

This research project started during QIntern2020 program of QWorld (July-August 2020) and later continued under the QResearch Department of QWorld.



(a) Qiskit transpilation



(b) Alternative implementation

Fig. 11. R_y gate implementations with the new basis set

We thank to anonymous reviewers for their helpful comments and Mārtiņš Kālis for giving a presentation during QIntern2020 program on the details of his master thesis and kindly answering our questions.

Birkan and Nurlu were partially supported by TÜBİTAK scholarship "2205 – Undergraduate Scholarship Program". Yakaryılmaz was partially supported by the ERDF project Nr. 1.1.1.5/19/A/005 "Quantum computers with constant memory".

References

1. Abraham, H., et al.: Qiskit: An open-source framework for quantum computing (2019). https://doi.org/10.5281/zenodo.2562110, doi:10.5281/zenodo.2562110
2. Ajtai, M., Iwaniec, H., Komlós, J., Pintz, J., Szemerédi, E.: Construction of a thin set with small fourier coefficients. Bull. London Math. Soc. **22**(6), 583–590 (1990)
3. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: Strengths, weaknesses and generalizations. In: 39th Annual Symposium on Foundations of Computer Science, FOCS '98. pp. 332–341. IEEE Computer Society (1998)
4. Ambainis, A., Nahimovs, N.: Improved constructions of quantum automata. Theoretical Computer Science **410**(20), 1916–1922 (2009)
5. Ambainis, A., Yakaryilmaz, A.: Automata and quantum computing. CoRR **abs/1507.01988** (2015), http://arxiv.org/abs/1507.01988
6. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical Review A **52**(5), 3457–3467 (1995)
7. Kālis, M.: Kvantu Algoritmu Realizācija Fiziskā Kvantu Datorā (Quantum Algorithm Implementation on a Physical Quantum Computer). Master's thesis, University of Latvia (2018)
8. Mereghetti, C., Palano, B., Cialdi, S., Vento, V., Paris, M.G.A., Olivares, S.: Photonic realization of a quantum finite automaton. Phys. Rev. Research **2**(1), 013089–013103 (2020)
9. Moore, C., Crutchfield, J.P.: Quantum automata and quantum grammars. Theoretical Computer Science **237**(1-2), 275–306 (2000)
10. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press, USA, 10th edn. (2011)
11. Sipser, M.: Introduction to the Theory of Computation. International Thomson Publishing, 1st edn. (1996)
12. Tian, Y., Feng, T., Luo, M., Zheng, S., Zhou1, X.: Experimental demonstration of quantum finite automaton. npj Quantum Inf **5**(56) (2019)